Helsinki University of Technology
Department of Signal Processing and Acoustics
Espoo 2008 Report 5

STUDIES ON HIGH-SPEED HARDWARE IMPLEMENTATION OF CRYPTOGRAPHIC ALGORITHMS

Kimmo Järvinen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Electronics, Communications and Automation for public examination and debate in Auditorium S1 at Helsinki University of Technology (Espoo, Finland) on the 21st of November, 2008, at 12 noon.

Helsinki University of Technology
Faculty of Electronics, Communications and Automation
Department of Signal Processing and Acoustics

Teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Signaalinkäsittelyn ja akustiikan laitos

Distribution:
Helsinki University of Technology
Department of Signal Processing and Acoustics
P.O. Box 3000
FIN-02015 HUT
Tel. +358-9-451 3211
Fax. +358-9-452 3614
E-mail: Mirja.Lemetyinen@hut.fi

ISBN 978-951-22-9589-0 (Printed)
ISBN 978-951-22-9590-6 (Electronic)
ISSN 1797-4267

Multiprint Oy
Espoo 2008

ABSTRACT OF DOCTORAL DISSERTATION	HELSINKI UNIVERSITY OF TECHNOLOGY P.O. BOX 1000, FI-02015 TKK http://www.tkk.fi
Author	
Name of the dissertation	
Manuscript submitted	Manuscript revised
Date of the defence	
Monograph	Article dissertation (summary + original articles)
Faculty Department Field of research Opponent(s) Supervisor Instructor	
Abstract	
Keywords	
ISBN (printed)	ISSN (printed)
ISBN (pdf)	ISSN (pdf)
Language	Number of pages
Publisher	
Print distribution	
The dissertation can be read at http://lib.tkk.fi/Diss/	

VÄITÖSKIRJAN TIIVISTELMÄ	TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi
Tekijä	
Väitöskirjan nimi	
Käsikirjoituksen päivämäärä	Korjatun käsikirjoituksen päivämäärä
Väitöstilaisuuden ajankohta	
Monografia	Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)
Tiedekunta Laitos Tutkimusala Vastaväittäjä(t) Työn valvoja Työn ohjaaja	
Tiivistelmä	
Asiasanat	
ISBN (painettu)	ISSN (painettu)
ISBN (pdf)	ISSN (pdf)
Kieli	Sivumäärä
Julkaisija	
Painetun väitöskirjan jakelu	
Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/	

Acknowledgments

THE RESEARCH WORK for this thesis was carried out in the Department of Signal Processing and Acoustics (formerly, Signal Processing Laboratory), Helsinki University of Technology (TKK). First of all, I would like to thank my supervisor, Prof. Jorma Skyttä, for the support he gave me during these years. He always took care of the ever-increasing university bureaucracy and let me concentrate almost 100 %-ly to my research work, which I greatly appreciate. Sincere thanks go also to Dr. Matti Tommiska for patient guidance in the early phases of this work.

The pre-examiners of this thesis, Dr. Panu Hämäläinen and Dr. Toomas P. Plaks, are greatly acknowledged for insightful comments which, I believe, helped me to improve the manuscript of this thesis significantly.

I had the privilege of being in the Graduate School of Electronics, Telecommunications and Automation (GETA) from 2004 to 2008. I thank former director of GETA, Prof. Iiro Hartimo, current director, Prof. Ari Sihvola, and coordinator, Marja Leppäharju, for making GETA such an exemplary graduate school. Without GETA's support making of this thesis would have been much more difficult, or perhaps impossible.

Signal Processing Laboratory was always a nice place to work. The thanks for that go to everyone who have worked there over the years, but I would like to thank, especially, Juha Forsten, Antti Hämäläinen, Jaakko Kairus, Esa Korpela, Pekka Korpinen, Keijo Länsikunnas, Dr. Jarno Martikainen, Sampo Ojala, Taneli Riihonen, Dr. Matti Rintamäki, Kati Tenhonen, and Dr. Matti Tommiska. Also, I express my gratitude to the laboratory's former secretary, Anne Jääskeläinen, and secretary, Mirja Lemetyinen, because they never hesitated to help me in various practical problems.

GETA, Prof. Graham Jullien, and Prof. Vassil Dimitrov arranged me an opportunity to visit the ATIPS Lab at the University of Calgary, Canada, for three months period in autumn 2005. It is hard to exaggerate the importance of this visit for my thesis work, and I am therefore most grateful for this wonderful opportunity that I had. Especially, I would like to thank Prof. Dimitrov, Prof. Michael Jacobson, Jr., Dr. Laurent Imbert, Dr. Zhun Huang, and Andy Chan, with whom I had the pleasure of working during my short stay in Calgary.

Since June 2008, I have been affiliated with the Department of Information and Computer Science at TKK. I would like to thank everyone there for warmly welcoming me. In particular, I thank my new boss, Prof. Kaisa Nyberg, for

giving me time to finalize this thesis, and Billy Bob Brumley for fruitful cooperation that started already long before I joined the department. He also deserves thanks for commenting the manuscript of this thesis.

The research work was financed by GETA and two projects: GO-SEC (2003–2005), financed by Tekes and several Finnish telecommunication companies, and PLA (2006–2008), financed entirely by Tekes. My doctoral thesis work was financially supported also by the Nokia Foundation, Emil Aaltosen säätiö, Tekniikan edistämissäätiö (TES), and Elektroniikkainsinöörien säätiö, and I greatly appreciate their generosity.

Finally, I would like to thank those who, although without having a direct effect on my research work, have influenced the outcome of this work in many other ways. Therefore, I thank my parents, brothers, and sister for providing me a solid background and always supporting me in everything I have chosen to do in my life. My warmest thanks go to Hanna for love, encouragement, and support she has given me over the years.

Espoo, October 2008

Kimmo Järvinen

Table of Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I. Kimmo Järvinen, Matti Tommiska and Jorma Skyttä, "Comparative Survey of High-Performance Cryptographic Algorithm Implementations on FPGAs," *IEE Proceedings: Information Security*, vol. 152, no. 1, Oct. 2005, pp. 3–12.

II. Kimmo U. Järvinen, Matti T. Tommiska and Jorma O. Skyttä, "A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor," in *Proceedings of the 11th ACM International Symposium on Field-Programmable Gate Arrays, FPGA 2003,* Monterey, California, USA, Feb. 23–25, 2003, pp. 207-215.

III. Kimmo Järvinen, Matti Tommiska and Jorma Skyttä, "A Scalable Architecture for Elliptic Curve Point Multiplication," in *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, FPT 2004*, Brisbane, Queensland, Australia, Dec. 6–8, 2004, pp. 303-306.

IV. Kimmo Järvinen and Jorma Skyttä, "On Parallelization of High-Speed Processors for Elliptic Curve Cryptography," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, Sep. 2008, pp. 1162–1175.

V. Kimmo Järvinen, Juha Forsten and Jorma Skyttä, "FPGA Design of Self-certified Signature Verification on Koblitz Curves," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2007*, Vienna, Austria, Sep. 10–13, 2007, Lecture Notes in Computer Science, vol. 4727, Springer, pp. 256-271.

VI. Kimmo Järvinen and Jorma Skyttä, "Fast Point Multiplication on Koblitz Curves: Parallelization Method and Implementations," *Microprocessors and Microsystems*, in press, 11 pages.

VII. Kimmo U. Järvinen and Jorma O. Skyttä, "High-Speed Elliptic Curve Cryptography Accelerator for Koblitz Curves," in *Proceedings of IEEE*

International Symposium on Field-programmable Custom Computing Machines, FCCM 2008, Stanford, California, USA, Apr. 14–15, 2008, in press, 10 pages.

VIII. Kimmo Järvinen, Juha Forsten and Jorma Skyttä, "Efficient Circuitry for Computing τ-adic Non-Adjacent Form," in *Proceedings of the 13th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2006,* Nice, France, Dec. 10–13, 2006, pp. 232-235.

IX. Billy Bob Brumley and Kimmo Järvinen, "Koblitz Curves and Integer Equivalents of Frobenius Expansions" in *Revised Selected Papers of the 14th Annual Workshop on Selected Areas in Cryptography, SAC 2007*, Ottawa, Ontario, Canada, Aug. 16–17, 2007, Lecture Notes in Computer Science, vol. 4876, Springer, pp. 126-137.

X. V.S. Dimitrov, K.U. Järvinen, M.J. Jacobson, Jr., W.F. Chan, and Z. Huang, "FPGA Implementation of Point Multiplication on Koblitz Curves Using Kleinian Integers," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, Yokohama, Japan, Oct. 10–13, 2006, Lecture Notes in Computer Science, vol. 4249, Springer, pp. 445-459.

XI. Vassil S. Dimitrov, Kimmo U. Järvinen, Michael J. Jacobson, Jr., Wai Fong (Andy) Chan and Zhun Huang, "Provably Sublinear Point Multiplication on Koblitz Curves and Its Hardware Implementation," *IEEE Transactions on Computers*, vol. 57, no. 11, Nov. 2008, pp. 1469–1481.

List of Abbreviations

3DES	Triple-DES
AES	Advanced Encryption Standard
AES-128	AES with a 128-bit key
AES-192	AES with a 192-bit key
AES-256	AES with a 256-bit key
ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ALUT	Adaptive Look-Up Table
ANSI	American National Standards Institute
ASIC	Application Specific Integrated Circuit
CBC	Cipher Block Chaining
CHES	Cryptographic Hardware and Embedded Systems
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CTR	CounTeR
DBNS	Double-Base Number System
DES	Data Encryption Standard
DLP	Discrete Logarithm Problem
DoS	Denial-of-Service
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
DSP	Digital Signal Processing / Processor
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
FAP	Field Arithmetic Processor
FCCM	Field-programmable Custom Computing Machines
FIPS	Federal Information Processing Standard
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
gcd	Greatest Common Divisor
GCM	Galois Counter Mode
GLV	Gallant, Lambert, and Vanstone

GNB	Gaussian Normal Basis
HDL	Hardware Description Language
HECC	Hyper-Elliptic Curve Cryptography
IDEA	International Data Encryption Algorithm
IEEE	Institute of Electrical and Electronics Engineers
IFP	Integer Factorization Problem
JSF	Joint Sparse Form
LAB	Logic Array Block
lsb	Least Significant Bit
LUT	Look-Up Table
MD	Message Digest
msb	Most Significant Bit
NAF	Non-Adjacent Form
NAF_w	Width-w Non-Adjacent Form
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NSA	National Security Agency
ONB	Optimal Normal Basis
PAL	Programmable Array Logic
PLA	Packet Level Authentication / Programmable Logic Array
RACE	Research and development in Advanced Communications technologies in Europe
RAM	Random Access Memory
RFID	Radio Frequency IDentification
RIPEMD	RACE Integrity Primitives Evaluation MD
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RSA	Rivest, Shamir, and Adleman
SHA	Secure Hash Algorithm
SPA	Simple Power Analysis
SPN	Substitution-Permutation Network
SSL	Secure Sockets Layer
VHDL	Very high speed integrated circuit HDL
XOR	eXclusive-OR
τJSF	τ-adic Joint Sparse Form
τNAF	τ-adic Non-Adjacent Form
τNAF_w	Width-w τ-adic Non-Adjacent Form

List of Symbols

$*$	Binary operation
$\otimes$	Bitwise logical and operation (AND)
$\oplus$	Bitwise logical exclusive-or operation (XOR)
$\lfloor\cdot\rfloor$	The largest smaller or equal integer
$\lceil\cdot\rceil$	The smallest larger or equal integer
$(\cdot)^{-1}$	Inverse
$\langle\cdot\rangle$	Bit string notation
$\langle\cdot\rangle_x$	Hexadecimal notation
$a, \ldots, e$	Elements of groups, rings, fields, and vector spaces
a_1, a_2, a_3, a_4, a_6	Curve parameters
$a(x), \ldots, d(x)$	Elements of $\mathbb{F}_{2^m}$ with polynomial basis
$A, \ldots, J$	Temporary variables of point operations in $\mathcal{P}$ and $\mathcal{LD}$
$\mathbf{A}$	Addition
$\mathcal{A}$	Affine coordinates
AddRoundKey$(\cdot,\cdot)$	Round key mixing transformation of the AES
C	Ciphertext
d	Exponent used in decryption
d_N	Number of ones in a multiplication matrix
$d_K(\cdot)$	Decryption function with a key K
D	Multiplier digit-size
D_i	Data after round i in block ciphers
$\deg(\cdot)$	Degree of
e	Exponent used in encryption
$e_K(\cdot)$	Encryption function with a key K
E	Elliptic curve
E_{a_2}	Koblitz curve ($a_2 \in \{0, 1\}$)
$E(\mathbb{F}_q)$	Group of points on an elliptic curve E over $\mathbb{F}_q$
$\mathbb{F}$	Field
$\mathbb{F}_q$	Finite field with q elements
$\mathbb{F}_q^*$	Multiplicative group of $\mathbb{F}_q$
$\mathbb{F}_p$	Prime field
$\mathbb{F}_{2^m}$	Binary field
$\mathbb{F}_q[x]$	Ring of polynomials over $\mathbb{F}_q$
f	Clock frequency

$f(x)$	Binary polynomial of $x \in \mathbb{F}$
$f(\cdot, \cdot)$	Round function of a block cipher
$F(\cdot, \cdot)$	F-function of normal basis multiplication
g	Generator of $\mathbb{F}_q$
G	Group
$gcd(\cdot, \cdot)$	Greatest common divisor
h	Hash
$H(\cdot)$	Hamming weight, the number of nonzero terms
$H_n(\cdot)$	Joint Hamming weight of n expansions
HASH$(\cdot)$	Cryptographic hash algorithm
$i, j, n, r, \ldots, u$	Positive integers
$\mathbf{I}$	Inversion
l	Latency
ℓ	Bit length
k	Positive (random) integer
$\mathbf{k}$	An array of binary expansions
K	Key
K_i	Round key for the ith round
K_e	Encryption key
K_d	Decryption key
KEYSCHEDULE$(\cdot)$	Key schedule of the AES
$\mathcal{LD}$	López-Dahab coordinates
m	Dimension of an extension field
M	Message, plain text
$\mathbf{M}$	Multiplication
MIXCOLUMNS$(\cdot)$	Column transformation of the AES
Nr	Number of rounds
ord$(\cdot)$	Order of
$\mathcal{O}$	The point at infinity
p	Prime number
p	The number of parallel operations
$p(x)$	Irreducible polynomial
P	Point on an elliptic curve, the base point
$\mathcal{P}$	Projective coordinates
q	Order of a finite field
Q	Point on an elliptic curve, the result point
R	Ring
ROUNDKEYS	Round keys of the AES
$\mathbf{S}$	Squaring
$S(\cdot)$	S-box
SHIFTROWS$(\cdot)$	Row permutation transformation of the AES
STATE	State in the AES
SUBBYTES$(\cdot)$	Substitution transformation of the AES
t	Computation time
T	Throughput
$T_i(\cdot)$	T-box i
$\mathbf{T}_N$	Multiplication matrix of normal basis
V	Vector space
w	Window size
x, y	Coordinates of a point in $\mathcal{A}$

X, Y, Z	Coordinates of a point in $\mathcal{P}$ or $\mathcal{LD}$
α	Normal element of $\mathbb{F}_{2^m}$
δ, μ	Auxiliary variables for Koblitz curves
Δ	Temporary variable of composite field inversion
λ	Temporary variable of point operations in $\mathcal{A}$
ξ	Element of a multiplication matrix $\mathbf{T}_N$
σ	Field isomorphism
τ	Complex root
$\varphi(\cdot)$	Euler's totient function
$\phi(\cdot)$	Frobenius endomorphism
ψ, ω	Coefficients of irreducible polynomials

Chapter 1

Introduction

INFORMATION SECURITY is a fundamental requirement for an operational information society. Although issues considered as information security, such as privacy of communication and reliable authentication, have been important throughout history, developments in digital computing and information technology have set new requirements and challenges for them. The importance of information security has grown because new technologies have made accessing and misusing confidential information easier and more profitable. Hence, information security, previously considered mostly by militaries and governments, has become an issue having relevance even for an average person living in a modern information society because of its significance in commerce, telecommunication, *etc.*

Cryptography, the art and science of hiding data, plays a central role in achieving information security. It has become a fundamental part of communication and commercial applications in the Internet as well as in many other digital applications. Cryptography is deployed with cryptographic algorithms, mathematical functions for hiding messages and retrieving hidden messages. This thesis studies and proposes methods to efficiently implement certain cryptographic algorithms.

1.1 Motivation for the Thesis

The motivation for this thesis, as perhaps for all engineering, arises from the classic utilitarian view which was well captured by Cilardo *et al.* [67] who stated,

> "A mathematical construction with practical applications, such as a cryptographic algorithm, has no real interest, in an engineering sense, as long as methods for feasible implementation are not available."

In other words, efficient implementation techniques for cryptographic algorithms are necessary in order to use them in practice.

Applications using cryptographic algorithms often set very tight requirements for implementations. Low-cost applications, such as mobile hand-held

devices, sensor networks, or smart cards, to name a few, often set strict constraints on available resources, such as logic, memory, or power [90]. On the other hand, high-speed applications, such as high-speed network servers, require very high computation speeds in order to prevent cryptography from becoming the bottleneck [105, 126]. Hence, designing implementations fulfilling the requirements is often a challenging task and implementing cryptographic algorithms has been widely studied in academia.

Naturally, general-purpose microprocessors can be used for implementing cryptographic algorithms but, in that case, the execution of computationally demanding cryptographic algorithms easily becomes the limiting factor for overall performance [63]. Hence, dedicated hardware implementations are commonly required to offload cryptographic algorithms away from the microprocessor [63]. Reprogrammable hardware has established itself as one of the most attractive platforms for cryptographic algorithms because of the combination of speed and flexibility [293]. This thesis concentrates on Field Programmable Gate Arrays (FPGAs) which are commonly considered very feasible implementation platforms among implementors of cryptographic algorithms [293]. However, a majority of the ideas and architectures presented in this thesis can be applied also to Application Specific Integrated Circuits (ASICs), and some even for software implementations.

1.2 Scope of the Thesis

The thesis concentrates on a secret-key cryptographic algorithm, Advanced Encryption Standard (AES) [206], and public-key cryptographic algorithms based on arithmetic on elliptic curves, Elliptic Curve Cryptography (ECC) [146, 194]. AES has become a true *de facto* algorithm for secret-key cryptography and it is used in countless applications worldwide ranging from RFID tags [95] and sensor networks [120] to heavily-loaded servers [105] and optical networks [126]. Because of the very wide use of AES, it is justified to focus this study solely to AES. ECC is in many ways superior to traditional public-key algorithms, such as the widely-used RSA [239], because it offers similar levels of security with considerably shorter keys [32, 164] and faster performance [88, 251]. In the recent years, ECC has gained popularity also in practical systems and it is used, *e.g.*, in Windows Vista [160], OpenSSL [218], and German biometric passports [43, 44], to name a few. The National Security Agency (NSA) of the United States is also strongly promoting the use of ECC [210]. ECC has thus become a mainstream method for public-key cryptography and its use is increasing. Hence, the selection of ECC is justified.

Because this thesis considers both secret-key and public-key cryptographic algorithms, the results also enable designing cryptosystems where a secret key is first shared by two parties with the public-key algorithm, *i.e.*, with ECC, and, then, the actual encryption is performed with the faster secret-key algorithm, *i.e.*, with AES. As mentioned, both AES and ECC are algorithms which are used in numerous practical applications and they are likely to remain popular also in the foreseeable future. Consequently, most of the findings of this thesis can be directly applied in practice.

Requirements for cryptographic implementations are diverse, but the primary requirement considered in this thesis is speed of computation. Before a

cryptographic algorithm can be adopted to a practical application, there must exist ways to implement it so that it matches the speed requirements of the application. For instance, if traffic in a high-speed network is encrypted with cryptographic algorithms, the implementations must be able to compute the algorithms with throughputs matching the throughput of the network, or otherwise they become the bottleneck for the entire communication. An example of an application that sets very high requirements for computation speed is the Packet Level Authentication (PLA) communication scheme [46, 47], which was the main target application, especially, for the implementations presented in **V** and **VII**. Other examples include heavily-loaded e-commerce servers [105] and optical network switches [126].

Most publications of this thesis concern ECC. The emphasis is on methods allowing increased use of parallelism in ECC operations, or more specifically in elliptic curve point multiplication, the principal operation of ECC. Point multiplication is recursive in nature and it is thus hard to parallelize. Many of the contributions focus on a class of elliptic curves, called Koblitz curves [148], which offer faster computation, but require certain conversions for integers. Koblitz curves have attained a lot of interest in theoretical cryptography research and they are listed in major ECC standards, such as [52, 205], but, prior to this thesis, they have attained only limited interest in the community implementing cryptographic algorithms in hardware.

The focus of this thesis is on the algorithmic level and in methods allowing efficient implementation. This thesis does not discuss higher level aspects, such as cryptographic protocols. Also lower levels, such as gate or transistor levels, are not discussed in this thesis. All contributions address the problem of achieving fast computation of cryptographic algorithms efficiently. The scope of the thesis can be described as finding answers to the following questions:

— Which algorithms are the most suitable for hardware (or FPGA) implementations and how could they be improved in order to increase their suitability?

— How should these algorithms be implemented in order to provide the best possible results?

Author's other publications on topics related to cryptography, in addition to the publications of this thesis, include an automated design generator for ECC [137], studies of cryptographic hash algorithms, MD5 and SHA-1 [138, 139], and a recent work on fast decompressions in ECC [41].

1.3 Contributions of the Thesis

This thesis has contributed to several areas of research in cryptographic implementations. More specifically, the contributions enable faster implementations of AES and ECC than those that were available previously, and they thus provide improvements predominantly to applications requiring high speed implementations. The following lists the main contributions of this thesis. More detailed discussions are given in the publications and in Ch. 7.

The AES-related contributions (**I**, **II**) include

— a review of existing FPGA-based implementations of widely-used secret-key cryptographic algorithms including AES and hash algorithms (**I**) and a review complementing and updating this review (Ch. 5),

— a study illuminating the relationship between achieved throughput and consumed area in AES implementations also when they use embedded memory of FPGAs (**I**), and

— a high-speed implementation of AES which was the fastest FPGA-based implementation at the time of publication (**II**).

The most significant contributions are those considering ECC (**III**–**XI**) and they include

— a review of existing ECC implementations on FPGAs and ASICs (Ch. 6),

— an implementation optimized for the structure of FPGAs (**III**),

— a study on the effects of parallelism in ECC implementations and tools for evaluating and optimizing parallelism in these implementations (**IV**),

— a method that splits point multiplications on Koblitz curves for parallel processors and FPGA implementations utilizing this method (**IV**),

— an FPGA design for computing signature verifications, the most demanding operations of the PLA, with very high speed (**V**),

— improvements to precomputation algorithms used in these signature verifications (**V**),

— a method utilizing parallelism and interleaved point operations on Koblitz curves (**VI**),

— implementations of the above method (**VI**) and an optimized implementation which further improves on this by utilizing similarity of certain point multiplication algorithms (**VII**); these implementations achieve the fastest point multiplications currently available in the literature,

— hardware-optimized algorithms for integer conversions related to Koblitz curves and their implementations (**VIII**, **IX**), and

— an introduction of a new multiple-base expansion to Koblitz curves and its FPGA implementations (**X**, **XI**).

These contributions increase knowledge of implementing AES and ECC in hardware, and more specifically in FPGAs. The most substantial results are those that provide methods to implement ECC with increased amounts of parallelism, because they result in very fast FPGA implementations. Especially, the results related to Koblitz curves dramatically improve existing solutions. The contributions of this thesis could be directly applied in various practical systems and, as a consequence, they also have significant practical relevance.

1.4 Structure of the Thesis

This thesis comprises an overview and eleven publications which are appended in the end. The remainder of this overview is structured as follows:

Ch. 2 presents an overview of cryptography.

Ch. 3 discusses implementation platforms of cryptographic algorithms.

Ch. 4 introduces preliminaries of finite fields and their implementations.

Ch. 5 discusses AES and its implementations.

Ch. 6 discusses ECC and its implementations.

Ch. 7 summarizes the results of the thesis.

Ch. 8 draws conclusions.

Ch. 2 and 3 give general descriptions of cryptography and cryptographic hardware, whereas Ch. 4–6 present detailed descriptions of both theory and practice of the algorithms discussed in the publications. Ch. 7 and 8 present the results and conclude their importance for the research field.

1.5 Summary of Publications and Author's Contribution

The following provides short summaries of the eleven publications of this thesis. The author had a major role in research and writing in all these publications, and the author's contribution in every particular publication is specified in the end of each summary.

Publication I The article presents a comparative survey of cryptographic algorithm implementations on FPGAs. The study concentrates on high-speed implementations of AES, International Data Encryption Algorithm (IDEA), MD5, and SHA-1, but other secret-key algorithms and hash algorithms are studied shortly as well. The survey addressed the well-recognized problem of comparing implementations using embedded memory to memory-free implementations. It was concluded that FPGAs suit well for cryptographic algorithms.

The author is responsible for all research and writing, except for the study of IDEA in Sec. 2.2 which was written by Dr. Matti Tommiska who also made improvement suggestions to the rest of the paper. Prof. Jorma Skyttä supervised the research.

Publication II The paper describes an implementation of AES. The implementation provided the highest reported throughput at the time of publication, and the most important contribution of the paper was thus the speed of the implementation. Composite fields were utilized in order to increase the efficiency of the implementation. Xilinx Virtex FPGAs were used for demonstration but the architecture itself is generic and applies to both FPGAs and ASICs.

All hardware designs are due to the author. The author wrote all of the paper; except Secs. 1, 6, and 7. Dr. Matti Tommiska provided extensive support in

both research work and writing of the paper. He also introduced the possibility of using composite fields to the author and wrote the above mentioned sections. Prof. Jorma Skyttä supervised the research.

Publication III The paper presents an implementation of ECC optimized for the structure of Xilinx Virtex FPGAs. The implementation was designed so that it could be easily generated with an automatic VHDL (Very high speed integrated circuit Hardware Description Language) generator which was presented in [137]. The achieved point multiplication times were among the fastest published results at the time of publication, and even the fastest ones for certain parameters.

The author designed and implemented all hardware architectures. The author also wrote the entire paper while Dr. Matti Tommiska supervised the work and made many observations in both research work and writing of the paper. Prof. Jorma Skyttä supervised the research.

Publication IV The article discusses parallelization of ECC processors and describes a generic processor architecture which is used in analyzing effects of parallelism. The article presents a comprehensive study of existing parallelization techniques and, most importantly, presents a novel parallelization technique using parallel processors which results in considerable reductions in latency, especially, on Koblitz curves but also on general curves with certain presuppositions. A number of FPGA-based implementations are presented for both general and Koblitz curves.

The author is responsible for all theory, hardware implementations, and the entire writing of the manuscript. Prof. Jorma Skyttä supervised the research and presented comments on the manuscript.

Publication V The paper presents an accelerator designed specifically for verifying self-certified signatures on Koblitz curves. This operation acts as the bottleneck in the PLA. It was concluded that up to 166,000 signatures can be verified in second with a single FPGA by using massive parallelism. The paper also presents certain improvements to precomputations involved in verifications. The main contribution of the paper is that it shows that considerable improvements in the number of operations per second are achievable by tolerating slightly longer computation times for single operations.

The author is responsible for everything presented in the paper and he wrote the entire paper. Juha Forsten helped in verifying the designs on an FPGA and he also made certain improvements to the manuscript. Prof. Jorma Skyttä supervised the research.

Publication VI The article discusses ECC on Koblitz curves. It introduces a new parallelization method which achieves optimal multiplier utilization by interleaving successive operations. The method has several advantages over the best previously presented methods (the ones presented in **IV**). First, the achievable critical path is shorter; second, the method enables implementing both general and Koblitz curves efficiently in the same hardware; third, the method increases the attractiveness of using Koblitz curves; and, fourth, efficient

implementations can be designed by using the method as demonstrated in the paper on FPGAs.

The author is alone responsible for all theory, implementations, and writing. Prof. Jorma Skyttä supervised the research.

Publication VII The paper presents an implementation designed specifically for Koblitz curves. The implementation can compute point multiplications and sums of up to three point multiplications and, hence, it supports all ECC operations required in the PLA, confer **V** which supports only verifications. The implementation utilizes the common structure of the algorithms and specific processing units optimized for different parts of the algorithms, especially, the method presented in **VI**. Both fast computation time for single operations and high throughput are achieved with the implementation and, to the author's knowledge, it outperforms all other implementations available in the literature.

The author is responsible for everything presented in the paper and all writing. Prof. Jorma Skyttä supervised the research.

Publication VIII The paper describes an architecture for integer conversions related to Koblitz curves. To the author's knowledge, it was (and still is) the only publication presenting such converters. The converter is compact and fast and, hence, it has significance in hardware realization of ECC on Koblitz curves. Prior to the publication of the paper, a general-purpose processor was used for the conversions in all publications. The contributions of the paper include modifications to conversion algorithms, which make them more suitable for hardware, and an efficient converter architecture.

The author is responsible for all research, implementations, and writing in the publication. Juha Forsten helped in testing the converter and gave some comments during the writing of the paper. Prof. Jorma Skyttä supervised the research.

Publication IX This paper discusses conversions related to Koblitz curves too, but to the other direction than in **VIII**. Such conversions have significance, *e.g.*, in digital signature algorithms. The benefit compared to conversions discussed in **VIII** is that conversions can be computed in parallel with other operations. The paper also presents efficient hardware implementations of the conversion, and such converters were not presented previously in the literature.

The author designed and implemented the hardware architecture. Theoretical work and ideas are due to Billy Bob Brumley. The author made certain suggestions on how to efficiently compute the conversions and wrote Sec. 4. Brumley wrote the rest of the paper.

Publication X The paper presents multiple-base expansions, analogous to the Double-Base Number System (DBNS) [79, 80], and they are useful in ECC on Koblitz curves. Two types of representations, one with two bases and the other with three, were suggested and it was proven that the number of terms in the three-base expansions is sublinear in the bit length of an integer. The same was conjectured for the two-base expansions based on extensive numerical evidence. The practical feasibility of the two-base method was shown by

designing an FPGA implementation. The implementation used the state-of-the-art algorithms and achieved high speed with moderate area requirements. The implementation was the fastest published ECC implementation at the time of publication. The suggested expansions were the first multiple-base expansions presented for Koblitz curves which were shown to be feasible in practice.

The author designed and implemented the entire hardware implementation, except the multiple-base expansion converter which was designed and implemented by Dr. Zhun Huang. Theoretical work and ideas on multiple-base expansions are due to Prof. Vassil Dimitrov and Prof. Michael Jacobson, Jr. The software implementation and numerical results were made by Andy Chan. The author is responsible for the writing of Sec. 4, excluding Sec. 4.3, and a part of Sec. 5. The co-authors wrote the rest of the paper.

Publication XI The article extends **X**. It discusses multiple-base expansions with more details and provides additional numerical evidence supporting feasibility of the expansions. A new section discussing inherent parallelism available in the expansions is included together with a description of an FPGA implementation utilizing this parallelism.

The most important new part, Sec. 5, was written solely by the author, and the author is alone responsible for all research related to that section. Otherwise, the responsibilities were the same as in **X**.

Chapter 2

Overview of Cryptography

Cryptography, the art and science of keeping messages secure, has a long history which can be traced back to ancient Egypt some 4000 year ago [189]. Undoubtedly, cryptanalysis, the art and science of revealing messages hidden by means of cryptography, has an equally long history. Together cryptography and cryptanalysis are called cryptology. When the history of cryptology is discussed, it must be emphasized that historical cryptology has little common with modern cryptology which started in the 20th century, and which is the topic of this thesis.

First, some fundamental terminology is introduced. The message, which is to be kept in secret, is referred to as plaintext. The process of hiding its content is called encryption and the encrypted message is referred to as ciphertext. The process of receiving the content of plaintext back from ciphertext is decryption. A cryptographic algorithm is the mathematical function used for encrypting and decrypting messages. A modern cryptographic algorithm always includes a key. A cryptographic algorithm, plaintexts, ciphertexts, and keys are referred to as cryptosystem.

Fig. 2.1 shows the basic communication model where Alice (A) and Bob (B) are communicating over an unsecured channel, and Eve (E) is trying to eavesdrop their communication. In order to prevent Eve from obtaining the content of a message, M, Alice uses a cryptographic algorithm to turn M into

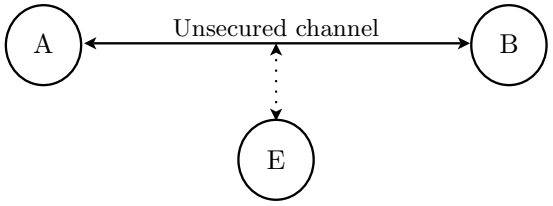

Figure 2.1: Communication model

a ciphertext, C, *i.e.* she performs

$$C = e_K(M) \tag{2.1}$$

where $e_K(\cdot)$ is the encryption function with the key, K. Alice sends C over the unsecured channel to Bob who performs

$$M = d_K(C) \tag{2.2}$$

where $d_K(\cdot)$ is the decryption function. Thus, Bob has received M but Eve, who does not have K, is unable to get M from C unless she can break the cryptographic algorithm. [122, 258]

Let K_e and K_d be the keys used in encryption and decryption, respectively. Cryptographic algorithms where K_e can be computed from K_d, and vice versa, are called secret-key[1] cryptographic algorithms. The secrecy of keys is essential for the security of secret-key cryptographic algorithms because if Eve obtains K_e, she can easily compute K_d. Thus, Alice and Bob must exchange keys by using a secured channel which makes key distribution one of the most difficult problems in many applications. The most efficient way to attack a secret-key algorithm should be an exhaustive search through the key space. Even in that case, a secret-key algorithm is weak if it is realistic to assume that it could be possible to build a machine that can perform an exhaustive search in some reasonable time with realistic cost. [258]

Secret-key algorithms were the only option to deploy cryptography until the mid-1970s when Whitfield Diffie and Martin E. Hellman published their landmark paper, "*New Directions in Cryptography*" [76]. They introduced the concept of public-key cryptography[2] where K_e can be easily computed from K_d but K_d cannot be computed from K_e (in any reasonable time). Thus, K_e can be made public. Communication between Alice and Bob operates so that Bob publishes his K_e. Alice uses it to encrypt M and sends C over the unsecured channel. Only Bob has access to K_d because it cannot be computed from K_e, and Bob is thus the only one able to decrypt C. Public-key cryptographic algorithms are based on hard problems where "hard" means that a problem is considered impossible to solve with any computational resources available currently or in the (near) future. In proportion, a problem is "weak" if there are methods for solving the problem with some amount of computational resources available at the time although acquiring such resources might be unreachable for all but the richest governments or corporations. Thus, the hardness of a problem is open to interpretation. [258]

Moreover, the required level of security depends on confidentiality of the data being encrypted. Hence, an adequate level of security may be achieved with weak algorithms. For example, Data Encryption Standard (DES) was used until late-1990s by some corporations although it was commonly considered weak already in the 1980s [164]. As a rule of thumb, if the cost of breaking a cryptosystem exceeds the value of encrypted data, then the cryptosystem can be considered adequately secure [258]. However, cryptography alone does not provide information security, but it is rather a set of techniques [189]. In addition to cryptographic algorithms, security relies also on many other aspects. In

[1] Terms such as private-key or symmetric(-key) are also commonly used in the literature.
[2] Also called as asymmetric(-key) cryptography.

fact, failures in practical cryptosystems, such as automatic teller machines, are commonly due to implementation and management errors [11]. Hence, in order to make a system secure, all sides of an application must be secured: protocols, algorithms, implementations, *etc.*, [164] and security considerations should be an organic part in designing applications where security is an issue [231]. The above does not diminish the importance of secure cryptographic algorithms because they have an essential role in building a secure system in the first place. Security is always compromised if a severe flaw is found in a cryptographic algorithm.

Secret-key cryptography is considered in more detail in Sec. 2.1 and public-key cryptography is discussed in Sec. 2.2. Cryptographic hash algorithms are one-way functions which have a great significance in many cryptosystems, and they are considered in Sec. 2.3.

2.1 Secret-key Cryptography

Secret-key cryptographic algorithms are categorized into either stream ciphers or block ciphers based on how they manipulate data. A stream cipher handles data one symbol, typically a bit, at a time whereas a block cipher encrypts data in fixed-length blocks. Most cryptographic algorithms used today are block ciphers and stream ciphers are used predominately in situations where transmission errors are probable and implementation resources are limited [189], such as mobile communication devices. Stream ciphers are not considered further in this thesis.

Modern block ciphers are usually constructed using a Feistel structure or a Substitution-Permutation Network (SPN), *e.g.*, DES uses a Feistel structure and AES is an SPN. In both cases, encryption and decryption are performed by iteratively applying a round function, $D_i = f(D_{i-1}, K_i)$, where D_{i-1} is the data at iteration $i-1$ and K_i is the round key for iteration i. Round keys are derived from the key, K, with a routine called key schedule. The round function, $f(\cdot, \cdot)$, includes substitutions performed with the so-called S-boxes which are predefined tables, permutations that rearrange the input data using fixed rules, and round key mixing which is typically performed with an exclusive-or (XOR). [274]

An enormous amount of different block ciphers have been proposed and are in use in different applications. Probably the two most important ones, DES and AES, are discussed in Secs. 2.1.1 and 2.1.2, respectively.

2.1.1 Data Encryption Standard (DES)

In 1972, National Bureau of Standards (NBS) of the United States, nowadays National Institute of Standards and Technology (NIST), initiated a program for computer security which included the development of an encryption standard. After initial problems in finding a candidate that would fulfill the requirements set for the standard, they received a promising candidate called Lucifer from IBM. Lucifer was evaluated and revised by the NSA before its adaptation as a Federal Information Processing Standard (FIPS) in 1976. The standard [203] was published in January 1977. The development process and introduction of DES was perhaps the most important landmark in the history of academic research on cryptosystems. It was the first time in history when a high-security

cryptographic algorithm became available to everyone and, hence, it marked the beginning of wide-scale academic research on cryptography. [258]

It had become obvious that the key length of DES was too short to be secure against exhaustive key search and, thus, a strengthened variant called Triple-DES (3DES) was included into the final reaffirmation of the standard in 1999 [204]. DES was withdrawn officially in 2005 and it has been replaced by AES. DES is not considered further in this thesis because of its decreasing importance in contemporary cryptosystems.

2.1.2 Advanced Encryption Standard (AES)

Because it had become obvious that DES had exceeded the end of its reasonable lifetime, NIST began the process to replace DES in January 1997. An open competition was announced for algorithms in September of the same year. The requirements for the algorithm were 128-bit block size, support for keylengths of 128, 192, and 256 bits, and royalty free availability. Altogether 15 candidate algorithms were submitted to the competition. After the first round, five finalist algorithms were selected; namely, Mars, RC6, Rijndael, Serpent, and Twofish. [45, 211]

Criteria for the selection of the winner included security, performance, and algorithm characteristics. All finalists were stated to be secure enough to be selected as AES, and there were no known intellectual property issues for any of the algorithms. Thus, performance was the key issue which finally decided the winner. Performance was evaluated on various platforms including 8-bit embedded processors, 32-bit Pentium processors (the most important), Digital Signal Processors (DSPs), FPGAs, and ASICs. [45, 211]

In October 2000, NIST announced that Rijndael was selected as the new AES algorithm. The reasons why Rijndael was selected were its strong performance on basically all platforms and the ease of hardware implementation [45, 211]. Rijndael was also the most popular choice in straw polls at the final AES conference, and its selection received an approving reception in the community [45]. Rijndael was officially adopted as the AES on December 6, 2001 [206]. Henceforth, AES always refers to Rijndael in this thesis[3]. Details of the algorithm are not considered here because they are discussed thoroughly in Sec. 5.1.

2.2 Public-key Cryptography

As mentioned, public-key cryptography was invented by Diffie and Hellman in the mid-1970s. The main idea that they presented in [76] was that different keys K_e and K_d could be used for encryption and decryption so that it is easy to derive K_e from K_d but it would be infeasible to find K_d from K_e. Diffie and Hellman suggested using discrete exponentiation as a method for obtaining K_e. Because of its fundamental nature, their key exchange method is reviewed in Sec. 2.2.1 and it is followed by short descriptions of certain other public-key algorithms currently used in many practical applications. Only basic versions

[3]To be precise, there is a small difference between AES and Rijndael: Rijndael accepts block sizes of 128, 192, and 256 bits, whereas AES defines only a fixed block size of 128 bits. In this thesis, both terms refer to the official AES unless stated otherwise.

are presented and they may be insecure against certain attacks, such as the-man-in-the-middle attack; see [258], for example.

Public-key cryptography is used in key exchange, encryption, and digital signatures. Key exchange and encryption are self-explanatory: Key exchange permits two parties to safely agree a shared secret key over an unsecured channel and encryption allows data to be encrypted without sharing a secret key. Digital signatures can be seen as digital counterparts for handwritten signatures and they provide unforgeable proofs of authorship and authenticity as well as non-repudiation.

2.2.1 Diffie-Hellman Key Exchange

First, Alice selects a random integer $K_{d,A}$ from an interval $[1, p-1]$ where p is a prime. Then, she computes and publishes

$$K_{e,A} = g^{K_{d,A}} \pmod{p} \tag{2.3}$$

where g is a fixed element of a finite field $\mathbb{F}_p$. Finite fields will be discussed in more detail in Ch. 4. Alice's private-key and public-key are $K_{d,A}$ and $K_{e,A}$, respectively. Similarly, Bob selects $K_{d,B}$ and computes and publishes $K_{e,B}$. Now, Alice and Bob get a common secret key K, which can be used in a secret-key cryptographic algorithm, by computing

$$K = g^{K_{d,A}K_{d,B}} = (K_{e,B})^{K_{d,A}} = (K_{e,A})^{K_{d,B}} \pmod{p}. \tag{2.4}$$

In order to compute K, Eve, who knows only $K_{e,A}$ and $K_{e,B}$, must compute either $K_{d,A}$ from $K_{e,A}$ or $K_{d,B}$ from $K_{e,B}$, *i.e.*, she must solve the following Discrete Logarithm Problem (DLP)

$$K_d = \log_g K_e \pmod{p}. \tag{2.5}$$

DLP is believed to be a hard problem if g and p are chosen carefully. Thus, it is impossible for Eve to obtain K. [76]

2.2.2 RSA Cryptosystems

Soon after Diffie and Hellman's paper, Ronald L. Rivest, Adi Shamir, and Leonard Adleman published a cryptosystem [239] which is now known, after its inventors, as the RSA cryptosystem. A key pair is generated in RSA so that one randomly selects two primes p_1 and p_2 of the same bit length $\ell/2$ where ℓ is referred to as the security parameter [122]. Then, one computes

$$n = p_1 p_2 \quad \text{and} \quad \varphi(n) = (p_1 - 1)(p_2 - 1), \tag{2.6}$$

and selects an integer e from the interval $[1, \varphi(n)]$ so that $gcd(e, \varphi(n)) = 1$, *i.e.*, the greatest common divisor (gcd) of e and $\varphi(n)$ is one[4]. Finally, one computes $d = e^{-1} \pmod{\varphi(n)}$. The public-key is $K_e = (n, e)$ and the private-key is $K_d = d$. RSA encryption and signature schemes are based on the fact that

$$M^{ed} \equiv M \pmod{n} \tag{2.7}$$

[4] $\varphi(n)$ is Euler's totient function which gives the number of positive integers smaller than n and coprime to n [274].

for all integers M. [239]

The RSA encryption operates so that Alice computes

$$C = M^{e_B} \pmod{n_B}, \tag{2.8}$$

by using Bob's public-key $K_{e,B} = (n_B, e_B)$. She sends C over the unsecured channel to Bob who decrypts the message by computing

$$M = C^{d_B} \pmod{n_B} \tag{2.9}$$

where d_B is Bob's private-key $K_{d,B}$. [239]

In the RSA signature scheme, K_d is used in signing (encryption) and K_e in verification (decryption), but this does not change the fact that K_d must be private while K_e is public. Bob signs a message M as follows. First, he computes a hash, $h = \textsc{Hash}(M)$, of M by using a cryptographic hash algorithm which are discussed in more detail in Sec. 2.3. At this point, it suffices to state that h is a fixed-length bit string which can be considered as the "fingerprint" of M. Second, Bob generates the signature s by computing

$$s = h^{d_B} \pmod{n_B} \tag{2.10}$$

and sends s and M to Alice. Alice verifies that M was signed by Bob so that, first, she computes h, as Bob did, and, second, she computes

$$h' = s^{e_B} \pmod{n_B}. \tag{2.11}$$

Alice accepts the signature if $h = h'$, else she rejects it. [239]

RSA is believed to depend on the difficulty of Integer Factorization Problem (IFP) because Eve is unable to compute d from e if she does not know $\varphi(n)$. Furthermore, she is unable to find $\varphi(n)$ unless she can factor p_1 and p_2 out from n, which requires solving the IFP. The best known method for factorization is currently the number field sieve factoring [18].

2.2.3 ElGamal Cryptosystems

Taher ElGamal [92] expanded the idea of Diffie and Hellman by introducing encryption and signature schemes based on the DLP in 1985. The ElGamal encryption scheme is considered next. Let private-keys and public-keys be as in Diffie-Hellman in Sec. 2.2.1. Then, Alice encrypts a plaintext, M, with the following equations by using a random integer $k \in [1, p-1]$ and Bob's public-key, $K_{e,B}$:

$$\begin{aligned} C_1 &= g^k \pmod{p} \\ C_2 &= M K_{e,B}^k \pmod{p}. \end{aligned} \tag{2.12}$$

Then, Alice sends (C_1, C_2) to Bob who decrypts the ciphertext by computing

$$M = C_2 C_1^{-K_{d,B}} \pmod{p}. \tag{2.13}$$

2.2.4 Digital Signature Algorithm (DSA)

Digital Signature Algorithm (DSA), first presented by David W. Kravitz in [155], is included in standards by American National Standards Institute (ANSI) [8], NIST [205], and Institute of Electrical and Electronics Engineers (IEEE) [132, 133], among others. DSA specifies how to sign and verify messages with cryptographic signatures. Consider a case where Bob signs a message, M, and Alice verifies the signature.

The domain parameters are agreed so that p is a prime, n is a prime divisor of $p-1$ and g is a generator of $\mathbb{F}_n$; see Ch. 4. First, Bob generates a key pair ($K_{d,B}$,$K_{e,B}$) by selecting a random integer $K_{d,B} \in [1, n-1]$ and by computing

$$K_{e,B} = g^{K_{d,B}} \pmod{p}. \tag{2.14}$$

Bob signs a message, M, by computing its hash, $h = \textsc{Hash}(M)$. The signature (s, r) is computed as follows:

$$\begin{aligned} r &= (g^k \bmod p) \pmod{n} \\ s &= k^{-1}(h + rK_{d,B}) \pmod{n} \end{aligned} \tag{2.15}$$

where $k \in [1, n-1]$ is a random integer and k^{-1} is its multiplicative inverse in $\mathbb{F}_n$; see Ch. 4. Bob published the domain parameters and $K_{e,B}$ and sends the message, M, and the signature, (r, s), to Alice. [205]

When Alice receives a message, M, and an attached signature, (r, s), which is claimed to be signed by Bob, she retrieves the domain parameters and Bob's public key $K_{e,B}$. She verifies the signature by computing $h = \textsc{Hash}(M)$ and the following formulae:

$$\begin{aligned} u_1 &= s^{-1}h \pmod{n} \\ u_2 &= s^{-1}r \pmod{n} \\ v &= (g^{u_1} K_{e,B}^{u_2} \bmod p) \pmod{n}. \end{aligned} \tag{2.16}$$

Alice accepts the signature if $v = r$, else she rejects it. [205]

If Bob indeed generated the signature Alice received and the message was not tampered after signing, s received by Alice is as in (2.15) which gives

$$k = s^{-1}(h + rK_{d,B}) = u_1 + u_2 K_{d,B} \pmod{n}. \tag{2.17}$$

Thus, $g^{u_1} K_{e,B}^{u_2} = g^{u_1} g^{K_{d,B}^{u_2}} = g^{u_1 + u_2 K_{d,B}} = g^k \pmod{n}$, and $v = r$ which shows that Bob generated the signature and M was not tampered after the generation. This proof was adapted from [140].

2.2.5 Elliptic Curve Cryptosystems

The use of elliptic curves in cryptography was independently proposed by Neil Koblitz [146] and Victor Miller [194] in 1985. Since then, large amounts of work has been done on ECC in academia and increasingly also in industry. The security of ECC is based on the hardness of a problem called Elliptic Curve Discrete Logarithm Problem (ECDLP) which is an elliptic curve analogue of the DLP. ECC operates analogously to the cryptosystems based on the DLP with the exception that exponentiations are replaced by elliptic curve operations.

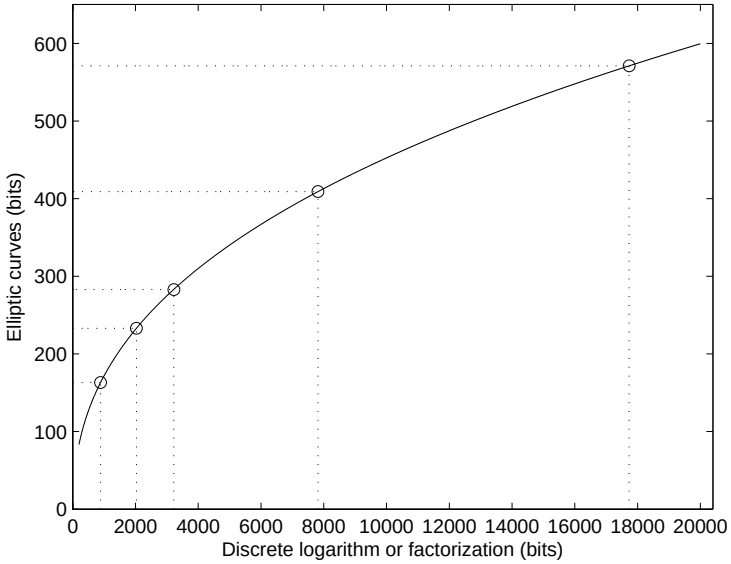

Figure 2.2: Approximative key sizes of ECC and public-key cryptosystems based on the DLP or the IFP for a similar level of security [32]. The corresponding key sizes for the curves specified by NIST [205] are plotted with dotted lines.

Elliptic curve operations are not considered here because a comprehensive review is presented in Ch. 6.

Standardization has great significance in ECC because there is a multitude of parameters to choose from, they have a large impact on design decisions, and implementations using different parameters usually fail to cooperate [67]. ECC is included in the following major standards: ANSI X9.62 [8] and X9.63 [9], IEEE 1363 [132, 133], FIPS 186-2 [205] (NIST), ISO/IEC 15946-1/2/3/4 [135] and 18033-2 [134], and SECG [51, 52].The DSA standard [205] also includes an elliptic curve variant of DSA called Elliptic Curve Digital Signature Algorithm (ECDSA) which is equivalent with the algorithm described in Sec. 2.2.4 with the exception that the exponentiations are replaced by elliptic curve operations [132, 205]. A thorough presentation of ECDSA is given in [140].

The main reason why ECC has attained so much interest during the past twenty years is depicted in Fig. 2.2. The curve in Fig. 2.2 shows an approximation of the required key sizes of ECC vs. conventional public-key cryptosystems, *i.e.*, schemes based on the DLP and the IFP, for a similar level of security. Fig. 2.2 is based on a formula from [32]. Fig. 2.2 also plots the corresponding key sizes for the five elliptic curves recommended by NIST [205], henceforth called the NIST curves. When the ECC key sizes are 163, 233, 283, 409, or 571 bits, the key sizes of cryptosystems based on the DLP or the IFP required for a similar level of security are approximately 890, 2030, 3220, 7820, or 17730 bits. These values are naturally highly approximative and they are based on the assumption that there are no sub-exponential algorithms for the ECDLP [32]. Further estimates on key sizes with equivalent security levels are provided in [164] where they favor ECC even more.

Technical reasons supporting the use of RSA instead of ECC are basically nonexistent. The reasons that have prevented ECC from becoming the primary public-key cryptosystems are most probably the tradition of using RSA and the fact that mathematics related to ECC are more difficult to understand. The field of ECC also contains many patents (many of them owned by Certicom Corporation [53], the leading company in ECC solutions) whose validity has not yet expired (confer RSA) which may have hindered the use of ECC. However, ECC has already been included in many widely-used standards and adapted to many major applications.

2.2.6 On Difficulties of the Hard Problems

Although public-key cryptosystems ultimately rely on conjectures that hard problems are indeed impossible to solve, several studies have supported such conjectures. In 1991, RSA Laboratories published a challenge where cash prizes up to $200,000 (US dollars) were offered for successful factorizations of products of two primes (n in Sec. 2.2.2). The challenge was set in order to motivate researchers and, thus, increase knowledge on the difficulty of the IFP [246]. The largest challenge solved is 640-bit n that was factored in November 2005 [246]. Nowadays, the smallest bit length of n recommended for use in practice is 1024 bits. The challenge ended in 2007 when RSA Laboratories stated that enough knowledge on the cryptographic strength of the IFP exists making the challenge irrelevant [246].

In 1997, Certicom Corporation launched a challenge for ECC, similar to the RSA challenge. The largest problem solved thus far is ECDLP with a 109-bit key which was solved in April 2004 [50]. They used 2600 computers running 17 months (roughly equivalent to one Athlon XP 3200+ running non-stop for 1200 years) [50]. When the key size is 163 bits, the smallest size recommended by NIST [205], the ECDLP is roughly 100,000,000 times harder [50]. In 2001, a model presented in [164] predicted with various presumed parameters[5] that 163-bit ECC will remain secure up to year 2021. More pessimistic parameters which assumed considerable advances in methods for solving the ECDLP gave year 2011 [164], but such advances have not yet been presented. A recent academic security analysis [192] estimated that manufacturing and running ASICs that could solve the ECDLP with a 163-bit key in one year would cost $2,200,000,000, half of which is for power consumption. Hence, the 163-bit ECDLP is likely to remain unsolvable for several years to come.

2.3 Cryptographic Hash Algorithms

Cryptographic hash algorithms play a central role in many modern cryptosystems because they can provide data integrity. A hash algorithm produces a hash[6] for an input message. The hash acts as a "fingerprint" of the message. This fingerprint can be used in ensuring integrity of a message because if the message changes, so does its hash. Thus, a user can check the integrity of a

[5] Such as, DES was secure up to 1982, computing power doubles every 18 months (Moore's law), budget doubles every 10 years, and no new cryptanalytic methods are developed for the ECDLP.

[6] Also commonly called message digest (MD), hash-code, hash-result, hash-value, *etc.*

message by computing a hash and comparing it to a previously computed hash. If the hashes are different, someone has edited the message. [189, 274]

Let $\textsc{Hash}(\cdot)$ be a cryptographic hash algorithm. Then the hash, h, of a message, M, is given by

$$h = \textsc{Hash}(M) \tag{2.18}$$

where M is a binary string with an arbitrary length and h has a fixed length. Two principal properties of a hash algorithm are compression and ease of computation [189]. Compression means that the function maps an input of an arbitrary length to an output of a relatively small fixed length. Ease of computation simply means that it is computationally easy to derive h from M.

In order to a hash algorithm to be secure, it must be resistant against the following three problems [274].

— It must be computationally infeasible to find a preimage, *i.e.*, given a hash, h, find M such that $h = \textsc{Hash}(M)$. If a preimage is hard to find, the function is said to be preimage resistant.

— It must be hard to find a second input M' which has the same hash with any specific input M. If it is hard to find $M' \neq M$ such that $\textsc{Hash}(M') = \textsc{Hash}(M)$, the hash algorithm is said to be second preimage resistant.

— The function must be collision resistant, *i.e.*, it must be hard to find two arbitrary inputs M and M', such that $M \neq M'$, for which $\textsc{Hash}(M) = \textsc{Hash}(M')$.

If the first and second problem are infeasible to solve, a hash algorithm is referred to as a one-way function [189]. If any of the three problems is "easy" to solve, a hash algorithm is broken in cryptographical sense. However, in some applications where cryptographic hash algorithms are used, it suffices that hash algorithms are resistant only against certain problems, *e.g.*, when used in a one-way password file, the only requirement is that the hash algorithm is preimage resistant [189].

Hash algorithms may involve a key, K, as well. Hash algorithms with a key are referred to as keyed hash algorithms whereas algorithms without a key are called unkeyed hash algorithms. Unkeyed hash algorithms can be viewed as a special case of keyed hash algorithms where the key is constant. If an unkeyed hash algorithm is used, the hashes must be protected so that they cannot be altered but, when a keyed hash algorithm is used with a secret key, the hashes can be public or transmitted over an unsecured channel without sacrificing the integrity of a message [274]. Unkeyed hash algorithms are commonly used in manipulation detection codes and keyed hash algorithms in message authentication codes [189].

Cryptographic hash algorithms commonly used in practical applications include MD algorithms proposed by Ronald L. Rivest of which MD5 [238] is the latest, Secure Hash Algorithms (SHA) [207] recommended by the NIST, RIPEMD[7] algorithms of which RIPEMD-160 and its extensions [81] are the newest, and Whirlpool [21]. The author has published FPGA implementations of MD5 [139] and a combined implementation of MD5 and SHA-1 [138].

[7] Research and Development in Advanced Communications Technologies in Europe (RACE) Integrity Primitives Evaluation Message Digest

Several cryptographic hash algorithms have been compromised recently so that collisions can be found "easily". MD5 and SHA-1 are among these algorithms. Thus, these hash algorithms should be used only in applications which do not require collision resistivity. [286, 287]

The SHA algorithms will be replaced with an algorithm developed through an open competition similar to AES. The call for new hash algorithms was published in November 2007 in [208] and the new algorithm is scheduled to be selected in 2012.

Chapter 3

Hardware Implementation of Cryptographic Algorithms

NATURALLY, in order to use cryptographic algorithms in practice they need to be implemented. This chapter presents the most common platforms on which cryptographic algorithms are implemented and discusses their suitability. The emphasis is on FPGAs because they are used in the publications.

Designing a good cryptographic implementation is a challenging task. There are at least the following requirements for implementations of cryptographic algorithms and security systems in general:

Speed An implementation should be fast enough to ensure that execution of cryptographic algorithms does not slow down a system significantly. Because computation of cryptographic algorithms dominates in computational complexity of many protocols, the need for fast implementations is evident [231]. Achieving fast performance is often a difficult task and it has attained considerable amount of interest in both industry and academia. This requirement often yields a need for hardware acceleration because many studies have shown that software implementations cannot achieve required performance levels with reasonable costs [118].

Resources Many environments set tight constraints to resources available for implementations of cryptographic algorithms [231]. If resources are low on the whole, only a small portion of those few resources is usually devoted for cryptography which complicates implementing high-security cryptographic algorithms. The aforementioned constraints may include power consumption, amount of available logic or memory, *etc.* Arguably, the importance of this requirement is increasing because of the emerging use of cryptographic algorithms in various low-cost applications [90], such as smart cards or mobile hand-held devices.

Costs Use of cryptographic algorithms should not increase the total cost of a product significantly. This requirement sets strict constraints for designers, especially, if other requirements, such as speed, are considered

too. This is naturally an important requirement for any product but its importance is apparent in low-price consumer products.

Invisibility Implementations of cryptographic algorithms and other security measures should be invisible, transparent, and easy to use for an end user. In other words, they should not cause any considerable harm for the user. The reason for this is that, if they do, it is likely that the algorithms will not be used or they will be misused in a way that compromises security. A classical example is an exhaustive use of complex passwords which can even weaken the system because users commit passwords to paper if they cannot remember them.

Exactness Designs must implement algorithms exactly according to the specifications, *i.e.*, cryptographic algorithms are in general intolerant for errors caused by optimizations. In many applications, such as Digital Signal Processing (DSP), it is possible to optimize implementations by allowing small errors caused by a reduced computation accuracy, for instance. However, cryptographic algorithms usually fail to operate correctly if even one bit is incorrect. On the other hand, optimizations inside the algorithm are possible as long as they do not change the output; see Sec. 5.2.2, for example.

Security An implementation should not leak any information which can potentially compromise security. Although this first seems as a trivial requirement, it is—possibly together with an incompetent user—one of the most important issues affecting overall security. Actually, it has been stated that it is more likely that a real world cryptosystem is broken because of the weaknesses of implementations rather than the weaknesses of cryptographic algorithms [294].

As this thesis is about hardware implementation of cryptographic algorithms, it is of interest to examine how the above requirements relate to hardware implementations. Invisibility requirement seldom has any direct impact on hardware implementations because the user interface is usually software engineer's work rather than hardware engineer's. Even hardware designers must indirectly take the end user's comfort into account through other requirements. All other requirements need to be considered by a hardware designer and the importance of a particular requirement depends on implementation platforms and applications.

3.1 Implementation Platforms

A coarse categorization for implementations, which has already been used above, is to divide them to software and hardware implementations. A software implementation is an implementation running on a general-purpose processor; such as Intel Core-2 Duo processors. The role of an implementor is to write software for a processor by using a programming language. These implementations are therefore referred to as software implementations although the processor itself is hardware. In a hardware implementation the implementor designs the hardware which performs the desired function.

Reprogrammable logic refers to digital logic devices whose functions can be reprogrammed throughout the device [281]. The place of reprogrammable logic in the aforementioned categorization is problematic because it can be categorized to either category on a sound basis. On one hand, an implementor does not design the actual hardware but produces a "software", *i.e.*, a programming file which programs the hardware to implement the desired function. On the other hand, the way in which an implementor works is almost identical to hardware design process and all decisions (s)he makes during the process are closer to implementing hardware than software. Besides, people working on reprogrammable logic in industry or academia almost always have a hardware background. In this thesis, reprogrammable logic is considered hardware. The implementations presented in the publications are all designed primarily for FPGAs. However, the ideas used in them apply usually to ASIC-based implementations, too.

General-purpose processors, ASICs, and certain low-cost environments are discussed in the following sections concentrating on their use for cryptographic algorithms. The prime target devices of this thesis, FPGAs, are discussed in more detail Sec. 3.2.

3.1.1 General-Purpose Processors

General-purpose processors execute a program which is a series of instructions. Instructions supported by a processor are referred to as an instruction set. Instruction sets usually include instructions for integer arithmetic, boolean operations, data handling, and control flow as well as, *e.g.*, for floating point arithmetic. Programmer writes a program, either in assembly or high-level languages, such as C, which is then compiled into instructions. The performance of a program depends on programmer's skill, the instruction set provided by a processor, and the efficiency of hardware.

The advantages of general-purpose processors are easy programming and updating and the fact that the same code is usually easy to use in different processors. Typical disadvantages are slow performance and high power consumption. These disadvantages emphasize if a cryptosystem is implemented using processors with instructions sets that do not naturally support operations required in cryptographic algorithms. For instance, instruction sets of microprocessors usually support integer and floating point arithmetic and boolean operations, whereas cryptographic algorithms often use more "exotic" arithmetic such as finite fields.

Academics—and more recently also processor manufacturers—have recognized the problems caused by limited instructions, and instruction set extensions for cryptography are gaining interest [89]. Such extensions have been proposed, *e.g.*, in [89, 112, 279]. On the commercial side, better support for cryptography has been included at least in Sun Microsystems' new UltraSPARC T2 microprocessors which include specific cryptographic accelerator blocks supporting various cryptographic algorithms including (3)DES, AES, SHA-1, MD5, RSA, and ECC [276]. Such instruction set extensions can provide considerable performance increases and, thus, make general-purpose microprocessors more feasible platforms for cryptography.

3.1.2 Application Specific Integrated Circuits (ASICs)

As the name suggests, ASICs are integrated circuits designed for a specific application. Contrary to general-purpose circuits, such as microprocessors, ASICs can perform only functions that they are designed to perform. In ASICs, a design is implemented directly on silicon and, hence, it cannot be changed after the chip is manufactured. The resulted implementation can be highly optimized in terms of speed, area, and power consumption, but the inability to change the design after manufacturing is a serious disadvantage. Modern ASIC processes are also very expensive and time consuming and, hence, they are nowadays used predominantly in large volume consumer products, such as mobile phones.

Traditionally cryptographic algorithms were always implemented as ASICs whereas, nowadays, a large majority of practical cryptosystems are undoubtedly software implementations running on general-purpose microprocessors. However, ASICs still have significant importance for cryptographic implementations in environments where microprocessors, or even FPGAs, cannot be used, *e.g.*, in environments requiring very low power consumption or high encryption speeds.

3.1.3 Low-Cost Environments

The use of cryptographic algorithms is increasing in low-cost environments, such as smart cards, radio frequency identification (RFID) tags, *etc.* [90]. The reason is that they often handle sensitive data, such as health-monitoring, anti-counterfeit, or biometric data, which yields a need for strong encryption [90]. Although implementations for low-cost environments use the same techniques that are used in implementations targeting to high speed, *i.e.*, ASICs, microprocessors, or FPGAs, the requirements set by these environments differ greatly from the ones discussed previously. As a consequence, low-cost environments are discussed as a separate topic. Low-cost implementations require difficult trade-offs between security, cost, and performance [90]. Besides, they are used in the most insecure environments [231] which makes implementations more vulnerable for physical attacks, as will be discussed in Sec. 3.4. Hence, cryptography in low-cost environments is an important and difficult research area.

Modern cryptographic algorithms, such as AES, have been developed so that they achieve good performance in software. This may not be a problem for smart cards because their processors have become powerful enough to run cryptographic algorithms originally designed for high-end microprocessors [229]. However, this sets challenges for designing efficient implementations for certain low-cost environments, such as RFID tags [90]. Hence, several block ciphers developed specifically for low-cost applications have been proposed recently including DESXL (a low cost variant of DES) [163], Hight [127], and Present [33]. Implementing public-key cryptography is costlier but ECC has been shown to be a viable solution even for low-cost environments [23, 116]. A recent survey of the field of low-cost cryptography is given by Eisenbarth *et al.* in [90].

3.2 Field Programmable Gate Arrays (FPGAs)

Implementations on general-purpose processors are flexible in the sense that programs are easily written and updated by using high-level programming languages. As a downside, they are slow and consume considerable amounts of

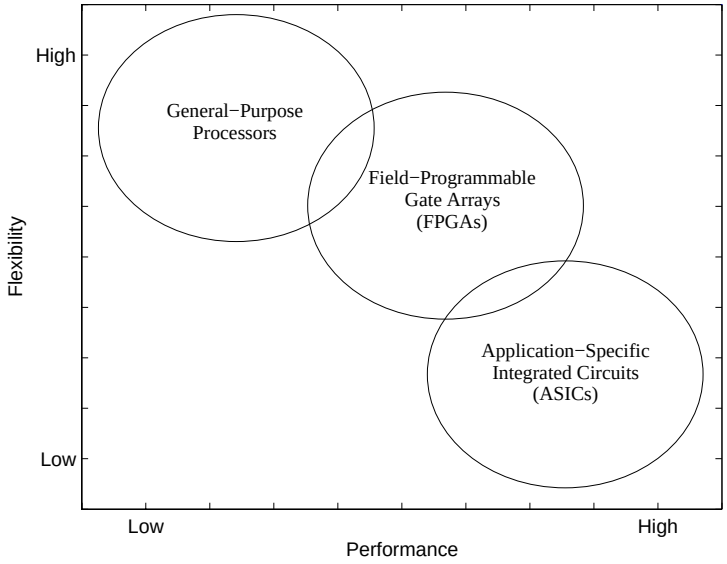

Figure 3.1: A performance-flexibility comparison of general-purpose processors, ASICs, and FPGAs [34, 281]. Performance can be understood as pure speed or power efficiency, or as their combination (throughput/Watt).

power which practically rule general-purpose processors out from many applications. ASICs, on the other hand, are less flexible but offer high performance and low power consumption. Thus, there is a significant gap between general-purpose processors and ASICs [34]. Reconfigurable logic fills this gap and—in an ideal case—combines the best of both; namely, the speed of ASICs to the flexibility of general-purpose processors [281]. This is illustrated in the performance-flexibility graph of Fig. 3.1 [34, 281].

Reconfigurable logic includes a wide scope of programmable devices including PLA (Programmable Logic Array), PAL (Programmable Array Logic), CPLD (Complex Programmable Logic Device), and FPGA. However, FPGAs are the only ones which have any major interest from the cryptography point-of-view and, henceforth, the terms reconfigurable logic and device always refer to FPGAs in this thesis. This presentation concentrates on FPGAs manufactured by the two leading vendors: Altera and Xilinx. Other FPGA manufacturers include Achronix, Actel, Atmel, Cypress, Lattice, and QuickLogic.

FPGAs consists of reconfigurable functional units, interconnections, and interface. Reconfigurable functional units are used for implementing the logic needed in a design and they are connected with the reconfigurable interconnections. Reconfigurable interfacing is used for communication with the rest of a system. As a rule of thumb, the more reconfigurable an architecture is the slower it is and the more power it consumes. [280]

Reconfigurability depends on the size of reconfigurable units referred to as granularity. An architecture where reconfigurable functional units are small is called fine-grained and an architecture where they are large is coarse-grained. A fine-grained functional unit typically operates on a small number of bits and acts

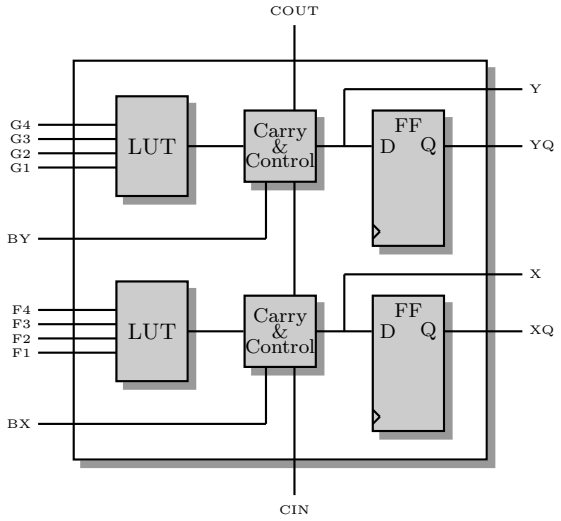

Figure 3.2: A simplified presentation of a Virtex-E/II slice (adapted from [299, 302]).

as a Look-Up Table (LUT) with a small number of inputs and a single bit output [280]. An example of a coarse-grained functional unit is an Arithmetic Logic Unit (ALU) [280] and, thus, coarse-grained architectures are usually better in applications which are intensive on arithmetic [281]. The granularity of reconfigurable interconnections also varies in different reprogrammable architectures so that fine-grained interconnections allow each bit to be connected separately whereas coarse-grained interconnections connect several bits at a time [280]. Most modern commercial FPGAs use fine-grained architectures where reconfigurable functional units are typically 4-to-1-bit LUTs which are arranged in clusters [280].

Programming an FPGA differs considerably from general-purpose processors because a general-purpose processor program is stored into a separate memory as instructions, whereas programming of an FPGA directly implements logic functions and interconnections inside the device [290].

3.2.1 Modern FPGAs and Recent Trends

In Xilinx Virtex-E and Virtex-II FPGAs, which are used as implementation platforms in **I**–**III** and **X**, the functional units are 4-to-1-bit LUTs which are arranged in clusters called slices and Configurable Logic Blocks (CLBs). A CLB comprises four slices which consists of two 4-to-1-bit LUTs, two registers, and carry and control logic [299, 302]. Fig. 3.2 presents a Xilinx Virtex slice. The LUTs can be configured also as a 16-bit RAM (Random Access Memory) [299, 302] or 16-bit shift register [302]. The Altera Stratix II architecture, used in **IV**–**IX** and **XI**, differs from the Virtex architecture so that the clusters called Logic Array Blocks (LABs) consists of eight subblocks named Adaptive Logic Modules (ALMs) which fracture into two Adaptive LUTs (ALUTs), two registers, and carry and control logic [6]. ALUTs are flexible so that they can implement up to a 7-to-1-bit LUT [6]. Virtex-E, Virtex-II, and Stratix II devices all contain

considerable amounts of embedded memory [6, 299, 302].

Modern FPGAs typically contain hardwired logic units specialized for certain commonly used tasks, such as DSP [280]. These units provide improvements in performance and power consumption in those specialized tasks but they are useless in many applications where these operations are not directly needed [280]. Virtex-II has hardwired 18×18-bit multipliers [302] and Stratix II has more flexible DSP blocks which are useful in various DSP-related tasks [6], for example.

In 2005, Todman *et al.* [280] listed main trends in reconfigurable architectures as follows. First, there is a trend to increase granularity of logic units in order to reduce the amount of interconnections, *e.g.*, Stratix II 7-input LUT [6]. Second, the number of specific embedded function units, *e.g.*, DSP blocks and embedded memories, is increasing. Third, the use of soft cores, and especially soft core processors, is growing. A soft core is implemented using reprogrammable logic by using synthesizable function provided by a vendor [280]. Although they are slower and require more area than hard cores which are hardwired onto the silicon, they provide upgradeability and they are easy to integrate into the rest of a system [280]. Xilinx offers a soft core processor called MicroBlaze [300] and Altera calls their newest soft core processor Nios II [5].

Since Todman's predictions, Xilinx has launched Virtex-5 devices [303] and Altera has introduced Stratix III family FPGAs [7]. Xilinx has increased the granularity of Virtex-5 by introducing 5-to-1-bit and 6-to-2-bit LUTs and Altera's Stratix III is still using up to 7-to-1-bit ALUTs. The number of specialized embedded function units has continued growing in the new device families, and also Xilinx has introduced more flexible DSP blocks [303]. Both Xilinx and Altera are still supporting their soft core processors. Hence, the development has obviously had the direction predicted in [280]. There is also an evident direction which began by the introduction of Virtex-4 and which has continued in Virtex-5 and Stratix III as well; namely, the vendors introduce sub-families tailored for certain types of applications. Virtex-4 has sub-families LX, SX, and FX for logic applications, DSP applications, and embedded platform applications, respectively [301]. Virtex-5 also has an additional sub-family specialized in advanced serial connectivity [303]. Stratix III has a balanced sub-family L for mainstream logic applications, a memory and multiplier rich sub-family E for data-centric applications, and a sub-family GX with high-speed connectivity for high bandwidth applications [7].

3.2.2 FPGA Design Flow

A design flow of FPGA-based implementations is presented in Fig. 3.3. The flow begins from design specifications and ends to a working design in an FPGA. The most laborious part is the writing of Hardware Description Language (HDL). VHDL was used as an HDL for all designs of this thesis. The functionality of the HDL is verified by functional simulations. The verified HDL is mapped to a gate-level netlist with logic synthesis. Placement in the place & route process dedicates operations to specific reconfigurable functional units and routing connects these units by using reconfigurable interconnections. The place & route is constraint-driven. A place & route of a complex design is computationally demanding and often takes several hours with a modern desktop computer. The place & route returns a programming bitfile and a timing model description of

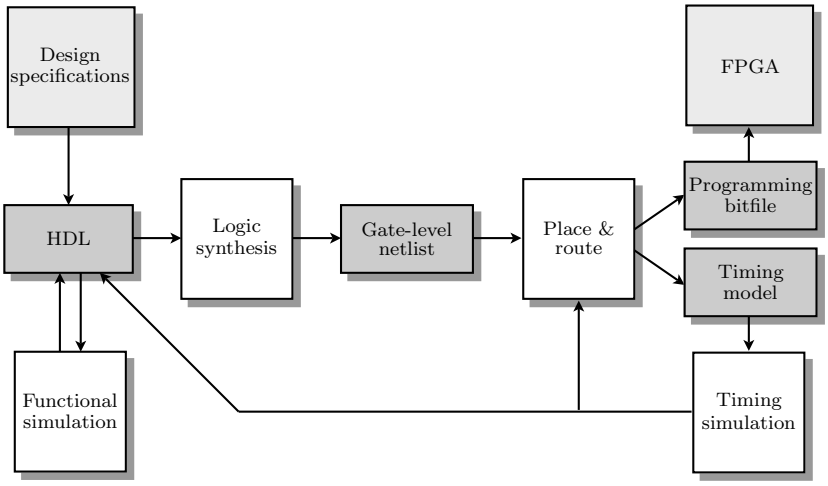

Figure 3.3: FPGA design flow

the result. The timing model is verified to fulfill timing requirements set in the specifications with timing simulations. Finally, the FPGA is programmed with the programming file.

The entire design flow can be performed within design softwares provided by FPGA vendors. Logic synthesis programs are included in the design softwares but several third party tools are available, too. Simulation software is often provided by a third party because only simple simulators are included in the design softwares. Place & route tools are always provided by the FPGA vendor as well as software used in programming the FPGA.

As mentioned, writing HDL is the most laborious part. Automatic HDL generators, which automatically map specifications to HDL, can considerably reduce time required in this step. Typically, such generators can produce competitive results compared to hand-optimized versions only if they are realized for a small and well-constrained set of specifications, *e.g.*, for one cryptographic algorithm with specific parameters. The author developed a simple generator for ECC in [137].

3.2.3 Cryptographic Algorithms in FPGAs

Cryptographic algorithms often dominate in computational costs of security protocols which yields a need for acceleration; for example, over one half of the processing time in Secure Sockets Layer (SSL) is consumed in cryptographic algorithms [231]. Acceleration is typically performed by delegating the most expensive operations to an accelerator while the rest of the application is implemented in the main system which is usually a general-purpose processor [281]. Consider DSA presented in Sec. 2.2.4 as an example. The most demanding task in DSA is the modular exponentiation and, hence, DSA can be accelerated by delegating exponentiations to an accelerator while the main processor performs other operations concurrently. Because this thesis predominantly considers implementing cryptographic algorithms using FPGAs, the following assumes that

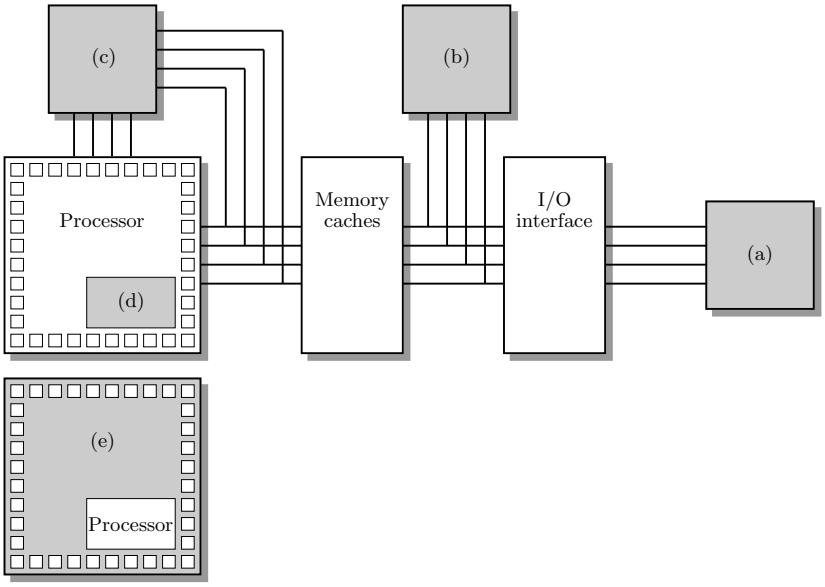

Figure 3.4: Classification of reconfigurable systems [69, 280]; (a) external stand-alone processing unit, (b) attached processing unit, (c) coprocessor, (d) reconfigurable functional unit, and (e) processor embedded in reconfigurable logic.

the accelerators are implemented with reconfigurable logic. Many of the principles are, however, valid also for other implementation platforms, such as ASICs.

Fig. 3.4 shows the classification of reconfigurable systems. The first four classes were presented by Compton and Hauck in [69] and their classification was complemented by Todman *et al.* in [280] with the class where the processor is embedded in reconfigurable logic. The closer the reconfigurable unit is to the control processor the faster the communication between the unit and the processor is but, as a downside, the amount of reconfigurability is usually smaller and more control is required [280]. The external stand-alone processing unit favors applications which are computationally intensive but require little communication and the reconfigurable functional unit, *e.g.*, reconfigurable instructions inside a processor, is ideal for small but frequently used tasks [69]. The attached processing unit and coprocessor classes can be seen as tradeoffs between these two extremes. The last class, where the processor is embedded in reconfigurable logic, is currently emerging in many applications because the growth in FPGA resources has enabled implementing entire systems within one FPGA. This approach combines many of the advantages of the other four classes but requires a large (and expensive) device. The embedded processor can be either hard core, such as PowerPC in Xilinx Virtex-4 FX [301], or soft core, such as Nios II in Altera FPGAs [5]. Notice that all classes except (d) can be implemented with a stand-alone FPGA. Thus, the implementations of this thesis suit for all other classes except (d).

FPGAs have several advantages in cryptographic applications [91, 293]:

Algorithm agility It is possible to easily switch from an operation to another. This has significant importance in cryptographic applications be-

cause modern cryptosystems must often support many encryption algorithms. It is also possible to delete broken algorithms and introduce new algorithms into the system.

Algorithm upload It is possible to easily update devices which are already in use. This is, again, important for cryptographic applications because parameters of the current cryptosystem may need to be changed, *e.g.*, key length needs to be increased or an algorithm needs to be changed as standards expire, *e.g.*, DES needs to be changed to AES.

Architecture efficiency Significantly more efficient implementations can be designed with fixed parameters in certain cases. Reprogrammability of FPGAs allows designing optimized implementations for all parameters separately because the device can be reprogrammed when parameters are changed whereas an ASIC design must support all parameters and such optimizations are out of reach. One example is the finite field multiplication in polynomial basis; see Ch. 4. If an irreducible polynomial is fixed, reductions required in multiplication can be hardwired resulting in faster and smaller multipliers. If the design must support arbitrary irreducible polynomials, such optimizations cannot be made.

Resource efficiency Many protocols are hybrid in the sense that a public-key algorithm is used in the beginning for key exchange after which the actual communication is encrypted using a secret-key algorithm. Because these algorithms are not used simultaneously, they can be switched on-the-fly by reprogramming the device which leads to a considerable reduction in required resources.

Algorithm modification Parts of standardized algorithms, *e.g.*, S-boxes, may need to be modified, and such modifications are not a problem because of reprogrammability.

Throughput Considerable increase in throughput is achievable with FPGAs when compared to general-purpose processors as shown in an extensive number of publications; see, *e.g.*, [59, 106, 107, 280].

Cost efficiency FPGA-based projects are typically significantly cheaper than ASIC-based projects when the products are targeted to low-to-medium sized markets. The main reasons are shorter time-to-market and lower initial costs. ASICs become cheaper when the production quantities rise because production of a single chip is cheaper after the often very large startup cost.

It has been stated based on the above reasonings that cryptographic algorithms are prime candidates for FPGA-based implementations [281]. The suitability of FPGAs for cryptography has been pointed out also in other publications, *e.g.*, [34, 280].

Although embedded function units, such as DSP blocks, certainly improve performance and power consumption in many applications, they have little use in most cryptographic applications, with the exception of embedded memories. The reason for this is that cryptographic algorithms commonly require "exotic" operations, such as arithmetic in finite fields, which are not supported by the

embedded blocks of modern FPGAs. Introduction of blocks specialized for cryptographic operations would naturally benefit the performance and power consumption of cryptographic algorithms on FPGAs, but currently none of the FPGAs available at market supports cryptography if hardwired cryptomodules for programming bitstream decryption are not taken into account.

3.2.4 Comparisons

Generally, FPGAs can provide speedups up to 500 times and power savings of 35–70% compared to general-purpose processors [280]. However, the achievable improvements depend on the implemented application. Cryptographic applications usually achieve significant speedups on FPGAs, *e.g.*, speedups of over 1000 times have been reported for ECC [59]. The efficiency of FPGAs compared to general-purpose processors originates from parallel and pipelined processing and from the fact that general-purpose processors are often ill-suited for operations used in cryptography. Operations causing problems include modular arithmetic and bit-level manipulations on bitvectors with uncommon lengths, *i.e.*, other than 2^w and longer than the wordlength. They cause problems, especially, in public-key algorithms [293].

A recent study in [158] showed that FPGA-based designs are on average 3.4 to 4.6 times slower than ASIC-based designs and implementations on FPGAs consume about 35 times more silicon area. Hence, the speed-area product is over 100 times better for ASICs than for FPGAs. FPGAs also consume about 14 times more dynamic power. Area and power differences reduce substantially by using dedicated memory and multiplier blocks whereas their effect on performance is small. These values suggest that the price of reprogrammability is relatively high. [158]

However, an obvious weakness of the study of [158] is that it simply synthesizes a design for both ASICs and FPGAs and compares the results. Although the values themselves are solid, omitting such advantages as architecture and resource efficiency skews the results because these advantages have a direct impact on the above values. For example, as discussed previously, FPGAs allow optimizing field multipliers for a fixed field but such optimizations are usually out of reach in ASICs. Thus, the difference of FPGAs and ASICs is certainly smaller in cryptographic applications where the effects of these advantages are often significant.

3.3 Metrics for Evaluating Implementations

Thus far, speed and performance were used as arbitrary terms and more precise definitions are necessary. Ways of measuring speed are diverse. Supposedly the simplest is to report computation time, t, *i.e.*, the time in seconds required for a single operation, encryption of a single block, for example. Another way of expressing speed is to provide the number of clock cycles required for a single operation, referred to as latency[1], l. In single clock systems, latency and

[1]Sometimes latency may also refer to computation time but, in this thesis, latency refers strictly to the number of clock cycles, except in **II** where it refers to computation time.

computation time are linked through the following equation:

$$t = \frac{l}{f} \tag{3.1}$$

where f is a clock frequency. Latency is usually more informative than computation time because it is independent of the clock frequency. However, latency and computation time are not always sufficient for measuring speed because implementations capable of computing several operations simultaneously may have long latencies and computation times but can still be considered fast. Throughput, T, the number of operations per second, has considerable importance in evaluating cryptographic processors. If a design can process p operations simultaneously, T, t, and l are related through the equation:

$$T = \frac{p}{t} = \frac{pf}{l}. \tag{3.2}$$

When designs implementing secret-key cryptographic algorithms are considered, it is a common practice to report the number of bits that the design can process in second. In that case, throughput given by (3.2) is multiplied by the number of bits processed in one operation, *e.g.*, with the block size.

Objectives vary in designing cryptographic implementations. A common objective is computation acceleration where the goal is in minimizing t and/or maximizing T. Another objective is implementing an algorithm with as few resources as possible. Supposedly, the most common objective is a combination of these two objectives; namely, maximization of speed under some constraints on resources. It can be measured with area-latency (or area-computation time, *etc.*) efficiencies, *e.g.*, with a product of latency and area of an implementation.

3.4 Side-Channel Attacks

Traditional cryptanalysis is out of the scope of this thesis but cryptanalytic methods called side-channel attacks form a serious threat for many cryptographic implementations and a few words about them are thus in order. Side-channel attacks are methods where Eve is trying to reveal secret keys by exploiting information leaked by an implementation. Side-channel attacks are based, *e.g.*, on time, power, and electromagnetic measurements and they have been successfully applied, especially, against smart cards [271]. Actually, the main threat for practical cryptosystems is not traditional cryptanalysis but rather the weaknesses of implementations [294].

Although smart cards and simple general-purpose processors are generally more vulnerable for side-channel attacks than hardware implementations, successful applications of certain attacks have been reported on FPGA implementations as well [271]. The first such attack was demonstrated in [223] where power analysis attacks were successfully applied to an ECC implementation. The following discusses certain principal side-channel attacks and their countermeasures.

Timing analysis was introduced by Paul C. Kocher in [152]. Timing attacks are based on information about computation time. It can be possible to retrieve information about a secret key by measuring computation time variations depending on the secret key [152]. A successful practical implementation of the attack was demonstrated for smart cards in [75].

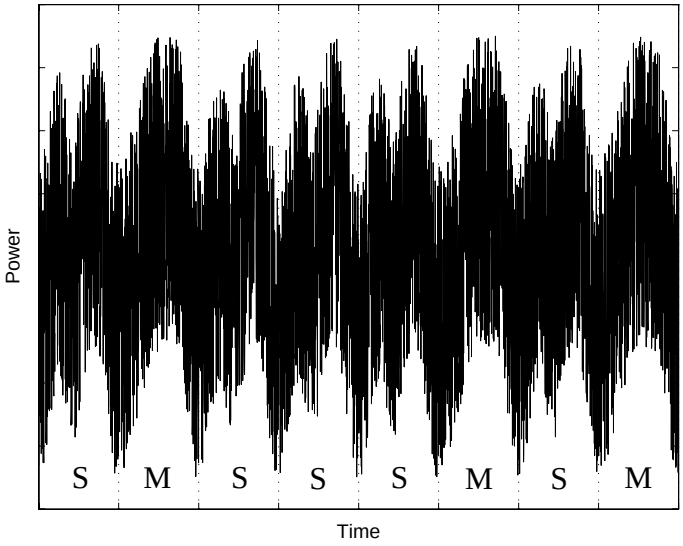

Figure 3.5: Simple artificial example of SPA in revealing a secret exponent in exponentiation using binary method; S stands for squaring and M for multiplication.

Simple Power Analysis (SPA) and Differential Power Analysis (DPA) were introduced by Kocher *et al.* in [151] and they exploit information leaked by power consumption. The idea is that instantaneous power consumption is linked to an instruction or operation currently being executed. Consider exponentiation as an example. Let the exponent be secret as in Diffie-Hellman key exchange, for example; see Sec. 2.2.1. If an exponentiation is performed with the binary method (see Sec. 6.3), each bit in an exponent results in a squaring operation, whereas multiplications are performed only for bits which are ones. It may be possible to reveal the secret exponent by observing instantaneous power consumption. Fig. 3.5 illustrates SPA by presenting an example power trace[2]. Although both squaring and multiplication require equal amount of time, they are clearly distinguishable because their power traces differ from each other. Thus, it is obvious that the bit string used as the exponent in Fig. 3.5 is $\langle 10011 \rangle$.

Because both timing analysis and SPA exploit the feature that some operations can be distinguished from each other by time or power consumption, the obvious way to countermeasure these attacks is to make these operations indistinguishable. In practice this is done by using side-channel atomic blocks which consist of exactly the same operations performed in the same order [60]. Usually achieving side-channel atomicity requires that certain dummy (fake) operations are performed thus reducing performance. It is, however, possible to achieve side-channel atomicity with low overhead as presented for exponentiation and elliptic curves in [60], for example.

SPA operates directly on a single power trace whereas DPA analyzes a large

[2]The power trace in Fig. 3.5 does not represent any actual measurement as it was artificially created for the sake of exemplification.

number of traces with statistical methods [151]. It is considerably harder to protect an implementation against DPA because it can distinguish dummy operations from real ones if there is enough data available and the power model of the implementation is good enough. Protecting implementations against DPA and other statistical attacks is generally very hard but attacking can be made considerably more difficult by time randomization, noise addition, masking, or with dynamic and differential logic styles [271].

Side-channel attacks can be based on electromagnetic emanations, as well. They are referred to as electromagnetic side-channel attacks and they, too, have simple and differential methods similar to SPA and DPA. Electromagnetic side-channel attacks were adapted to AES in FPGAs in [49].

Although side-channel attacks form a serious threat in many applications, hardware implementations are usually much harder to attack than software implementations [271] and FPGAs are sometimes even harder to attack than ASIC implementations [294]. In general, provably secure implementations cannot be made. Thus, the designer's role is to evaluate possible threats and implement countermeasures that measure up with them. As the implementations of this thesis mostly target to high-performance applications where the risk that an attacker is able to measure, *e.g.*, power consumption is low, countermeasures against side-channel attacks have not been considered by any great extent in the publications.

Development of side-channel attacks has been intensive during the past few years. The most important forum for developments in side-channel attacks is probably the Workshop on Cryptographic Hardware and Embedded Systems (CHES) and, thus, interested readers should consult the proceedings of CHES for further information. Various attacks are discussed also in [10]. Comprehensive reviews of power analysis attacks and their countermeasures are given in [180, 226].

Chapter 4

Finite Fields

This chapter presents preliminaries of finite fields which have importance in many cryptographic algorithms. Both AES and ECC use finite field arithmetic and, thus, finite fields are considered before discussing the algorithms. The efficiency of arithmetic in an underlying finite field greatly determines the efficiency of an entire cryptosystem and implementation techniques are thus discussed in the end of this chapter.

Finite fields are commonly referred to as Galois fields in order to honor the contributions of Évariste Galois, a French mathematician, who lived in the early 19th century. His work includes some of the most fundamental results of the theory of finite fields. Finite fields have importance in many branches of mathematics. In addition to their extensive use in cryptography, finite fields have practical applications also in coding theory; see, *e.g.*, [171]. Because of these practical applications, finite fields and, especially, their implementations have been studied also by computer scientists and electrical engineers. Finite fields are extensively studied and an enormous amount of literature exists. Hence, it is impossible to provide an all-inclusive survey. The following study concentrates on the most relevant topics for implementations of cryptographic algorithms with the emphasis being on issues related to efficient hardware implementation.

The following discussion is twofold: First, a more theoretical description is given and, second, practical implementation related aspects are discussed. The basic properties of algebraic structures of groups, rings, fields, and vector spaces are introduced in Sec. 4.1. Two types of finite fields primarily used in cryptographic algorithms, namely prime and binary fields, are considered in Secs. 4.2 and 4.3, respectively. Sec. 4.4 discusses the most complex field operation, inversion. Implementation related aspects are discussed in Sec. 4.5.

4.1 Preliminaries

This section presents preliminaries of algebraic structures: groups, rings, fields, and vector spaces. The discussion is based on comprehensive presentations available in [85, 122, 189].

A set G and a binary operation $*$ form a group $(G, *)$ if they satisfy the following axioms

1. The operation is associative, *i.e.*, $a * (b * c) = (a * b) * c$ for $a, b, c \in G$.
2. There exists an identity element $e \in G$ such that $e * a = a * e = a$ for all $a \in G$.
3. Every element $a \in G$ has an inverse element $b \in G$ such that $a * b = b * a = e$.

Furthermore, the group $(G, *)$ is said to be Abelian (or commutative) if it satisfies

4. $a * b = b * a$ for all $a, b \in G$.

If the group operation is multiplication $\times$, the group $(G, \times)$ is said to be a multiplicative group. In that case, the identity element is denoted by 1 and the inverse element by a^{-1}. If the group operation is addition $+$, the group $(G, +)$ is an additive group, and the identity element is 0 and the inverse element is $-a$.

The order of the group, $\mathrm{ord}(G)$, is the number of elements in the set G. If $\mathrm{ord}(G)$ is finite, the group G is finite. The order of an element $a \in G$, denoted by $\mathrm{ord}(a)$, is the smallest positive integer, n, for which $a^n = e$. If there is an element $a \in G$ such that for each $b \in G$ there is an integer i for which $b = a^i$, the group G is cyclic and a is a generator of G.

A ring $(R, +, \times)$ consists of the set R and two binary operations $+$ (addition) and $\times$ (multiplication), and it satisfies the following axioms

1. $(R, +)$ is an Abelian group with an identity element 0.
2. The operation $\times$ is associative, *i.e.*, $a \times (b \times c) = (a \times b) \times c$ for all $a, b, c \in R$.
3. There is a multiplicative identity element 1 such that $1 \neq 0$ with the property that $1 \times a = a \times 1 = a$ for all $a \in R$.
4. The operation $\times$ is distributive over the operation $+$, *i.e.*, $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in R$.

Furthermore, the ring $(R, +, \times)$ is a commutative ring if it satisfies

5. $a \times b = b \times a$ for all $a, b \in R$.

An element $a \in (R, +, \times)$ is an invertible element if there exists an element $b \in (R, +, \times)$ such that $a \times b = 1$.

A field, henceforth denoted by $\mathbb{F}$, is a commutative ring in which all nonzero elements are invertible. A field with q elements is said to be finite if q is finite, and a finite field is denoted[1] by $\mathbb{F}_q$. Again, the order of $\mathbb{F}_q$, $\mathrm{ord}(\mathbb{F}_q)$, is the number of elements in $\mathbb{F}_q$, and $\mathbb{F}_q$ exists if and only if q is a prime or a power of a prime, *i.e.*, $q = p^m$ where m is a positive integer. If $m = 1$, $\mathbb{F}_p$ is called a prime field and, if $m \geq 2$, $\mathbb{F}_{p^m}$ is called an extension field. Extension fields, where $p = 2$, *i.e.*, $\mathbb{F}_{2^m}$, are called binary fields (or fields with characteristic two), and they are commonly deployed in cryptosystems. There is essentially

[1] The notation $GF(q)$ (Galois field) is also commonly used in the literature and in **I**–**III**.

only one finite field with an order q and therefore two fields $\mathbb{F}_q$ and $\mathbb{F}'_q$ are structurally the same and only the labeling of elements is different, *i.e.*, $\mathbb{F}_q$ and $\mathbb{F}'_q$ are isomorphic.

By definition, $\mathbb{F}_q$ has two operations: addition and multiplication. Subtraction in $\mathbb{F}_q$ is defined through addition so that $a - b = a + (-b)$ for all $a, b \in \mathbb{F}_q$ so that $-b$ is a unique element with the property: $b + (-b) = 0$. Similarly, division is defined for all $a \in \mathbb{F}_q$ and $b \in \mathbb{F}_q \backslash \{0\}$ using multiplication so that $a/b = a \times b^{-1}$ where b^{-1}, the inverse of b, is unique and fulfills $b \times b^{-1} = 1$. Hereafter, multiplication is denoted by juxtaposition and $\times$ is omitted. The nonzero elements in $\mathbb{F}_q$, *i.e.*, the elements in $\mathbb{F}_q \backslash \{0\}$, form a multiplicative group of $\mathbb{F}_q$, denoted by $\mathbb{F}_q^*$, which is cyclic with $\text{ord}(\mathbb{F}_q^*) = q - 1$. Hence,

$$a^q = a \tag{4.1}$$

for all $a \in \mathbb{F}_q$, also known as Fermat's Little Theorem: $a^p \equiv a \pmod{p}$.

A vector space V over a field $\mathbb{F}$ is an additive Abelian group $(V, +)$ with a multiplication, $\mathbb{F} \times V \to V$, if the following axioms are satisfied for all $a, b \in \mathbb{F}$ and $c, d \in V$.

1. $a(c + d) = ac + ad$.
2. $(a + b)c = ac + bc$.
3. $(ab)c = a(bc)$.
4. $1c = c$.

The elements of V are called vectors and the elements of $\mathbb{F}$ are scalars. The finite field $\mathbb{F}_{p^m}$ can be considered as a vector space over $\mathbb{F}_p$. Thus, the elements of $\mathbb{F}_{p^m}$ are vectors and the elements of $\mathbb{F}_p$ are scalars. The dimension of the vector space is m and there are many ways to select a basis.

4.2 Prime Fields

Let p be a prime. Integers modulo p form the set $\{0, 1, 2, \ldots, p-2, p-1\}$ which constructs a finite field with addition and multiplication modulo p as discussed above. The field is denoted by $\mathbb{F}_p$ and called prime field.

Operations in $\mathbb{F}_p$ are computed by using normal integer arithmetic and by reducing the result modulo p. The result is thus the unique integer remainder obtained with a division by p. Consider the prime field $\mathbb{F}_{17}$, for example. The elements in $\mathbb{F}_{17}$ are $\{0, 1, 2, \ldots, 16\}$. An addition $12 + 7 = 2$ because the remainder of 19 divided by 17 is 2. Similarly, $2 - 16 = 3$ and $5 \times 12 = 9$ because $-14 \equiv 3 \pmod{17}$ and $60 \equiv 9 \pmod{17}$, respectively. Furthermore, $5^{-1} = 7$ because $5 \times 7 \equiv 1 \pmod{17}$.

Modular reductions using divisions of large integers are expensive. If p is fixed or changed infrequently (which usually is the case in cryptography), modular reductions can be performed more efficiently with a series of multiplications, additions, and shifts by using precomputed data [22]. This technique is known as the Barrett's reduction.

A widely-used method circumventing the problem of expensive reductions for arbitrary p was introduced by Peter L. Montgomery in [196]. It uses Montgomery representation for integers. The Montgomery representation of $a \in$

$[0, p-1]$ is $aR \bmod p$ where $R > p$ such that $gcd(R, p) = 1$. The value, $R = 2^{\lceil \log_2 p \rceil}$, where $\lceil \cdot \rceil$ denotes rounding to the closest larger integer, is commonly used in practice. For $b \in [0, Rp-1]$, Montgomery reduction is given by $bR^{-1} \bmod p$. When R is a power of the radix, the Montgomery reduction requires only multiplications, additions, and shifts. The conversion from the Montgomery representation to conventional integer representation is performed with the Montgomery reduction. The Montgomery representation does not give any computational advantage for a single multiplication. However, as multiplications in the Montgomery representation are cheap compared to conventional modular multiplications, serious enhancements in computational requirements occur for several multiplications performed in the Montgomery representation. Hence, Montgomery representation is useful, *e.g.*, in modular exponentiation (Diffie-Hellman, RSA, and ElGamal) and ECC over $\mathbb{F}_p$.

Consider multiplication $5 \times 12 \bmod 17$. Let $R = 2^5 = 32$. Hence, $R^{-1} \equiv 8 \pmod{17}$. The Montgomery representations are 7 and 10 for 5 and 12, respectively. An integer multiplication gives $7 \times 10 = 70$ and the Montgomery reduction is $70 \times 8 = 16 \pmod{17}$ which is cheap to compute. The result of the multiplication is finally given by the Montgomery reduction: $16 \times 8 = 9 \pmod{17}$.

Certain reduction methods are developed for primes of special forms. Reductions for primes with the form $p = 2^t - c$ where c is small (fits into one word) can be computed with additions and two multiplications by c [70]. This technique is known as the Crandall reduction. It is commonly known that if p is a Mersenne number, *i.e.*, $p = 2^t - 1$, modular reductions can be computed with a single addition (the most significant half plus the least significant half). Unfortunately, Mersenne primes (primes which are Mersenne numbers) are rare. Generalized Mersenne primes [265] consist of a small number of powers of two, *e.g.*, $p = 2^{192} - 2^{64} - 1$. Modular reduction with generalized Mersenne primes are inexpensive (only additions and subtractions) [265]. The work of [265] was recently generalized for even wider class of primes with some reductions in performance [64]. Many practical cryptosystems use primes of special form. For instance, all primes listed by NIST in [205] are (generalized) Mersenne primes and [52] includes primes for which Crandall reduction can be applied.

4.3 Binary Fields

The elements of $\mathbb{F}_{2^m}$ are represented as a vector space over $\mathbb{F}_2$ with respect to a basis. As mentioned, there are many alternatives for the basis, and all representations with different bases are isomorphic. Because the two elements of $\mathbb{F}_2$ can be represented with a bit, m bits are required in total to represent elements of $\mathbb{F}_{2^m}$. Addition of two elements in $\mathbb{F}_{2^m}$ is simply performed coefficient wise, *i.e.*, bitwise, but multiplication requires information about dependencies between the elements. [85]

$\mathbb{F}_2$ has two elements: $\{0, 1\}$. Addition is computed modulo 2 and thus $0 + 0 = 0$, $0 + 1 = 1$, and $1 + 1 = 0$, *i.e.*, addition is a logical XOR operation denoted by $\oplus$. Multiplication is $0 \times 0 = 0$, $0 \times 1 = 0$, and $1 \times 1 = 1$, which is computed by using a logical AND operation denoted by $\otimes$. The absence of carry makes binary fields suitable from implementation point-of-view, and binary fields are usually considerably faster than prime fields.

4.3.1 Polynomial Basis

Let $f(x)$ be a polynomial whose coefficients are elements of $\mathbb{F}$, *i.e.*,

$$f(x) = \sum_{i=0}^{\infty} f_i x^i \tag{4.2}$$

where $f_i \in \mathbb{F}$. The ring of polynomials over $\mathbb{F}$ is denoted by $\mathbb{F}[x]$. Let $\deg(f(x))$ denote the degree of $f(x)$, *i.e.*, the largest integer m for which $f_i \neq 0$. A polynomial $f(x)$ is an irreducible polynomial over $\mathbb{F}$ if it cannot be presented as a product of two polynomials with positive degrees [189]. Henceforth, an irreducible polynomial is denoted by $p(x)$.

Let $p(x)$ be an irreducible polynomial over $\mathbb{F}_2$ with $\deg(p(x)) = m$ given by

$$p(x) = x^m + \sum_{i=0}^{m-1} p_i x^i = x^m + p'(x) \tag{4.3}$$

where $p_i \in \mathbb{F}_2$. Then, the binary field $\mathbb{F}_{2^m}$ can be constructed by setting

$$\mathbb{F}_{2^m} : \mathbb{F}_2[x]/p(x), \tag{4.4}$$

and the elements of $\mathbb{F}_{2^m}$ can be represented with the basis $\{1, x, x^2, \ldots, x^{m-1}\}$. From now on, such basis is referred to as polynomial basis[2]. An element $a(x) \in \mathbb{F}_{2^m}$ is represented with a polynomial basis as follows:

$$a(x) = \sum_{i=0}^{m-1} a_i x^i \tag{4.5}$$

where $a_i \in \mathbb{F}_2$. In order to simplify, a bitvector representation is commonly used so that $a(x) = \langle a_{m-1} a_{m-2} \ldots a_1 a_0 \rangle$. The identity element of addition, 0, is $\langle 00 \ldots 00 \rangle$ and the identity element of multiplication, 1, is $\langle 00 \ldots 01 \rangle$. [83, 122]

Addition and multiplication are performed modulo $p(x)$. An addition $a(x) + b(x)$ where $a(x), b(x) \in \mathbb{F}_{2^m}$ is simply given by

$$a(x) + b(x) = \sum_{i=0}^{m-1} (a_i \oplus b_i) x^i \ . \tag{4.6}$$

Thus, addition is a bitwise XOR of the bitvectors representing $a(x)$ and $b(x)$. Subtraction is the same because XOR is its own inverse operation. [122]

Multiplication $a(x)b(x)$ where $a(x), b(x) \in \mathbb{F}_{2^m}$ is more complex. First, $a(x)$ and $b(x)$ are multiplied by using ordinary polynomial multiplication and, second, the result is given by taking the remainder after polynomial division by $p(x)$. The result of the ordinary polynomial multiplication, denoted by $c(x) = \sum_{i=0}^{2m-2} c_i x^i$, is a polynomial with $\deg(c(x)) < 2m - 1$. The coefficients of $c(x)$ are given by

$$c_i = \sum_{k=0}^{i} a_k \otimes b_{i-k} \ . \tag{4.7}$$

The number of ANDs can be reduced at the expense of more XORs by adapting techniques presented in [143], called Karatsuba multiplication. [122]

[2] Also called standard or canonical basis

Finally, the multiplication result is obtained by computing

$$a(x)b(x) = c(x) \bmod p(x). \tag{4.8}$$

For $p(x)$ given by (4.3), the computation of (4.8) is based on the following congruence: [122]

$$\begin{aligned} c(x) &= c_{2m-2}x^{2m-2} + \ldots + c_1x^1 + c_0 \\ &\equiv (c_{2m-2}x^{m-2} + \ldots + c_m)p'(x) \\ &\quad + c_{m-1}x^{m-1} + \ldots + c_1x + c_0 \pmod{p(x)} \end{aligned} \tag{4.9}$$

which is applied one bit at a time starting from the msb. The procedure is continued while $\deg(c(x)) \geq m$. Fixing $p(x)$ drastically reduces computational complexity, especially, if $p(x)$ is sparse, *i.e.*, only few p_i are ones in (4.3). Hence, trinomials (three nonzero terms) or pentanomials (five nonzero terms) are often used in practical cryptosystems. [122]

A special case of multiplication, where $a(x) = b(x)$, is called squaring and denoted by $a^2(x)$. Squaring is computationally cheaper than multiplication because $c(x)$ is simply given by

$$c(x) = \sum_{i=0}^{m-1} a_i x^{2i}, \tag{4.10}$$

i.e., $c(x)$ is obtained by inserting zeros between each bit in the bitvector representing $a(x)$ [122]. The result of squaring is obtained by computing (4.8). Complexities of squarings are discussed in more detail in [296], for example.

Consider the field $\mathbb{F}_{2^4}$ as an example. The polynomial $p(x) = x^4 + x + 1$ is irreducible over $\mathbb{F}_2$ and, thus, $\mathbb{F}_{2^4}$ can be constructed as $\mathbb{F}_2[x]/p(x)$. Now, consider two elements of $\mathbb{F}_{2^4}$, $a(x) = \langle 0110 \rangle = x^2 + x$ and $b(x) = \langle 1011 \rangle = x^3 + x + 1$. Then, $a(x) + b(x) = \langle 1101 \rangle$ and $a(x)b(x) = \langle 1111 \rangle$ since $(x^2 + x)(x^3 + x + 1) = x^5 + x^4 + x^3 + x$ and $(x^5 + x^4 + x^3 + x) \bmod p(x) = x^3 + x^2 + x + 1$. The inverse of $a(x)$ is $a^{-1}(x) = \langle 0111 \rangle$ because $(x^2 + x)(x^2 + x + 1) = x^4 + x$ and $(x^4 + x) \bmod p(x) = 1$.

Let $p_1(x)$ be an irreducible polynomial over $\mathbb{F}_2$ with $\deg(p_1(x)) = m_1$. Then, $\mathbb{F}_{2^{m_1}} : \mathbb{F}_2[x]/p_1(x)$. Let $p_2(y)$ be an irreducible polynomial over $\mathbb{F}_{2^{m_1}}$ with $\deg(p_2(y)) = m_2$. Then, $\mathbb{F}_{(2^{m_1})^{m_2}} : \mathbb{F}_{2^{m_1}}[y]/p_2(y)$. These fields are called composite fields. Composite fields introduce computational advantages over conventional binary fields, because arithmetic operations in $\mathbb{F}_{(2^{m_1})^{m_2}}$ can be computed with a series of operations in $\mathbb{F}_{2^{m_1}}$. Naturally, this can be extended to $\mathbb{F}_{(\ldots(2^{m_1})^{m_2}\ldots)^{m_n}}$. Composite fields are frequently used, *e.g.*, in hardware implementations of AES where computational complexity of inversion in $\mathbb{F}_{2^8}$ reduces considerably with composite field representations; see Sec. 5.2.2. However, ECC over composite fields has vulnerabilities against certain cryptanalytic attacks [263]. Prior to finding these vulnerabilities, and even afterwards, they have been used in implementations for efficiency reasons.

Because reductions are expensive also in $\mathbb{F}_{2^m}$, their computation requires attention. The idea of the Barrett's reduction can be applied also for $\mathbb{F}_{2^m}$ over polynomial basis [74]. An adaption of the Montgomery multiplication for $\mathbb{F}_{2^m}$ was introduced in [149] and improved in [298]. Furthermore, analogously to using primes of special forms, the use of special form $p(x)$, *e.g.*, trinomials,

pentanomials, all-one polynomials, or equally-spaced polynomials, offers considerably faster reductions than general irreducible polynomials as discussed above.

4.3.2 Normal Basis

A normal basis is constructed by using a normal element over $\mathbb{F}_2$. The element $\alpha \in \mathbb{F}_{2^m}$ is a normal element if $\alpha, \alpha^2, \ldots, \alpha^{2^{m-1}}$ are linearly independent over $\mathbb{F}_2$. Then, $\{\alpha, \alpha^2, \alpha^{2^2}, \ldots, \alpha^{2^{m-1}}\}$ is a normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. It is commonly known that a normal basis can be found for all $\mathbb{F}_{2^m}$ [202]. An element $a \in \mathbb{F}_{2^m}$ can be represented by using the basis as follows:

$$a = \sum_{i=0}^{m-1} a_i \alpha^{2^i} \tag{4.11}$$

where $a_i \in \mathbb{F}_2$. A bitvector notation is used also for normal basis. The identity element of addition, 0, is $\langle 00 \ldots 00 \rangle$ and the identity element of multiplication, 1, is $\langle 11 \ldots 11 \rangle$. [85]

Addition $a+b$ where $a, b \in \mathbb{F}_{2^m}$ is essentially the same as in polynomial basis and it is given by

$$a + b = \sum_{i=0}^{m-1} (a_i \oplus b_i) \alpha^{2^i}. \tag{4.12}$$

Normal bases are attractive mainly because squaring is very efficient with them. Obviously, squaring (4.11) gives

$$a^2 = \sum_{i=0}^{m-1} a_i \alpha^{2^{i+1}} \tag{4.13}$$

and (4.1) gives that $\alpha^{2^m} = \alpha$. Thus, squaring is a cyclic shift of the bitvector representing a, *i.e.*,

$$a^2 = \langle a_{m-2} a_{m-3} \ldots a_0 a_{m-1} \rangle. \tag{4.14}$$

Multiplication ab, where $a, b \in \mathbb{F}_{2^m}$, is more involved and it is computed using a multiplication matrix, $\mathbf{T}_N$, with entries, $\xi_{i,h}$, satisfying [83]

$$\alpha^{2^i} \alpha = \sum_{h=0}^{m-1} \xi_{i,h} \alpha^{2^h} \quad \text{so that} \quad \alpha^{2^i} \alpha^{2^j} = \sum_{h=0}^{m-1} \xi_{i-j,h-j} \alpha^{2^h} \ . \tag{4.15}$$

Then the bits of $c = ab$ are given by [83]

$$c_h = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \xi_{i-j,h-j} \otimes a_i \otimes b_j \ . \tag{4.16}$$

The number of ones, d_N, in $\mathbf{T}_N$ defines the complexity of multiplication, and it is bounded by $2m - 1 \leq d_N \leq m^2$ [202]. If d_N is on the lower bound, the basis is said to be an Optimal Normal Basis (ONB). There are two types of ONBs, referred to as Type I and Type II ONBs [202]. An ONB does not exist for all m. In fact, the density is about 23 % for $m < 1200$ [202]. For example, an ONB does not exist for $\mathbb{F}_{2^{163}}$ and, as a consequence, the basis used

in **IV**–**VI**, **X**, and **XI** is not an ONB. A comprehensive evaluation of the lowest complexities achievable with different m is provided in [183]. A low complexity normal basis, called Gaussian Normal Basis (GNB), can be constructed if an ONB does not exist [14]. GNBs have been included in many ECC standards, such as [8, 132, 205], and the basis used in the above mentioned publications of this thesis is a GNB.

As shown in (4.16), the leftmost bit of $c = ab$, c_{m-1}, can be computed from a and b with a $2m$-to-1-bit logic function as $c_{m-1} = F(a, b)$. The function is called F-function. Because squaring is a cyclic shift, the leftmost bit of the squaring, c^2, is c_{m-2}. On the other hand, the leftmost bit is again given by the F-function as $c_{m-2} = F(a^2, b^2)$. Following this down to c_0 leads to the following formula:

$$c = ab = \sum_{i=0}^{m-1} F(a^{2^{m-1-i}}, b^{2^{m-1-i}})\alpha^{2^i} \tag{4.17}$$

which defines the Massey-Omura multiplier. [217, 285]

4.4 Inversion

Inversion, *i.e.*, given an element $a \in \mathbb{F}_q$ finding an element $a^{-1} \in \mathbb{F}_q$ such that $aa^{-1} = 1$, is much more involved than addition or multiplication. Moreover, it is commonly required in practical applications of finite fields and, hence, its efficient implementation is important. There are essentially two ways to compute an inversion in $\mathbb{F}_q$: extended Euclidean algorithm and Fermat's Little Theorem.

Extended Euclidean algorithm is an extension of the Euclidean greatest common divisor algorithm which, in addition to finding $gcd(a, b)$, also gives c and d satisfying

$$ac + bd = gcd(a, b) \ . \tag{4.18}$$

If b is a prime and $a < b$ or if b is irreducible with $\deg(b) = m$ and $\deg(a) < m$, $gcd(a, b) = 1$. Thus, by setting $b = p$, the algorithm returns c such that $ac \equiv 1 \pmod{p}$, *i.e.*, $c = a^{-1}$ in $\mathbb{F}_p$. Analogously, if $b = p(x)$, then $c(x) = a^{-1}(x)$ in $\mathbb{F}_{2^m}$. [122]

The standard version of the extended Euclidean algorithm requires several integer (for $\mathbb{F}_p$) or polynomial (for $\mathbb{F}_{2^m}$) divisions, which can be costly. A binary gcd algorithm which requires only additions, shifts (divisions by 2), and parity tests was introduced in [273]. An algorithm for binary inversion in $\mathbb{F}_{2^m}$ was given in [30]. An algorithm, called almost inverse algorithm, which returns a weighted inverse, $x^t a^{-1}(x)$, was presented in [260]. Algorithms for efficient inversions in the Montgomery representation were given in [142, 256]. Normally, division b/a is computed by multiplying b with a^{-1}, but modifications of the above algorithms performing a direct division have been given in [54, 106], for example.

Inversion based on Fermat's Little Theorem uses consecutive squarings and multiplications in $\mathbb{F}_q$. It follows directly from (4.1) that for $a \in \mathbb{F}_q$

$$a^{-1} = a^{q-2} \ . \tag{4.19}$$

Hence, inversion in $\mathbb{F}_{2^m}$ is given by $a^{-1} = a^{2^m-2}$. Optimizing this exponentiation returns to the theory of addition chains[3]. In exponentiation, the addition chain consists of multiplications and squarings required to exponentiate a. A straightforward application of the binary method results in an addition chain with the cost of $m-2$ multiplications and $m-1$ squarings [285], because $2^m - 2 = \langle 111\ldots1110\rangle_2$ (the length is m bits). Because m is known *a priori*, addition chains with lower costs than binary method can be searched for exponentiation. Efficient addition chains proposed in [136], called Itoh-Tsujii inversion, reduce the cost to $\lfloor \log_2(m-1) \rfloor + H(m-1) - 1$ multiplications and $m-1$ squarings, where $\lfloor \cdot \rfloor$ denotes rounding to the closest smaller integer and $H(\cdot)$ is the Hamming weight, the number of ones in the binary representation. The Itoh-Tsujii inversion is efficient, especially, in normal basis because squarings are almost free [136], but it can be efficiently used in polynomial basis, too [113, 295].

Because inversions are typically more expensive than other field operations, it is of interest to trade inversions to other operations. A method, known as Montgomery's trick [197], trades inversions to multiplications. It is based on the observation: $a^{-1} = b(ab)^{-1}$ and $b^{-1} = a(ab)^{-1}$, which is generalized for n inversions resulting in a cost of one inversion and $3(n-1)$ multiplications [197, 261]. Montgomery's trick is a general method, but some specific methods for trading inversions to other operations are presented in Secs. 5.2.2 and 6.2, which discuss AES S-box optimization with composite fields and elliptic curve arithmetic in projective coordinates, respectively.

4.5 Notes on Implementations

The following presents certain notes on implementation related aspects. This discussion is not all-inclusive because of the superfluity of papers discussing efficient implementation of finite fields in various applications. However, the following points give a brief overview of issues related to implementations of finite field arithmetic and present the most commonly used multiplier structures. The discussion focuses on $\mathbb{F}_{2^m}$ and multiplication because $\mathbb{F}_p$ is not used in the publications of this thesis and multiplication dominates in both computation time and resource requirements of practical cryptosystems. The subject is revisited in Sec. 6.5 which discusses ECC processors.

A bit-serial multiplier computes one bit of the product of two elements of $\mathbb{F}_{2^m}$ in one clock cycle, thus, requiring m clock cycles in total. A bit-parallel multiplier computes all bits of the product simultaneously in one clock cycle. Obviously, the bit-parallel multiplier consumes a lot of area while the bit-serial multiplier is slow. A digit-serial multiplier is a tradeoff between these two extremes and it computes D bits of the product in one clock cycle resulting in $\lceil m/D \rceil$ clock cycles for an entire multiplication. As a result of the rounding, only certain values of D reduce latency and only they should be considered for use in practical applications in order to prevent unnecessary use of area. For instance, D should be chosen from the set $\{1-15, 17, 19, 21, 24, 28, 33, 41, 55, 82, 163\}$ for $\mathbb{F}_{2^{163}}$ used in all ECC publications of this thesis. Furthermore, there exists a value of D optimizing the latency-area product of a multiplier. This value

[3] An addition chain is the chain of additions required to get a starting from 1 by using only values that have appeared previously in the chain.

is usually small (see **IV**) which challenges practical feasibilities of digit-serial multipliers with large D and, especially, bit-parallel multipliers.

A multiplier is said to be fixed if it supports only one specific field and the ability to support several fields is called flexibility. As a rule of thumb, the more flexible a multiplier is the slower it is and the more area it requires. An extreme example is squaring in polynomial basis which can be computed in one clock cycle with a fixed (sparse) irreducible polynomial but requires several clock cycles with an arbitrary irreducible polynomial.

A classical bit-parallel multiplier for polynomial basis was presented in [181, 182], and it is nowadays referred to as the Mastrovito multiplier after its invertor. A Mastrovito multiplier combines the polynomial multiplication and reduction phases to a single matrix multiplication. An optimized Mastrovito multiplier for trinomials was developed in [277] which was later generalized for arbitrary irreducible polynomials in [117]. A systematic approach for designing Mastrovito multipliers was presented in [305]. A disadvantage of Mastrovito multipliers is that flexible multipliers cannot be designed. Other bit-parallel multipliers have been presented, *e.g.*, in [233, 235].

Because m is usually large in public-key cryptography applications, bit-serial or digit-serial multipliers are commonly preferred in practice. Ordinary polynomial multiplication, as given by (4.7), and Karatsuba multiplication [109, 143, 283] can be implemented in both bit-serial and digit-serial fashion. Complexity of reduction has been studied with various types of irreducible polynomials in [295, 297]. The following multipliers interleave polynomial multiplication and reduction steps but still incorporate different circuitries for both of them contrary to the Mastrovito multiplier. A multiplier, called super-serial multiplier, suitable for applications where resources are very scarce was presented in [219] and it allows trading off resources to even longer latency than that of the bit-serial multiplier. Digit-serial multipliers for polynomial basis were introduced in [269]. They were improved in [156] by introducing new multiplier architectures with different numbers of accumulators in order to minimize the critical path and by studying their optimality in terms of speed and area. A digit-serial multiplier utilizing LUTs suitable for software implementations was introduced in [123], but it can be efficiently implemented in hardware as well [123, 177]. It was used in **VI** and **VII**.

If the reduction step is interleaved with polynomial multiplication, it can be computed very efficiently even with arbitrary irreducible polynomials by using the so-called partial reduction techniques [115] where a result of ordinary polynomial multiplication is reduced only to a congruent polynomial with a degree higher than m. Partial reduction embedded in an efficient digit-serial multiplier can provide support for arbitrary irreducible polynomials (up to certain m) with half the speed compared to fixed polynomials [87]. Combination of specific reduction circuitries for a few fixed irreducible polynomials and circuitry for arbitrary irreducible polynomials provides flexible multipliers with fast computation for frequently used fields [87, 114, 115].

A number of papers have discussed multiplication in normal basis. A classical multiplier structure for normal basis is the Massey-Omura multiplier described in [217, 285]. Normal basis multiplication is easy to parallelize because the same F-function is used for all bits of the result, as shown in (4.17). Bit-serial architectures have been presented in [1, 217, 285], digit-serial multipliers in [232, 234], and bit-parallel multipliers in [278, 285], to name a few. Several

multipliers exploit specific types of normal basis. A multiplier structure for ONB was presented in [1]. A bit-parallel multiplier, called Sunar-Koç multiplier, that is smaller than the bit-parallel Massey-Omura multiplier was given for Type II ONBs in [278]. Multiplier architectures specifically for GNBs were presented in [159, 236].

Two bit-parallel polynomial basis multipliers, based on ordinary polynomial multiplication and modified Karatsuba method [109, 143], and normal basis multipliers, digit-serial ($D = 117$) Massey-Omura and bit-parallel Sunar-Koç multipliers, were compared in [109] on a Xilinx Virtex-II FPGA in the case of $\mathbb{F}_{2^{233}}$ for which exists a Type II ONB. They concluded that their modified Karatsuba multiplier outperforms ordinary polynomial multiplier. The Massey-Omura and Sunar-Koç multipliers had similar time-area products, but Sunar-Koç was faster. Sunar-Koç was also faster than both polynomial basis multipliers, but Karatsuba had clearly the best time-area product.

Polynomial basis is superior over normal basis in software; see, *e.g.*, [121, 213]. Both bases result in efficient implementations in hardware and, thus, normal basis is a viable choice especially in applications where the number of squarings is high, such as ECC with Koblitz curves. Commonly support for normal basis is implemented in software by utilizing the isomorphism of different representations of $\mathbb{F}_q$ so that elements are converted to polynomial basis where computations are carried out followed by a conversion back to normal basis. However, such conversions can be costly and direct implementation can be more efficient in certain applications [213] and software implementations of normal basis have been proposed, such as [213, 232, 242]. Conversions between normal basis and polynomial basis are very efficient with Type I ONB and all-one polynomials which enables efficient implementations as shown in [150, 166]. However, Type I ONBs are not practical in ECC because they can exist only if m is even [202], although it should be a prime for security reasons; see, *e.g.*, [99].

Because many standards in (elliptic curve) cryptography, *e.g.*, [51, 52, 205], define both prime and binary fields, an implementation must support both $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ in order to cover all standardized fields. Hence, processors combining support for $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ have attained interest. The first such processors were presented in [106, 257] which presented unified Montgomery multipliers supporting both prime and binary fields with arbitrary field sizes. The work of [106] was later extended in [107]. Another multiplier for $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ based on Montgomery multiplication was given in [255]. A multiplier which does not use Montgomery representation was presented in [111]. A low-power arithmetic unit supporting addition, multiplication, and extended Euclidean algorithm in $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ was presented in [291] and instruction set extensions for a 32-bit RISC (Reduced Instruction Set Computer) processor supporting both $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ were developed in [112]. A multiplier supporting both $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ typically requires only slightly more area than an $\mathbb{F}_p$ multiplier and, in fact, the support for $\mathbb{F}_{2^m}$ can be added to an $\mathbb{F}_p$ multiplier with no additional area cost [255]. In all cases, $\mathbb{F}_{2^m}$ operations clearly outperform $\mathbb{F}_p$ operations with comparable field sizes.

As mentioned, inversion is the most complex field operation. Hence, its fast computation is important in applications where inversions are frequently used. Inversion algorithms based on extended Euclidean algorithm are preferred in software implementations, because they usually result in slightly faster results. However, Fermat's Little Theorem computes inversions without an additional

inverter circuitry, namely with successive multiplications and squarings, which makes it a feasible alternative in hardware implementations.

To summarize, an enormous amount of work has been done on realizing finite field arithmetic and many architectures are available having merits in various different target applications. The research field can be considered mature and this thesis does not introduce any major contributions to this field. A scalable multiplier that can be easily generated with an automatic generator and that suits for the 4-to-1-bit LUT structure of Xilinx FPGAs was, however, introduced in **III**. Otherwise, the contributions of this thesis are on higher algorithm levels than field arithmetic and they are discussed in the following chapters.

Chapter 5

Advanced Encryption Standard

THIS CHAPTER studies AES from the hardware implementation point-of-view and provides an extensive review of implementation methods developed for AES. The objective is to review the state-of-the-art of AES implementations and to point out the relationship of **I** and **II** to other studies. The focus is again on FPGAs, but also ASIC implementations are discussed in detail in order to provide an inclusive view of the state of hardware implementation of AES and because many techniques used in FPGAs have been first introduced for ASICs. Software implementations are not discussed in the following description but, to the author's knowledge, the fastest reported software has been written by Helger Lipmaa in assembly and it achieves throughput of over 1.5 Gbps on a 3.2 GHz Pentium 4 [172]. This value is a useful benchmark for hardware implementations. It should be noted, however, that slower hardware implementations can also be useful because low price or power consumption may be required which typically cannot be achieved with general-purpose microprocessors.

AES algorithm is presented in Sec. 5.1. Different implementation techniques presented for AES are discussed in Sec. 5.2 and Sec. 5.3 reviews and compares published ASIC and FPGA implementations of AES.

5.1 Description of AES

The following description of AES is based on [71, 206]. AES has a block size of 128 bits and there are three options for the key size: 128, 192, or 256 bits. AES-128, AES-192, and AES-256 henceforth refer to AES and the key size. AES is an iterative SPN and the number of rounds, Nr, depends on the key size so that $Nr = 10$ for AES-128, $Nr = 12$ for AES-192, and $Nr = 14$ for AES-256.

AES operates on an intermediate result called the STATE. At the beginning of encryption, the 128-bit M is interpreted as a two dimensional table or a matrix with bytes as elements where the first byte of M is $s_{0,0}$ and the last one is $s_{3,3}$. The rows and columns of the matrix are referred to as rows and columns of the STATE. AES involves four different transformations which are all applied to the STATE. Separate round keys are used in different rounds and they are derived in a key schedule. Once all rounds have been processed, C is build up

so that $s_{0,0}$ becomes the first byte of C and $s_{3,3}$ becomes the last one. Alg. 5.1 shows the outline of the AES algorithm. The transformations are considered in the following.

Algorithm 5.1 Outline of the AES encryption

Input: M (128 bits), K (128, 192 or 256 bits)
Output: C (128 bits)

```
1. State ← M
2. RoundKeys ← KeySchedule(K)
3. State ← AddRoundKey(State, RoundKeys[0])
4. for i = 1 to Nr − 1 do
5.    State ← SubBytes(State)
6.    State ← ShiftRows(State)
7.    State ← MixColumns(State)
8.    State ← AddRoundKey(State, RoundKeys[i])
9. end for
10. State ← SubBytes(State)
11. State ← ShiftRows(State)
12. State ← AddRoundKey(State, RoundKeys[Nr])
13. C ← State
```

The AddRoundKey$(\cdot,\cdot)$ transformation embeds key into the State by performing a 128-bit bitwise XOR, *i.e.*,

$$\text{AddRoundKey}(\text{State}, \text{RoundKeys}[i]) = \text{State} \oplus \text{RoundKeys}[i]. \quad (5.1)$$

AddRoundKey$(\cdot,\cdot)$ is the only transformation in AES involving the key.

The SubBytes$(\cdot)$ transformation is a non-linear operation which operates on each byte of the State independently. A byte is considered to be an element of the field

$$\mathbb{F}_{2^8} : \mathbb{F}_2[x] \,/\, x^8 + x^4 + x^3 + x + 1 \ . \quad (5.2)$$

It is, thus, presented as $\sum_{i=0}^{7} b_i x^i$ so that b_0 is the lsb. SubBytes$(\cdot)$ consists of two phases:

1. Computation of an inversion in $\mathbb{F}_{2^8}$. The element $\langle 00 \rangle_x$ maps to itself.
2. An affine transformation over $\mathbb{F}_2$ defined as

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (5.3)$$

   where $c = \langle 63 \rangle_x$ and $0 \leq i \leq 7$.

The ShiftRows$(\cdot)$ transformation operates on the 32-bit rows of the State independently. Each byte of the row i is shifted cyclically to the left by i bytes with $0 \leq i \leq 3$.

The MixColumns$(\cdot)$ transformation operates on the 32-bit columns of the State independently. Each column is considered as an element of the finite field

$$\mathbb{F}_{(2^8)^4} : \mathbb{F}_{2^8}[x] \,/\, x^4 + 1 \quad (5.4)$$

where $\mathbb{F}_{2^8}$ is defined by (5.2). The columns are then multiplied with a fixed polynomial

$$a(x) = \langle 03 \rangle_x x^3 + \langle 01 \rangle_x x^2 + \langle 01 \rangle_x x + \langle 02 \rangle_x \ . \quad (5.5)$$

The KeySchedule($\cdot$) routine expands the length of the key, K, to $128(Nr+1)$ bits. The result is used as RoundKeys$[i]$ in encryption as presented in Alg. 5.1. K is placed into the beginning of RoundKeys and, thus, K forms the first 128, 192, or 256 bits of RoundKeys depending on the key size. From this point forward, KeySchedule($\cdot$) builds RoundKeys one 32-bit word at a time with SubBytes($\cdot$), XORs, and shifts.

More detailed descriptions of AES are available in [71, 206].

5.1.1 Decryption

Decryption inverts all transformations performed in encryption. Thus, the order in which transformations are performed is reversed and the transformations are replaced by their inverses. However, because the inverse of SubBytes($\cdot$) operates byte-wise, it can be computed before the inverse of ShiftRows($\cdot$) and a structure similar to Alg. 5.1 can be devised also for decryption [71, 206].

The AddRoundKey($\cdot,\cdot$) is its own inverse and it remains unchanged. The inverse of the ShiftRows($\cdot$) shifts bytes to the right, not to the left, and the inverse of the SubBytes($\cdot$) performs an inverse of the affine transformation followed by an inversion in $\mathbb{F}_{2^8}$ which is its own inverse operation by definition. The inverse of the MixColumns($\cdot$) multiplies a column with $a(x)^{-1} = \langle 0b\rangle_x x^3 + \langle 0d\rangle_x x^2 + \langle 09\rangle_x x + \langle 0e\rangle_x$. The KeySchedule($\cdot$) is exactly the same for both encryption and decryption but the RoundKeys$[i]$ are used in reversed order. [71, 206]

5.2 Implementation of AES

Although AES is a well-defined standard, there are certain choices that can be made and both academic and commercial AES implementations typically implement only a part of the full AES. For instance, only encryption or decryption is implemented or support only for 128-bit keys is included in order to minimize area and increase speed. Many published designs, *e.g.*, [37, 42, 55, 86, 154, 169, 201, 309] and **II**, include only AES-128 encryption. Some implementations, such as [86, 91, 97, 201], do not implement KeySchedule($\cdot$) and they require that RoundKeys are computed by an external processor or that the key K is fixed and RoundKeys are precomputed. The support for KeySchedule($\cdot$) is referred to as key agility and if an implementation can switch keys without additional delays, it is fully key agile. Achieving full key agility can require large amounts of area, especially, if several blocks are encrypted or decrypted simultaneously. Implementations supporting the full AES, *i.e.*, all key sizes with key agility, encryption, and decryption, are presented in [4, 179, 228, 275].

AES can be implemented with basic techniques used in implementing block ciphers [91]. They include unrolling and pipelining. The following discussion is based on categorizations from [100, 120, 306]. Different unrolling and pipelining techniques are depicted in Fig. 5.1. An iterative architecture implements only one round of the algorithm which is then used iteratively. A loop-unrolled architecture performs several rounds in one iteration. If all rounds are unrolled, an architecture is referred to as fully unrolled. Naturally, the more rounds are unrolled the more area is required to implement the architecture. Pipelining can be categorized as outer-round or inner-round pipelining. Outer-round pipelining

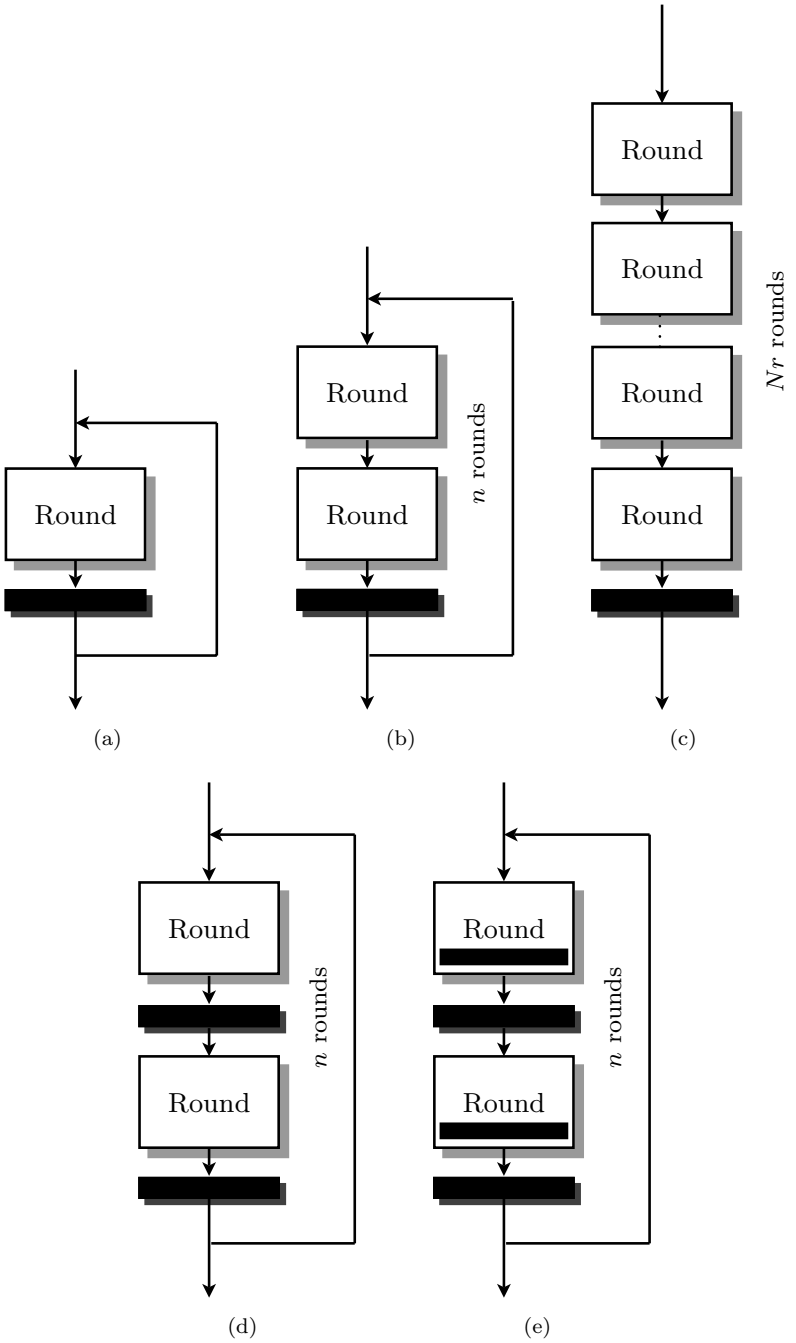

Figure 5.1: Implementation techniques (adapted from [100, 120, 306]): (a) iterative, (b) loop-unrolled, (c) fully unrolled, (d) loop-unrolled with outer-round pipeline, and (e) loop-unrolled with inner-round and outer-round pipeline. The black bars depict register stages. The number of inner-round pipeline stages may be higher than one and the number of outer-round pipeline stages is bounded by the number of unrolled rounds.

includes registers between two or several rounds, whereas inner-round pipelining adds registers inside a round between or in transformations. In either case, the total number of pipeline stages determines the number of simultaneous encryptions. The highest throughputs are achieved with fully unrolled architectures utilizing aggressive inner-round pipelining.

If the entire STATE is operated simultaneously, the width of the data path is 128 bits. However, even the iterative architecture with a 128-bit data path may be too large for certain applications. A technique called folding uses smaller data path, such as 32 or 8 bits, so that the same circuitry is used for different parts of the STATE during several clock cycles. Folding is nontrivial because SHIFTROWS(·) complicates accessing bytes of the STATE and results in a large switching circuitry [61]. The penalty in speed can be considerable in folded architectures but, nonetheless, even lower speed is adequate for many consumer applications where low cost is often essential [104].

Throughput of an AES implementation is given by the following equation which is similar to (3.2) but gives the number of bits processed per second

$$\mathrm{T} = 128\,\frac{\mathrm{pf}}{\mathrm{l}} \tag{5.6}$$

where f, l, and p are as defined in Sec. 3.3; hence, p is the number of pipeline stages. In a fully unrolled (and pipelined) design, $\mathrm{p} = \mathrm{l}$ resulting in $\mathrm{T} = 128\mathrm{f}$.

Block ciphers are used in different modes of operations. The simplest mode of operation is Electronic Code Book (ECB) mode where each block is encrypted separately. Thus, the same plaintext encrypted with the same key always results in the same ciphertext. This can be avoided by using more sophisticated modes of operations. They commonly require that the ciphertext of the previous block is available in encryption of the next block which prevents efficient pipelining. However, certain modes of operations, *e.g.*, Cipher Block Chaining (CBC) mode, allow pipelining in decryption but not in encryption whereas certain modes of operations, *e.g.*, counter (CTR) mode, allow pipelining in both. The reader is referred to basic textbooks on cryptography, *e.g.*, [258, 274], for further information.

5.2.1 Memory-based Implementations

As discussed in Sec. 5.1, SUBBYTES(·) operates the STATE byte-wise. It computes an inversion in $\mathbb{F}_{2^8}$ and performs an affine transformation. These operations can be combined and performed with a single look-up from a precomputed LUT of 256 8-bit elements and such a table is called S-box. This approach is particularly feasible for software, but also for FPGAs where logic resources can be saved at the expense of a few embedded memory blocks, *e.g.*, BlockRAMs in Xilinx FPGAs. In total 16 S-boxes are required if the entire SUBBYTES(·) is computed in parallel, and the KEYSCHEDULE(·) requires additional 4 S-boxes. This approach is straightforward and it has been successfully used in numerous papers, such as [62, 97, 185, 248, 272, 304]. However, memory requirements may restrict the use of this approach in certain low-cost environments and fully unrolled designs. There are also certain other issues that require commenting.

Because SUBBYTES(·) and its inverse differ, efficient implementation of both encryption and decryption in the same design is not easy. A straightforward implementation requires two different LUTs: one for encryption and one for

decryption. This problem can be circumvented with an approach using combinatorial logic for the affine transformation and its inverse and a LUT only for inversion in $\mathbb{F}_{2^8}$ [176, 240]. Hence, both encryption and decryption can be implemented with the same memory requirements than plain encryption or decryption with an increase in logic requirements and computation delay. Another approach is to use two ROMs (Read-Only Memories) including precomputed values for the S-boxes of SubBytes(·) and its inverse [185]. Encryption and decryption are then carried out with LUTs implemented in programmable embedded memory, *e.g.*, BlockRAMs in Xilinx Virtex FPGAs, whose contents are loaded from one of the ROMs at the beginning [185]. However, switching latencies may become a problem if encryption and decryption modes need to be switched frequently.

SubBytes(·), ShiftRows(·), and MixColumns(·) can be combined into column-wise LUTs, referred to as T-boxes. This approach was first proposed by the inventors of AES for 32-bit software [71], but it can be efficiently adapted to memory-rich FPGAs, too, and it has been used at least in [56, 86, 97, 103, 186]. The quadruple memory requirements compared to the S-box approach [97] are a downside of this method.

Let $S(a)$ denote an S-box. Then, four T-boxes needed in AES encryption are given as follows [71]:

$$\begin{aligned} T_0(a) &= \begin{bmatrix} S(a) \times \langle 02 \rangle_x \\ S(a) \times \langle 01 \rangle_x \\ S(a) \times \langle 01 \rangle_x \\ S(a) \times \langle 03 \rangle_x \end{bmatrix} \quad T_1(a) = \begin{bmatrix} S(a) \times \langle 03 \rangle_x \\ S(a) \times \langle 02 \rangle_x \\ S(a) \times \langle 01 \rangle_x \\ S(a) \times \langle 01 \rangle_x \end{bmatrix} \\ T_2(a) &= \begin{bmatrix} S(a) \times \langle 01 \rangle_x \\ S(a) \times \langle 03 \rangle_x \\ S(a) \times \langle 02 \rangle_x \\ S(a) \times \langle 01 \rangle_x \end{bmatrix} \quad T_3(a) = \begin{bmatrix} S(a) \times \langle 01 \rangle_x \\ S(a) \times \langle 01 \rangle_x \\ S(a) \times \langle 03 \rangle_x \\ S(a) \times \langle 02 \rangle_x \end{bmatrix} \end{aligned} \tag{5.7}$$

each of which can be implemented with a LUT of 256 32-bit elements. The values of a new column are received by using the bytes of an old column as inputs for the T-boxes and XORing the 32-bit outputs. In T-boxes used in decryption, $\langle 01 \rangle_x$, $\langle 02 \rangle_x$, and $\langle 03 \rangle_x$ are replaced by $\langle 0b \rangle_x$, $\langle 0d \rangle_x$, $\langle 09 \rangle_x$, and $\langle 0e \rangle_x$.

As shown in Alg. 5.1, MixColumns(·) is not computed in the last round and, hence, an implementation must be capable of computing only SubBytes(·) (or its inverse), *i.e.*, $S(a)$. This is not a major problem in encryption because the T-boxes include plain S-boxes as well (those multiplied by $\langle 01 \rangle_x$) [71]. However, in decryption the S-boxes are not available and they must be multiplied out of the T-boxes which requires additional logic.

The T-boxes provide faster implementations than the S-boxes, but expectedly with increased memory requirements [97]. Because modern FPGAs are usually memory-rich, the T-boxes can provide very fast performance with low logic requirements as shown in [56, 86, 186] as well as very small logic requirements [245].

5.2.2 Combinatorial Implementations

Memory must be available in order to implement the previously discussed approaches efficiently. However, it is important to have an efficient method for

implementing SubBytes(·) with combinatorial logic, especially, in ASIC implementations. Efficient combinatorial implementation of the S-boxes is important also in high-speed implementations on FPGAs because throughput can be increased by pipelining the S-boxes which is not possible with memory-based implementations. Straightforward derivations of combinatorial expressions for SubBytes(·), either directly from the truth-table of the S-box or with Fermat's Little Theorem, result in fast but large circuitries [201]. Composite fields, discussed in Sec. 4.3.1, reduce the complexity of combinatorial inversion considerably.

Using composite fields in the S-boxes was proposed by Rijmen [237]. The idea is that although all representations of $\mathbb{F}_{2^8}$ are isomorphic, computational complexities differ between representations and it is more efficient to compute inversions in a composite field rather than in the field specified by (5.2).

A byte is first mapped with an isomorphism, σ, to

$$\mathbb{F}_{(2^4)^2} : \mathbb{F}_{2^4}[y]/y^2 + \psi y + \omega \ . \tag{5.8}$$

An inversion for an element, $a = a_1 y + a_0$ where $a_i \in \mathbb{F}_{2^4} : \mathbb{F}_2[y]/y^4 + y + 1$, is then computed with the following equation

$$(a_1 y + a_0)^{-1} = a_1 \Delta y + (a_1 + a_0 \psi)\Delta \tag{5.9}$$

where $\Delta = (a_1^2 \omega + a_1 a_0 \psi + a_0^2)^{-1}$ [237]. Finally, the result is mapped back to the original representation with an isomorphism, σ^{-1}. Hence, an inversion in $\mathbb{F}_{2^8}$ has been replaced by two isomorphisms and an inversion and some multiplications, squarings, and additions in $\mathbb{F}_{2^4}$. [237]

The constants ψ and ω effect both the complexity of operations of (5.9) and mappings, σ and σ^{-1}. The values can be chosen freely as long as $y^2 + \psi y + \omega$ remains irreducible over $\mathbb{F}_{2^4}$. If $\psi = 1$, there are four possible values for ω and, for each of them, there are seven alternatives for σ and σ^{-1} [247].

The above approach can be used further so that operations of $\mathbb{F}_{2^8}$ are decomposed into operations of $\mathbb{F}_2$ with irreducible trinomials as follows [254]

$$\mathbb{F}_{((2^2)^2)^2} : \mathbb{F}_{(2^2)^2}[y]/y^2 + \psi_0 y + \omega_0 \tag{5.10}$$

$$\mathbb{F}_{(2^2)^2} : \mathbb{F}_{2^2}[y]/y^2 + \psi_1 y + \omega_1 \tag{5.11}$$

$$\mathbb{F}_{2^2} : \mathbb{F}_2[y]/y^2 + \psi_2 y + \omega_2 \ . \tag{5.12}$$

If $\psi_{0,1,2} = 1$, then there are eight possible values for ω_0, two for ω_1, and one for ω_2 [308]. Decompositions beyond $\mathbb{F}_{2^4}$ can be beneficial for ASIC implementations but they do not reduce the complexity of inversion in typical FPGA implementations because the 4-to-1-bit LUT structure provides all arithmetic operations in $\mathbb{F}_{2^4}$ already with the minimum cost of 4 LUTs [105]. Besides, it is simple to directly derive equations for operations in $\mathbb{F}_{2^4}$ and they provide more efficient inversions in $\mathbb{F}_{2^4}$ than further decompositions [128, 307]. However, the simplest isomorphisms, σ and σ^{-1}, have been presented by decomposing $\mathbb{F}_{2^8}$ to $\mathbb{F}_{((2^2)^2)^2}$ [48, 308]. Naturally, there are no restrictions for the basis of $\mathbb{F}_{2^4}$, and normal bases have been suggested at least in [48, 275].

The first practical implementation of composite fields was presented in [247] where the composite field $\mathbb{F}_{(2^4)^2}$ was used for all operations of AES, *i.e.*, also MixColumns(·) was performed in the composite field. Thus, σ was applied

to the plaintext and the key once in the beginning and the resulted ciphertext was mapped with σ^{-1} in the end. This was also the approach taken in **II**. However, it has become evident that it is more efficient for the overall area consumption and critical path to perform only inversions in composite fields [48, 254, 292, 307, 308]. The reasons are that the simple multiplications of MixColumns$(\cdot)$ become much more complex in composite fields and either σ or σ^{-1} can be embedded in affine transformations which reduces the overhead of these mappings. The first implementations using composite fields to compute only the inversion were presented in [254] using $\mathbb{F}_{((2^2)^2)^2}$ and [292] using $\mathbb{F}_{(2^4)^2}$.

Evaluations of the optimal composite field constructions have been presented in [48, 190, 308]. The study of [190] optimized the implementation of [254] by studying effects of different irreducible polynomials to the complexity of the isomorphisms. In [48], various different constructions including normal bases were studied resulting in the smallest S-box available in the literature. Slightly larger circuit with shorter critical path was given in [308]. The exact values are 91 XORs and 36 ANDs for [48] with a critical path of 23 XORs and 4 ANDs and 120 XORs and 35 ANDs for [308] with a critical path shorter by 4 XORs. The difference in area consumption is notable with all above mentioned decompositions compared to straightforward implementations from the truth-table of the S-box which requires about 1500-3000 ASIC gates with different techniques [201]. A study on the most energy efficient construction in $\mathbb{F}_{((2^2)^2)^2}$ was provided in [200]. The above mentioned complexity studies are not valid for FPGAs where LUT structures change both area requirements and critical paths from the above values as shown for one construction in [309]. However, optimal pipeline constructions in composite fields for FPGAs were studied in [103, 105].

Although SubBytes$(\cdot)$ dominates in area consumption and computation time, MixColumns$(\cdot)$ also deserves attention, especially, when combined with its inverse; see, *e.g.*, [98, 129, 254, 306]. A basic implementation technique, which was described already in the standard [206], builds around multiplications by $\langle 02 \rangle_x$ which can be implemented with shifts and 4 XORs [306]. Both MixColumns$(\cdot)$ and its inverse can be implemented with these multiplications and additions (XORs). When both encryption and decryption are implemented, area can be reduced by sharing multiplications by $\langle 02 \rangle_x$ [307] or by decomposing MixColumns$(\cdot)$ out of its inverse [71]. The latter approach utilizes the fact that the polynomial used in multiplication in the inverse of MixColumns$(\cdot)$, $a^{-1}(x)$, can be expressed by using $a(x)$ of MixColumns$(\cdot)$, (5.5), as $a^{-1}(x) = (\langle 04 \rangle_x x^2 + \langle 05 \rangle_x) a(x) \pmod{x^4+1}$. As a consequence, the inverse of MixColumns$(\cdot)$ is computed by first applying multiplications by $\langle 04 \rangle_x$ and $\langle 05 \rangle_x$ followed by MixColumns$(\cdot)$.

5.3 Literature Review of AES Implementations

This literature review complements the review presented in **I**. This review surveys progress that has took place after the writing of **I** and considers also published ASIC implementations which were not discussed in **I**. This review also updates the previous ones presented in [118, 120, 293] by including implementations which appeared after their publication. The scope is also wider compared to [120], which focused only on low-cost implementations. Details of implementations are collected into Table 5.1. Selected implementations from

I are also discussed in order to avoid discontinuations, and in particular included in Table 5.1 for reader's convenience. If a publication presents several implementations, only the ones which are the most comparable with other implementations in the literature and, especially, with **II** were selected. Commercial AES implementations were not included in this review because only limited and marketing-oriented information is available.

5.3.1 ASIC Implementations

This review begins with ASIC implementations. The first studies on ASIC-based implementation of AES (Rijndael) were published during the AES selection process by Ichikawa *et al.* [131] and Weeks *et al.* [288]. Rijndael was found to be superior compared to other algorithms in [131] whereas the study of [288] refused to put algorithms in any particular order, but the values seem to support both Rijndael and Serpent. Rijndael was found superior to Serpent in a study [176] that appeared after the decision of AES.

A multitude of ASIC implementations have been published since the selection of Rijndael as AES. Most of them are iterative or loop-unrolled architectures with 128-bit data paths achieving throughputs ranging from several hundred Mbps to Gbps level. Such architectures have been presented, for example, by Alam *et al.* [4], Hsiao *et al.* [128], Kosaraju *et al.* [153], Kuo and Verbauwhede [157], Lutz *et al.* [176], Mangard *et al.* [179], and Su *et al.* [275]. Important aspects that have attained interest in the papers are methods to combine encryption and decryption with key agility and support for different key sizes. Alam *et al.* [4] presented an ASIC-based implementation supporting the full AES (all key sizes with encryption and decryption) and additional plaintext sizes of 192 and 256 bits. They used composite fields in the implementation. Hsiao *et al.* [128] optimized logic functions of all transformations of AES and described an ASIC implementation supporting encryption and decryption with key agility. Kosaraju *et al.* [153] presented a straightforward ASIC implementation which achieves full key agility. Kuo and Verbauwhede [157] presented a fully key agile implementation of the full AES. Their design uses S-boxes implemented with combinatorial logic without composite fields and the data path is 256 bits. Special attention was given for an efficient implementation of KeySchedule($\cdot$). The implementation of [157] was later improved in [282] which also studied power consumption. Lutz *et al.* [176] compared implementations of Rijndael and Serpent as mentioned above. Mangard *et al.* [179] presented a highly regular and scalable iterative ASIC implementation for both encryption and decryption with full key agility and support for the CBC mode of operation. Su *et al.* [275] implemented the full AES with key agility by utilizing composite field $\mathbb{F}_{(2^4)^2}$ with a normal basis. Next, two interesting special cases, very low cost and high throughput implementations, are discussed.

Low cost, meaning low area and/or power consumption, is required in many applications, including smart cards, home consumer electronics, and mobile devices, and there is an obvious need for minimizing the cost of AES implementations. As discussed earlier, the key techniques for achieving low cost are data path folding and composite fields for the S-boxes. The first implementation aiming to a small area was presented by Satoh *et al.* [254]. They presented both small and high throughput ASIC implementations which use the composite field $\mathbb{F}_{((2^2)^2)^2}$ in SubBytes($\cdot$). They also presented how encryption and decryption

Table 5.1: AES implementations. The six implementations on the top were included in **I** and the implementations from **II** are in the bottom.

Reference	Year	Device	Area requirements[1]	Features[2]	T (Mbps)
Chodowiec [61]	2003	Spartan-II 30-6	222 sl., 3 BRAM	EDK1	166
Hodjat [125]	2004	Virtex-II Pro 20-7	9446 sl.	EK1	21640
Pramstaller [228]	2004	Virtex-E 1000-8	1125 sl.	EDK123	215
Rouvroy [245]	2004	Spartan-III 50-4	163 sl., 3 BRAM	EDK1	208
Zambreno [304]	2004	Virtex-II 4000	16938 sl.	EK1	23570
Zhang [307]	2004	Virtex-E 1000-8	11022 sl.	EDK1	21556
Alam [4]	2007	0.18 μm CMOS	21000 gt.	EDK123M	384
Brokalaikis [37]	2005	Virtex-II 2000-5	1122 sl., 8 BRAM	EK1	1941
Bulens [42]	2008	Virtex-5	400 sl.	EK1	4100
Charot [55]	2003	Apex 20	——	EK1	10800
Chaves [56]	2006	Virtex-II Pro 20-7	3513 sl., 80 BRAM	ED1	34760
Chaves [56]	2006	Virtex-II Pro 20-7	515 sl., 12 BRAM	ED1	2332
Drimer [86]	2008	Virtex-5 SX95T-3	428 sl., 80 BRAM, 160 DSP	E1	55000
Feldhofer [96]	2005	0.35 μm CMOS	3400 gt.	EDK1	9.9
Good [103]	2005	Spartan-III 2000-5	17425 sl.	EDK1	25107
Good [103]	2005	Virtex-E 2000-8	16693 sl.	EDK1	23654
Good [104]	2006	Spartan-II 15-6	122 sl., 2 BRAM	EDK1	2.18
Good [105]	2007	Spartan-III 4000-5	20720 sl.	EDK1	30835
Good [105]	2007	Virtex-II 8000-5	31674 sl.	EDK1	28526
Hämäläinen [119]	2006	0.13 μm CMOS	3100 gt.	EK1	121
Hsiao [128]	2005	0.18 μm CMOS	18540 gt.	EDK1	1500
Hwang [130]	2006	0.18 μm CMOS	199000 gt.	EK1	3840
Hwang [130]	2006	0.18 μm CMOS	596000 gt.	EK1C	990
Ichikawa [131]	2000	0.35 μm CMOS	612834 gt.	EDK1	1950.03
Kosaraju [153]	2006	0.35 μm CMOS	——	EDK1	232.7
Kotturi [154]	2005	Virtex-II Pro 70-7	5408 sl., 200 BRAM	EK1	29770
Kuo [157]	2001	0.18 μm CMOS	173000 gt.	EK123M	1820
Liberatori [169]	2007	Spartan-III 200	822 sl.	EK1	224.12
Lutz [176]	2002	0.6 μm CMOS	$\sim$300000 trans.	EDK1	2260
Mangard [179]	2003	0.6 μm CMOS	10799 gt.	EDK1	128
Morioka [201]	2004	0.13 μm CMOS	167566 gt.	E1	11600
Satoh *et al.* [254]	2001	0.11 μm CMOS	21337 gt.	EDK1	2609.11
Satoh *et al.* [254]	2001	0.11 μm CMOS	5400 gt.	EDK1	311.09
Satoh [253]	2006	0.13 μm CMOS	297542 gt.	EK123G	42670
Su [275]	2003	0.35 μm CMOS	58430 gt.	EDK123	2008
Verbauwhede [282]	2003	0.18 μm CMOS	173000 gt.	EK123M	2290
Zhou [309]	2007	Virtex-4 LX40-12	16396 sl.	EK1G	20608
II	2003	Virtex-E 1000-8	11719 sl.	EK1	16540
II	2003	Virtex-II 2000-5	10750 sl.	EK1	17800

[1] sl. = slice, gt. = gate, BRAM = BlockRAM, DSP = DSP block, trans. = transistor

[2] E = Encryption, D = Decryption, K = Key agile, 1 = AES-128, 2 = AES-192, 3 = AES-256, G = Galois Counter Mode, M = Also 192 and 256-bit M, C = Countermeasures against DPA

can be compactly combined in SubBytes(·) and MixColumns(·). Feldhofer *et al.* [96] implemented key agile AES-128 encryption and decryption with only 3400 gates with throughput of nearly 10 Mbps by using an 8-bit data path. Hämäläinen *et al.* [119] introduced an 8-bit implementation of the AES-128 encryption using parallel AES round and KeySchedule(·, ·) and they achieved a throughput of 121 Mbps with only 3100 gates. This represents the smallest published ASIC implementation. However, it must be emphasized when compared to [96] that [119] does not include decryption. A review of low-cost AES implementations was presented in [120] which concluded that the best architectures for wireless sensor networks requiring only low encryption speed with very low cost are the 8-bit dedicated architectures from [96, 119].

Throughputs exceeding 10 Gbps are required in heavily-loaded servers and optical network switches, for example, and fully unrolled and pipelined architectures are therefore of interest. The implementation of Rudra *et al.* [247] was the first one to utilize composite fields. They implemented AES so that the entire AES was computed in the composite field $\mathbb{F}_{(2^4)^2}$, but they provided only estimates for speed and area of such an implementation. Morioka and Satoh [201] presented that a throughput of over 11 Gbps can be reached even with an iterative AES implementation by utilizing combinatorial T-boxes derived directly from the truth-table of the S-box. As a consequence, their implementation supports also the CBC mode of operation with maximal throughput. Satoh [253] presented an implementation supporting all key sizes with a throughput of over 40 Gbps for AES-128. The design also implements all logic required in the GCM (Galois Counter Mode) mode of operation [209], *i.e.*, also a 128-bit finite field multiplier is included. Hodjat and Verbauwhede [126] presented area-throughput tradeoffs for fully unrolled ASIC-based architectures achieving throughputs of over 30 Gbps. They concluded that the best results are achieved by using composite fields for SubBytes(·) with an offline KeySchedule(·) unit. They did not present any exact implementation results but showed that throughputs ranging from 30 Gbps to nearly 70 Gbps are achievable with 150000-275000 gates.

As discussed in Sec. 3.4, side-channel attacks form a serious threat in certain applications and countermeasures are thus required. Hwang *et al.* [130] presented an AES coprocessor using differential logic style which aims to an equal power consumption regardless of the operation being executed and thwarts DPA attacks as a result. They showed how differential logic style can make DPA attacks considerably more difficult, but with relatively high costs in area, power consumption, and throughput. They claimed that their coprocessor is the first practical implementation that is resistant against DPA.

5.3.2 FPGA Implementations

FPGAs are appealing platforms for cryptographic algorithms as discussed in Sec. 3.2 and, therefore, numerous FPGA-based implementations of AES have been published. Three fastest FPGA implementations included in **I** were presented by Hodjat and Verbauwhede [125], Zambreno *et al.* [304], and Zhang *et al.* [307]. They all used full unrolling and pipelining and utilized composite fields in SubBytes(·) [125, 304, 307]. High throughput with FPGAs has been the target in numerous other publications, too. Charot *et al.* [55] presented a fully key agile AES-128 implementation which supports the CTR mode of op-

eration on an Altera APEX FPGA and studied effects of pipelining. Good and Benaissa [103] presented implementations where throughput was maximized by carefully balancing pipeline registers so that the critical path was formed by 3 LUTs. The implementations were optimized further in [105] where the critical path was reduced to only 2 LUTs. This required careful design and floorplanning in order to prevent routing delays from degrading the results. As a consequence, they achieved a throughput of over 30 Gbps even in a low-cost Spartan-III FPGA [105]. Kotturi *et al.* [154] presented a straightforward fully unrolled and aggressively pipelined implementation of AES-128 encryption using BlockRAMs. Brokalakis *et al.* [37] presented a straightforward, but efficient, iterative implementation of AES-128 encryption using BlockRAMs for S-boxes. Chaves *et al.* [56] presented folded and fully unrolled FPGA-based AES implementations utilizing T-boxes implemented in BlockRAMs. Zhou *et al.* [309] presented an AES implementation in the GCM mode of operation and they optimized their implementation for 4-to-1-bit LUTs.

As discussed in Sec. 3.2.1, the recent FPGA families have introduced several new features, such as larger memories and granularity, and they have changed the ways how AES can be implemented. Utilization of the new FPGA architectures has been studied in two very recent papers by Bulens *et al.* [42] and Drimer *et al.* [86]. Bulens *et al.* [42] studied the effects of the new Virtex-5 architecture for AES implementations and they concluded that the new 6-bit LUT structure is well-suited for implementing SubBytes$(\cdot)$. A straightforward implementation of the S-box directly from the truth-table requires only 32 6-bit LUTs whereas it requires 144 4-bit LUTs [42]. Hence, the increase in granularity has positive effects for AES implementations. Drimer *et al.* [86] presented an FPGA implementation which extensively utilizes BlockRAMs for the T-boxes and DSPs for AddRoundKey$(\cdot,\cdot)$. As a result, their fully unrolled implementation of AES-128 encryption using 80 BlockRAMs and 160 DSPs requires only 428 slices and achieves a throughput of 55 Gbps [86]. This represents the fastest FPGA-based implementation currently available in the literature.

Although FPGAs are generally unsuitable for low-cost applications because of their high power consumption compared to ASICs, several low-cost FPGA implementations of AES have been presented. Three smallest implementations considered in **I** were presented by Chodowiec and Gaj [61], Pramstaller and Wolkerstorfer [228] and Rouvroy *et al.* [245]. Only slices were used in [228] whereas [61] used BlockRAMs to store the State and the S-boxes and [244] used them to implement the T-boxes. Liberatori *et al.* [169] presented a straightforward implementation with a 64-bit data path that requires fewer slices than [228] and no BlockRAMs, but it supports only AES-128 encryption. To the author's knowledge, the smallest FPGA-based AES implementation was introduced by Good and Benaissa [103] and it was later considered in more detail in [104]. In the architecture, the S-box was implemented with combinatorial logic and other operations were implemented with a single multiplier-accumulator in $\mathbb{F}_{2^8}$ [103, 104]. The implementation utilized an 8-bit data path and optimized instruction sets and microcodes.

5.3.3 Comparisons

Difficulty of comparing published AES implementations is a problem that is generally recognized in the community. In the author's opinion, the most thor-

ough discussion on aspects related to these difficulties was provided by Drimer *et al.* in [86] and, hence, the following discussion is based mainly on their paper. The most obvious problem is that published AES implementations commonly implement different functions (different key sizes, encryption and/or decryption, inclusions of KeySchedule$(\cdot,\cdot)$, modes of operation, *etc.*). The large range of target applications and implementation platforms, ASICs with different CMOS (Complementary Metal Oxide Semiconductor) processes and various FPGAs, makes comparison very challenging. The lamentable fact that authors sometimes left out necessary information, such as speed grades or sizes of devices, makes comparison even more difficult. Even though functions were the same and all information was available, fair comparison of FPGA implementations is not easy because, as Drimer *et al.* concluded, resources combining several smaller resources, such as slices, ALMs, *etc.*, are not good metrics for comparisons for three reasons [86]:

— Their "definitions" differ among device families, *e.g.*, slices in Virtex families consist of different numbers of LUTs with different sizes; see Sec. 3.2.

— It cannot be differentiated whether only part or all of the resources are used, *e.g.*, one can use only one LUT and no registers or all LUTs and registers from a slice or an ALM.

— Listing and comparing only the number of such resources neglects the number of other blocks, *e.g.*, embedded memory or DSP blocks. Hence, metrics such as throughput per slice are seldom sufficient for fair comparison.

The same points also cause problems when LUTs are considered [86]. Publishing source codes would help in comparisons because the same target devices and design softwares could be used [86]. However, it would not solve problems entirely because authors would probably—even without intention—optimize their own designs more carefully [86].

Hence, unconditionally fair comparison is impossible without a specific application in mind [86] and implementations presented in Table 5.1 should not be compared one-to-one only based on the values given in the table. For instance, the implementation of [86] using numerous BlockRAMs and DSPs but only few slices can be the most useful if logic resources are scarce, but BlockRAMs and DSPs are unused [86]. If the situation is vice versa, other architectures, such as [105, 304, 307], are more feasible.

As discussed, difficulty of fair comparison is an issue that requires attention. **I** includes one of few efforts for providing fair(er) comparisons. BlockRAMs and slices were mapped to a common quantity by utilizing equations from [248] which helped to compare designs implemented with older Virtex family FPGAs. However this method does not provide unconditionally fair comparisons either, because it does not differentiate between different utilization factors inside slices or BlockRAMs and also neglects the subjective factor that in certain applications BlockRAMs can indeed be considered free. Values obtained similarly as in **I** are not included in Table 5.1 because the considerably wider scope of target devices makes the use of such values impossible. Notice that the above discussion on the difficulty of comparisons is valid also for ECC implementations discussed in Ch. 6.

Despite the above, certain implementations which represent the state-of-the-art of AES implementation are raised above others and discussed next. These are discussed in contexts of different target applications in order to provide fairer comparison. However, it should be noted that some of the designs discussed above, but which are not given in the following selections, may have merits which make them the best suitable solutions for certain applications.

If an application requires a low-cost ASIC implementation, then implementations by either Feldhofer *et al.* [96] or Hämäläinen *et al.* [119] should be preferred so that [96] should be selected if both encryption and decryption are needed, [119] otherwise. When low-cost FPGA implementations are required, then one of the implementations by Chodowiec and Gaj [61], Good and Benaissa [104], Liberatori *et al.* [169], or Rouvroy *et al.* [245] should be chosen depending on the target device. Of these, the smallest one was presented in [104] and it does not set any specific requirements for the target device. For very high throughput ASIC implementations, implementation techniques proposed by Hodjat and Verbauwhede [126], Morioka and Satoh [201], or Satoh [253] should preferred depending on which modes of operations are required. In FPGAs, the best high throughput architectures are available in the papers presented by Good and Benaissa [105], Chaves *et al.* [56], and Drimer *et al.* [86], and their mutual superiority depends on how much BlockRAMs and DSPs are available on the target device. If support for the full AES is needed in ASICs, the design presented by Su *et al.* [275] is the best option. The only design supporting the full AES that has been targeted to FPGAs was presented by Pramstaller and Wolkerstorfer [228]. When pure speed is considered, speedups of several tens of times can be achieved compared to the fastest software implementation by Lipmaa [172].

The above review and comparisons demonstrate that AES is suitable for various environments and means for its implementation for nearly all imaginable applications exist. Consequently, the research field of AES implementation can be considered mature.

Chapter 6

Elliptic Curve Cryptography

This chapter discusses both ECC algorithms and implementations. The emphasis is, again, on issues related to hardware implementation and, especially, FPGAs. The objectives of this chapter are to provide an overview of ECC, to present details of algorithms used in the publications, **III**–**XI**, and to review existing literature on ECC implementations.

Many of the publications, namely **IV**–**IX**, were written during the PLA project. The PLA is a countermeasure against Denial-of-Service (DoS) attacks in computer networks [46, 47], *e.g.*, in the Internet. A cryptographic signature is attached to each packet and it is verified from node to node when the packet proceeds through the network [46, 47]. Hence, the computational requirements are very high making hardware acceleration essential. ECC and Koblitz curves are used in the PLA in order to provide short signatures and adequate performance. Hardware implementations for the PLA are presented in the publications of this thesis, and a comparable software implementation is presented in [40].

This chapter is organized as follows. Preliminaries of ECC are presented in Sec. 6.1 followed by detailed descriptions of algorithms used in ECC in Secs. 6.2 and 6.3. A special class of elliptic curves called Koblitz curves, which has been extensively used in the publications, is discussed in Sec. 6.4. Review of existing literature on hardware implementations of ECC is given and implementations of this thesis are compared to them in Sec. 6.5.

6.1 Preliminaries

Theory of elliptic curves is rich and deep. Elliptic curves have been studied by mathematicians over a hundred years [122]. They arise naturally in many mathematical problems but were considered only as a beautiful branch of mathematics with little practical use. However, in the recent decades elliptic curves have been used in solving a variety of problems. They have been used, *e.g.*, in factoring integers [165] and proving Fermat's Last Theorem [289]. In 1985, Neal Koblitz [146] and Victor Miller [194] independently proposed the use of elliptic curves in public-key cryptography after which an enormous amount of work has been done on elliptic curves in academia.

An elliptic curve is defined as the set of solutions for the following so-called affine version of the Weirstraß equation [32]:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \tag{6.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$.

A point $P = (x, y)$ is on the curve E if it satisfies (6.1). Also a point called the point at infinity, denoted by $\mathcal{O}$, is considered as a point on E. The origin of the name is that $\mathcal{O}$ can be thought of as lying infinitely far up the y-axis when curves are defined over real numbers [32]. Hence, the set of points on E defined over $\mathbb{F}$ is

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \mathcal{O}. \tag{6.2}$$

It is possible to define an addition of two points, $P_1, P_2 \in E(\mathbb{F})$, and produce a third point, $P_3 = P_1 + P_2 \in E(\mathbb{F})$, by using arithmetic operations in $\mathbb{F}$. This addition is called point addition. The set $E(\mathbb{F})$ forms an additive Abelian group with the addition rule and $\mathcal{O}$ serves as the identity element; see Sec. 4.1. [122]

Elliptic curve point multiplication[1] is defined over the Abelian group and it is given as follows

$$Q = kP = \underbrace{P + P + \ldots + P}_{k \text{ times}} \tag{6.3}$$

where $Q, P \in E(\mathbb{F})$ and k is a positive integer. The point P is called the base point and Q is the result point.

ECDLP is the problem of finding k if P and Q are known. Methods for solving ECDLP are not considered in this thesis; however, extensive reviews can be found in [32] and Part V of [68], for example.

The order of a point P, $\mathrm{ord}(P)$, is the smallest integer, t, for which $tP = \mathcal{O}$. The integer k in (6.3) should be chosen so that it fulfills

$$1 < k \leq \mathrm{ord}(P) - 1. \tag{6.4}$$

When curves over $\mathbb{F}_q$ are considered, the length of the binary expansion of k, ℓ, is bounded by $\ell \leq \lceil \log_2 q \rceil$. When $q = 2^m$, this gives $\ell \leq m$. [122]

Elliptic curve point multiplication, (6.3), is hierarchical in nature. The operation decomposes into three levels as shown in Fig. 6.1. This thesis concentrates in improving the two highest levels of the hierarchy and offers only minor contributions to the level of finite field arithmetic.

The lowest level, finite field arithmetic, was discussed in detail in Ch. 4, and it is not considered further here. Point operations are computed by using field arithmetic. Computational costs of point operations, *i.e.*, the number of required field operations, can be reduced mainly by representing points in different coordinates. When a point is represented with two coordinates as $P = (x, y)$ as above, it is said to be in affine coordinates, or $\mathcal{A}$ for short. If inversions in $\mathbb{F}$ are considerably more expensive than multiplications (which usually is the case), it is more efficient to use projective coordinates which are considered in more detail in Sec. 6.2 which discusses elliptic curve arithmetic, *i.e.*, point operations.

The highest level, elliptic curve point multiplication, is computed by using point operations and these algorithms are discussed in Sec. 6.3. If the base point

[1] The term scalar multiplication is also commonly used in the literature

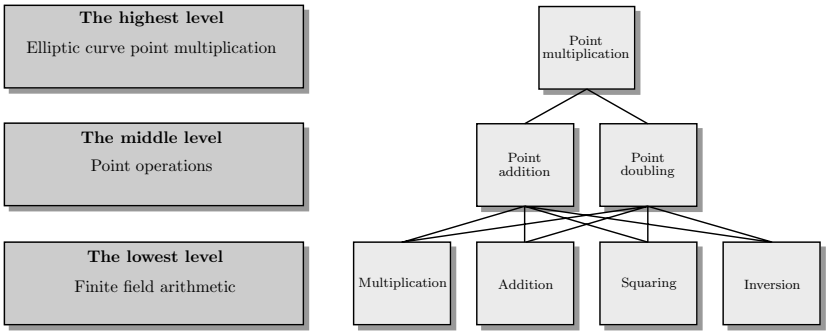

Figure 6.1: Hierarchy of point multiplication is depicted on the left-hand side. Typical construction of operations in this hierarchy is shown on the right-hand side. Notice, however, that all operations may not be needed or implemented in all algorithms, *e.g.*, squaring may not be included, or additional operations can be used, *e.g.*, point halvings or Frobenius maps. Inversion can be interpreted also as a separate level between the lowest level and the middle level because it can be computed with a series of other finite field operations.

P is fixed, *i.e.*, P is the same for all point multiplications, the computation can be accelerated considerably with precomputations involving P. These methods will be discussed in Sec. 6.3.1. Certain applications, including digital signature algorithms, require computing sums of point multiplications, such as $k_{(1)}P_{(1)} + k_{(2)}P_{(2)}$. There are methods for speeding up these so-called multiple point multiplications and they are discussed in Sec. 6.3.2. A family of curves on which point multiplication can be computed considerably faster than on general curves is called Koblitz curves and they are considered in detail in Sec. 6.4.

Hyper-Elliptic Curve Cryptography (HECC) [147], a generalization of ECC, achieves a similar level of security than ECC with smaller finite fields, thus, resulting in faster finite field arithmetic. However, point operations are more complex. Recent studies [249, 250] suggest that HECC is slower than ECC with similar levels of security. HECC is not studied in detail in this thesis; the reader should consult [68], for example, for details.

6.2 Arithmetic on Elliptic Curves

The following discusses arithmetic on elliptic curves, the middle level of the hierarchy of Fig. 6.1. The discussion is restricted to so-called non-supersingular elliptic curves over $\mathbb{F}_{2^m}$, because they are used in the publications. Non-supersingular curves are preferred also in other published ECC implementations. Supersingular curves, which were preferred previously for implementation efficiency reasons, should be avoided because ECDLP can be reduced to a DLP in an extension of the underlying field [188]. Formulae for curves over $\mathbb{F}_p$ are available in [84, 122], for example. The following formulae and notations are presented as in [84].

Non-supersingular curves over $\mathbb{F}_{2^m}$ are defined by

$$E : y^2 + xy = x^3 + a_2x^2 + a_6 \tag{6.5}$$

with $a_2, a_6 \in \mathbb{F}_{2^m}$ so that $a_6 \neq 0$. The implementations of this thesis mainly focus on elliptic curves standardized by NIST in FIPS 186-2 [205]. Specifically, the curves NIST B-163, K-163, K-233, and K-283 were considered in the publications and they all have the form of (6.5). The K-curves are Koblitz curves, and they will be discussed in more detail in Sec. 6.4.

Elliptic curve point multiplication, (6.3), is typically computed using two principal operations on the curve; namely, point addition and point doubling. Let P_1 and P_2 be two points in $E(\mathbb{F})$. Point addition is $P_3 = P_1 + P_2$ with $P_1 \neq \pm P_2$ and it results in $P_3 \in E(\mathbb{F})$. Point doubling refers to a case where $P_1 = P_2$, and it is denoted by $P_3 = 2P_1$. Computational costs of elliptic curve operations depend on the number of operations required in $\mathbb{F}$. This discussion focuses on curves over $\mathbb{F}_{2^m}$. The following notations are used for operations in $\mathbb{F}_{2^m}$: **I** denotes inversion, **M** multiplication, **S** squaring, and **A** addition.

6.2.1 Affine Coordinates

Point addition $P_3 = P_1 + P_2 = (x_3, y_3)$ for points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ such that $P_1 \neq \pm P_2$ is given by

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}, \quad x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2, \quad y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \ . \tag{6.6}$$

Point doubling $P_3 = 2P_1 = (x_3, y_3)$ for a point $P_1 = (x_1, y_1)$ is given by

$$\lambda = x_1 + \frac{y_1}{x_1}, \quad x_3 = \lambda^2 + \lambda + a_2, \quad y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \ . \tag{6.7}$$

Thus, point addition costs $\mathbf{I} + 2\mathbf{M} + \mathbf{S} + 8\mathbf{A}$ and point doubling costs $\mathbf{I} + 2\mathbf{M} + \mathbf{S} + 6\mathbf{A}$ which are almost the same because additions are cheap. Given a point $P = (x, y)$, its point negation, $-P$, is given by $(x, x + y)$ which costs **A**. [84]

As discussed in Ch. 4, inversions are more expensive than other field operations; *e.g.*, an Itoh-Tsujii inversion requires $9\mathbf{M} + 162\mathbf{S}$ in $\mathbb{F}_{2^{163}}$ [136]. Inversions can be avoided in point addition and point doubling by using projective coordinates with the expense of more multiplications. Projective coordinates usually results in considerable speed enhancements in practical cryptosystems. Two projective coordinate systems used in the publications are discussed in the following.

6.2.2 Standard Projective Coordinates

By using standard projective coordinates, $\mathcal{P}$, a point is represented with the triple (X, Y, Z) so that it represents the point $(X/Z, Y/Z)$ in $\mathcal{A}$ if $Z \neq 0$ and $\mathcal{O} = (0, 1, 0)$ otherwise.

Point addition $P_3 = P_1 + P_2$, where $P_i = (X_i, Y_i, Z_i)$ and $P_1 \neq \pm P_2$, is given by:

$$\begin{gathered} A = Y_1 Z_2 + Z_1 Y_2, \quad B = X_1 Z_2 + Z_1 X_2, \quad C = B^2, \\ D = Z_1 Z_2, \quad E = (A^2 + AB + a_2 C)D + BC, \\ X_3 = BE, \quad Y_3 = (AX_1 + Y_1 B)CZ_2 + (A + B)E, \quad Z_3 = (BC)D. \end{gathered} \tag{6.8}$$

Point doubling $P_3 = 2P_1$ is given by:

$$\begin{gathered} A = X_1^2, \quad B = A + Y_1 Z_1, \quad C = X_1 Z_1, \\ D = C^2, \quad E = B^2 + BC + a_2 D, \\ X_3 = CE, \quad Y_3 = (B + C)E + A^2 C, \quad Z_3 = CD. \end{gathered} \tag{6.9}$$

Neither of the operations requires inversions, and the costs are $16\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ and $8\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$ for point addition and point doubling, respectively. Point negation is given by $(X, X + Y, Z)$, and it costs $\mathbf{A}$. It depends on the cost of an inversion whether this representation is faster than $\mathcal{A}$. [84]

A very efficient algorithm for point multiplication is achievable in $\mathcal{P}$ by adapting Montgomery's idea from [197] to curves with the form of (6.5) as presented by Lopéz and Dahab in [175]. This method, henceforth referred to as Montgomery point multiplication, relies on the fact that y-coordinate is not needed during point multiplication because it can be recovered in the end [175, 197]. The method will be discussed in more detail in Sec. 6.3, but the point operation level formulae are given in the following. The x-coordinate of point addition, $P_1 + P_2$, is obtained with the following formulae [175]:

$$Z_3 = (X_1 Z_2 + X_2 Z_1)^2, \quad X_3 = x Z_3 + X_1 Z_2 X_2 Z_1 \tag{6.10}$$

where x is the x-coordinate of the base point P. The x-coordinate of point doubling, $2P_1$, is given by the formulae [175]:

$$X_3 = X_1^4 + a_6 Z_1^4 = (X_1^2 + \sqrt{a_6} Z_1^2)^2, \quad Z_3 = X_1^2 Z_1^2. \tag{6.11}$$

The cost of (6.10) is $4\mathbf{M} + \mathbf{S} + 2\mathbf{A}$ while (6.11) costs $2\mathbf{M} + 3\mathbf{S} + \mathbf{A}$ if $\sqrt{a_6}$ is precomputed [175]. The y-coordinate is recovered by computing $x_1 = X_1/Z_1$ and $x_2 = X_2/Z_2$ and by using the formula [175]:

$$y_1 = \frac{(x_1 + x)\left((x_1 + x)(x_2 + x) + x^2 + y\right)}{x} + y \tag{6.12}$$

where (x, y) is the base point P. The y-coordinate can be recovered with the cost of $\mathbf{I}+10\mathbf{M}+\mathbf{S}+6\mathbf{A}$ [175]. Standard projective coordinates, $\mathcal{P}$, and Montgomery point multiplication were used in **III** and **IV**.

6.2.3 López-Dahab Coordinates

When López-Dahab coordinates [174], $\mathcal{LD}$, are in use, a point is represented with the triple (X, Y, Z) which represent the point $(X/Z, Y/Z^2)$ in $\mathcal{A}$ when $Z \neq 0$ and $\mathcal{O} = (1, 0, 0)$ otherwise.

Point addition, $P_3 = P_1 + P_2$, is carried out with the formulae [124]

$$\begin{gathered} A = X_1 Z_2, \quad B = X_2 Z_1, \quad C = A^2, \quad D = B^2, \quad E = A + B, \\ F = C + D, \quad G = Y_1 Z_2^2, \quad H = Y_2 Z_1^2, \quad I = G + H, \quad J = IE, \\ Z_3 = F Z_1 Z_2, \quad X_3 = A(H + D) + B(C + G), \\ Y_3 = (AJ + FG)F + (J + Z_3)X_3, \end{gathered} \tag{6.13}$$

and point doubling, $P_3 = 2P_1$, is given by the following formulae [161]:

$$\begin{gathered} A = X_1 Z_1, \quad B = X_1^2, \quad C = B + Y_1, \quad D = AC, \\ Z_3 = A^2, \quad X_3 = C^2 + D + a_2 Z_3, \quad Y_3 = (Z_3 + D)X_3 + B^2 Z_3. \end{gathered} \tag{6.14}$$

Table 6.1: The costs of point operations

Operation	P_1	P_2	P_3	Cost
$P_3 = P_1 + P_2$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathbf{I} + 2\mathbf{M} + \mathbf{S} + 8\mathbf{A}$
$P_3 = 2P_1$	$\mathcal{A}$	-	$\mathcal{A}$	$\mathbf{I} + 2\mathbf{M} + \mathbf{S} + 6\mathbf{A}$
$P_3 = -P_1$	$\mathcal{A}$	-	$\mathcal{A}$	$\mathbf{A}$
$P_3 = P_1 + P_2$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$16\mathbf{M} + \mathbf{S} + 6\mathbf{A}$
$P_3 = 2P_1$	$\mathcal{P}$	-	$\mathcal{P}$	$8\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$
$P_3 = -P_1$	$\mathcal{P}$	-	$\mathcal{P}$	$\mathbf{A}$
$P_1 \mapsto P_3$	$\mathcal{P}$	-	$\mathcal{A}$	$\mathbf{I} + 2\mathbf{M}$
$P_3 = P_1 + P_2$ (x-coord.)	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$4\mathbf{M} + \mathbf{S} + 2\mathbf{A}$
$P_3 = 2P_1$ (x-coord.)	$\mathcal{P}$	-	$\mathcal{P}$	$2\mathbf{M} + 3\mathbf{S} + \mathbf{A}$
$P_1 \mapsto P_3$ (and y-coord.)	$\mathcal{P}$	-	$\mathcal{A}$	$\mathbf{I} + 10\mathbf{M} + \mathbf{S} + 6\mathbf{A}$
$P_3 = P_1 + P_2$	$\mathcal{LD}$	$\mathcal{LD}$	$\mathcal{LD}$	$13\mathbf{M} + 4\mathbf{S} + 9\mathbf{A}$
$P_3 = 2P_1$	$\mathcal{LD}$	-	$\mathcal{LD}$	$5\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$
$P_3 = P_1 + P_2$	$\mathcal{LD}$	$\mathcal{A}$	$\mathcal{LD}$	$9\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$
$P_3 = -P_1$	$\mathcal{LD}$	-	$\mathcal{LD}$	$\mathbf{M} + \mathbf{A}$
$P_3 = P_1 \mapsto P_3$	$\mathcal{LD}$	-	$\mathcal{A}$	$\mathbf{I} + 2\mathbf{M} + \mathbf{S}$

The costs of the above point addition and point doubling are $13\mathbf{M}+4\mathbf{S}+9\mathbf{A}$ and $5\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$, respectively. Point negation is given by $(X, XZ + Y, Z)$ which costs $\mathbf{M} + \mathbf{A}$. The attractiveness of $\mathcal{LD}$ coordinates is based on a very efficient point addition when the point P_2 is represented in $\mathcal{A}$. This is called mixed coordinate point addition and the formulae for $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$ are given by [3]

$$\begin{aligned} A = Y_1 + y_2 Z_1^2, \quad B = X_1 + x_2 Z_1, \quad C = BZ_1, \\ Z_3 = C^2, \quad D = x_2 Z_3, \\ X_3 = A^2 + C(A + B^2 + a_2 C), \\ Y_3 = (D + X_3)(AC + Z_3) + (y_2 + x_2) Z_3^2. \end{aligned} \tag{6.15}$$

This has the cost of $9\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$. Further reductions by $\mathbf{M}$ if $a_2 \in \{0, 1\}$ and by $\mathbf{A}$ if $a_2 = 0$ occur on Koblitz curves; see Sec. 6.4. The López-Dahab coordinates, $\mathcal{LD}$, were used in **IV**–**VII**, **X**, and **XI**.

Table 6.1 summaries all above-discussed point operations and their costs.

6.3 Elliptic Curve Point Multiplication

Elliptic curve point multiplication is computed with algorithms which are analogous to those used in exponentiation. A comprehensive survey of exponentiation is provided in [108]. This section gives a review of a few most commonly used algorithms for elliptic curve point multiplication by concentrating on the algorithms used in the publications.

A standard method for computing (6.3) is called binary method[2] where the integer k is represented with binary expansion as

$$k = \sum_{i=0}^{\ell-1} k_i 2^i \tag{6.16}$$

[2]Also called double-and-add method (or square-and-multiply method in exponentiation).

where $k_i \in \{0, 1\}$. When binary method operates k from the right to the left, *i.e.*, from the lsb to the msb, $2^i P$ is added to the result point Q when $k_i = 1$. Consider $18P$ as an example. Binary expansion of 18 is $\langle 10010 \rangle$. One first computes $2P$ with one point doubling and adds it to Q after which $16P$ is computed with three point doublings and the result is added to Q giving $18P$. A modification which operates k from the left to the right is commonly used in practical implementations and in **IV**–**VII**, **X**, and **XI**. It is presented in Alg. 6.1 and it has the benefit that only Q needs to be accumulated while P remains unchanged.

Algorithm 6.1 Left-to-right binary method for elliptic curve point multiplication

Input: $P \in E(\mathbb{F})$, integer $k = \sum_{i=0}^{\ell-1} k_i 2^i$ where $k_i \in \{0, 1\}$
Output: $Q = kP$

```
Q ← O
for i = ℓ − 1 downto 0 do
  Q ← 2Q
  if k_i = 1 then
    Q ← Q + P
  end if
end for
return(Q)
```

Obviously, the computational cost of the binary method depends on the number of ones in the binary expansion of k. Again, $H(k)$ denotes the Hamming weight of k, *i.e.*, the number of nonzero terms in k. The computational cost of the binary method is $\ell - 1$ point doublings and $H(k) - 1$ point additions because the first point addition is simply a substitution. On average, half of the bits in a binary expansion are ones; thus, $H(k) \approx \ell/2$.

Because $H(k)$ has a direct impact on performance, it is of interest to reduce it. Signed-binary representations, $k = \sum_{i=0}^{\ell-1} k_i 2^i$ where $k_i \in \{0, \pm 1\}$, are particularly interesting because point subtractions have approximately the same cost with point additions[3] [198]. Point subtraction, $P_1 - P_2$, is simply point addition where P_2 is negated, and point negation is cheap as discussed in Sec. 6.2.

A Non-Adjacent Form[4] (NAF) is a representation that has the minimum $H(k)$ among all signed-binary representations which makes it well-suited for practical applications. When k is in NAF, two consecutive digits are never nonzero, *i.e.*, $k_i k_{i+1} = 0$ for all i. Every positive integer k has a unique NAF with a length which is at most one bit longer than its binary expansion. Most importantly, the average density of nonzero terms in NAF is 1/3 which results in $H(k) \approx \ell/3$. An NAF is constructed from a binary expansion starting from the lsb by replacing strings of j ones with $10 \ldots 0\bar{1}$, where the number of zeros is $j - 1$ and $\bar{1} = -1$.[82]

Consider $1853P$ as an example. The binary method requires seven point additions because $1853 = \langle 11100111101 \rangle$. The number of point additions and

[3] Signed-binary representations cannot be used as efficiently in exponentiation, *e.g.*, in RSA or ElGamal, unless precomputations are allowed, because divisions are considerably more expensive than multiplications.

[4] In DSP, such representations are commonly called canonic signed-digit representations.

subtractions reduces to four if 1853 is given in NAF as $\langle 100\bar{1}01000\bar{1}01\rangle$. Point multiplication requires in this case an additional point doubling but the speedup is, nonetheless, considerable.

If used naïvely, NAF doubles requirements for storage space compared to binary expansion because two bits are needed to represent one signed-bit. However, if storage space is an issue, the easily-disposable compact encoding presented in [141] allows representing NAF with at most one extra bit compared to binary expansion. The same paper suggest also a technique for selecting well-distributed random NAFs [141].

One advantage of using NAF is that $H(k)$ can be reduced without any precomputations involving the base point P. If such precomputations are allowed, $H(k)$ can be reduced by using methods which are discussed in more detail in Sec. 6.3.1. If the base point is fixed and there are no strict memory constraints, those methods can provide considerable speedups.

Computational complexity of point multiplication can be reduced also by using a DBNS in representing k. When k is represented in the DBNS, it has the form:

$$k = \sum_{i,j} k_{i,j} 2^i 3^j \tag{6.17}$$

where $k_{i,j} \in \{0, 1\}$ and i, j are non-negative integers [79, 80]. Obviously, such representations are not unique and it is possible to find representations which are very sparse, *i.e.*, $H(k)$ is small [80]. Hence, fewer point operations are required but extra computation is needed in conversions to the DBNS. In [79, 80], Dimitrov *et al.* proposed the use of the DBNS in DSP applications and exponentiation. The idea has been adapted to elliptic curve point multiplication in [66, 77, 78].

Binary method has the weakness that, if point additions and point doublings are distinguishable, it may be possible to retrieve confidential information with side-channel attacks because point additions are performed only when $k \neq 0$. This can be avoided by using a structure called Montgomery's ladder [197] where point doubling and point addition are both performed for each bit. Montgomery point multiplication is outlined in Alg. 6.2 [197]. As mentioned in Sec. 6.2.2, Montgomery point multiplication can be adapted to curves with a form of (6.5) as shown by López and Dahab in [175] and it results in a very efficient point multiplication algorithm. Point multiplication is carried out in $\mathcal{P}$ without infor-

Algorithm 6.2 Point multiplication using Montgomery's ladder

Input: $P \in E(\mathbb{F})$, integer $k = \sum_{i=0}^{\ell-1} k_i 2^i$ where $k_i \in \{0, 1\}$ and $k_{\ell-1} = 1$
Output: $Q = kP$

```
P1 ← P and P2 ← 2P
for ℓ − 2 downto 0 do
  if k_i = 0 then
    P2 ← P1 + P2 and P1 ← 2P1
  else
    P1 ← P1 + P2 and P2 ← 2P2
  end if
end for
return(Q = P1)
```

mation about the y-coordinate, *i.e.*, only X and Z are updated. In the beginning P_1 and P_2 are computed from P with $2\mathbf{S}+\mathbf{A}$. Point multiplication is computed with Alg. 6.2 so that point addition and point doubling are performed for all bits with (6.10) and (6.11), respectively, requiring together only $6\mathbf{M}+4\mathbf{S}+3\mathbf{A}$ per iteration. The y-coordinate is recovered in the end by computing (6.12). [175]

Computation of the binary method can be accelerated also by replacing point doublings with more efficiently computable operations. Point doublings can be replaced with very cheap Frobenius maps on Koblitz curves as will be discussed in Sec. 6.4. Koblitz curves are defined over $\mathbb{F}_{2^m}$, but analogously efficiently-computable endomorphisms are used in the GLV (Gallant, Lambert, and Vanstone) method for a class of curves over $\mathbb{F}_p$ [101]. Another approach which gives smaller improvements but applies to a wider class of curves is called point halving, proposed independently by Erik W. Knudsen [145] and Richard Schroeppel (at the rump session of CRYPTO 2000). Point halving, the inverse operation of point doubling, is the operation $P_3 = \frac{1}{2}P_1$ such that $2P_3 = P_1$. After manipulations on k, computationally cheaper point halvings can be used instead of point doublings which leads to faster point multiplication [145].

6.3.1 Methods with Precomputations

The common idea in the methods presented in this section is that the computational cost of (6.3) is reduced by precomputing some data depending on P. These methods are especially useful if P is fixed because, in that case, precomputations need to be performed only once; however, considerable speedup may be achievable even though P is not fixed. Naturally, storage for precomputed data must be available. Analogous methods exist for exponentiation and the following methods are their adaptations for elliptic curves; see [82, 108] for comprehensive reviews.

The first method, which dates to 1939, is called 2^w-ary algorithm [35] where the integer k is represented in radix-2^w as

$$k = \sum_{i=0}^{\ell-1} k_i (2^w)^i \tag{6.18}$$

where $k_i \in [0, 2^w - 1]$. The precomputed points are $3P, 5P, \ldots, (2^w-1)P$. Each k_i is given in the form $k_i = 2^s u$ where u is odd; $s = w$ and $u = 0$ if $k_i = 0$ [82]. Then each k_i transforms into the operation $2^s(2^{w-s}Q + uP)$ on the curve. The computation of (6.3) requires $\ell - 1$ point doublings and on average $(2^w - 1)/2^w\lceil \ell/w \rceil - 1$ point additions.

Consider $846P$ as an example with $w = 3$. Precomputed points are $3P, 5P$, and $7P$ and k is represented as $\langle 1\,\underline{101}\,\underline{001}\,\underline{110}\rangle_2 = \langle 1516\rangle_{2^3}$. First, one sets $Q = P$ because $k_3 = 1$. Because $5 = 2^0 \cdot 5$, one computes $2^0(2^3P + 5P) = 13P$, and continues by computing $2^0(2^3(13P)+P) = 105P$ because $1 = 2^0 \cdot 1$. Finally, the result is given by $2(2^2(105P) + 3P) = 846P$ because $6 = 2^1 \cdot 3$. Hence, point multiplication requires 9 point doublings and 3 point additions excluding precomputations requiring 3 point additions and 1 point doubling.

The 2^w-ary algorithm slices k by using a fixed window. A sliding window analogue results in slightly more optimal representations because it skips consecutive zeros [82]. In the case of the above example, a sliding window with $w = 3$ results in the following windowing: $\langle 1\,\underline{101}\,00\,\underline{111}\,0\rangle_2$, and the number of point

additions reduces by one. The window methods can be used also for signed-digit representations. A generalization of NAF called width-w NAF, or NAF_w for short, gives a representation whose $H(k) \approx m/(w+1)$ with precomputed points $3P, 5P, \ldots, (2^{w-1}-1)P$ [122].

The above ideas can be used in accelerating (6.3) even if P is not fixed. The following methods are useful only with a fixed P. The simplest idea is to precompute $2^i P$ for all integers $i \in [1, m-1]$. This shortens runtime by eliminating all point doublings, but requires storage for m points. Combination of the previous idea with the 2^w-ary algorithm results in an algorithm [36], called fixed-base windowing method. The method is based on the following equation [122]:

$$kP = \sum_{i=0}^{t-1} k_i (2^{wi} P) = \sum_{j=1}^{2^w-1} \left(j \sum_{i:k_i=j} 2^{wi} P \right) \tag{6.19}$$

where $k_i \in [0, 2^w - 1]$, $t = \lceil \ell/w \rceil$ and points $2^{wi}P$ are precomputed for integers $i \in [0, t-1]$. The fixed-base windowing method computes (6.3) with approximately $2^w + t - 3$ point additions [36, 122]. The idea can be generalized to NAF which results in an average runtime of $2^{w+1}/3 + \lceil (\ell+1)/w \rceil - 2$ point additions [36, 122]. Furthermore, Montgomery's trick, discussed in Sec. 4.4, can be applied to reduce the number of inversions if $\mathcal{A}$ coordinates are in use [195].

The fixed-base comb method [170] is also a method involving precomputations with P. The integer k is represented as a binary array of w rows and $t = \lceil \ell/w \rceil$ columns. Points are precomputed for all possible values of the columns. Point multiplication operates on the array one column at a time so that a precomputed value of the column is added. Moving to the next column implies a point doubling. It requires $t-1$ point doublings and, on average, $(2^w - 1)/2^w t - 1$ point additions [122, 170]. The number of point doublings can be reduced to $\lceil t/2 \rceil - 1$ by using two precomputation tables [122, 170].

Methods for speeding up (6.3) when the integer k is fixed exist but they are not considered in this thesis because they have little practical importance in ECC. Such methods are reviewed in [82], for example.

6.3.2 Multiple Point Multiplication

Multiple point multiplication refers to a sum of point multiplications, *i.e.*,

$$Q = \sum_{i=0}^{n-1} k_{(i)} P_{(i)} \tag{6.20}$$

where $k_{(i)}$ are integers and $P_{(i)} \in E(\mathbb{F})$. Multiple point multiplications are used in various schemes including digital signatures, *e.g.*, multiple point multiplications with $n = 2$ are required in ECDSA verifications [140] and verifications of self-certified identity based signatures in the PLA require multiple point multiplications with $n = 3$ [38]. Of course, multiple point multiplications can be computed naïvely with n separate point multiplications and $n-1$ point additions combining their results. This requires $\sum_{i=0}^{n-1} H(k_{(i)}) - 1$ point additions and $\sum_{i=0}^{n-1} (\ell_{(i)} - 1)$ point doublings and it is possible to do much better.

Shamir's trick, a method attributed to Shamir by ElGamal in [92], operates in the following way when $n = 2$. Bits of $k_{(0)}$ and $k_{(1)}$ are scanned from the left

to the right and point doubling is performed when moving from a bit to another. When the present bits of $\begin{smallmatrix} k_{(0)} \\ k_{(1)} \end{smallmatrix}$ are $\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}$, $\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$, or $\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}$ points $P_{(0)}$, $P_{(1)}$, or $P_{(0)} + P_{(1)}$ are added, respectively. The point $P_{(0)} + P_{(1)}$ should be precomputed. Clearly, Shamir's trick is easy to generalize for n integers. Let $\boldsymbol{k}$ denote an array of n binary expansions and let $H_n(\boldsymbol{k})$ be its joint Hamming weight, *i.e.*, the number of nonzero columns in $\boldsymbol{k}$. Shamir's trick requires $H_n(\boldsymbol{k}) - 1$ point additions and $\ell - 1$ point doublings excluding precomputations which require $2^n - n - 1$ point additions.

As the computational cost depends on $H_n(\boldsymbol{k})$, it is of interest to reduce it once again. Because the probability of a zero bit in a binary expansion is 0.5, the probability of a zero column in an array of n integers is simply 0.5^n which gives $H_n(\boldsymbol{k}) \approx \ell(1 - 0.5^n)$. If the integers are in NAF, the probability becomes $(2/3)^n$. This can be improved because signed-binary representations are not unique. Thus, one can use signed-binary representations which maximize the number of zero columns in the array. One such representation is referred to as Joint Sparse Form (JSF) and it is unique and has the minimum $H_n(\boldsymbol{k})$ among all signed-binary representations of n integers [230, 268]. Jerome A. Solinas presented an algorithm for obtaining JSF for a pair of integers in [268]. A generalization for n integers is due to John Proos [230]. Probabilities of nonzero columns are presented in Table 6.2 with different values of n. A variant of JSF called simple JSF, which also has a minimal $H_n(\boldsymbol{k})$ but is more efficient to compute, was introduced in [110].

If n is relatively small—which usually is the case—it can be advantageous to combine window methods with multiple point multiplications. As a consequence, a large number of points need to be precomputed, but their computational cost can be reduced by eliminating inversions with Montgomery's trick [215], discussed in Sec. 4.4. Montgomery's trick provides considerable speedups even though window methods are not in use as shown in **V** for $n = 3$.

6.4 Koblitz Curves

Koblitz curves [148] are a class of non-supersingular curves defined over $\mathbb{F}_2$ by

$$E_{a_2} : y^2 + xy = x^3 + a_2x^2 + 1 \tag{6.21}$$

where $a_2 \in \{0, 1\}$. The NIST curves K-163, K-233, and K-283 [205], which are used in **IV**–**XI**, are Koblitz curves and have the above form. Koblitz curves are an attractive family of curves because Frobenius endomorphisms can replace point doublings. The Frobenius endomorphism, $\phi : E_{a_2}(\mathbb{F}_{2^m}) \to E_{a_2}(\mathbb{F}_{2^m})$, is a

Table 6.2: Probabilities of nonzero columns in n integers with different representations [39, 230]

n	Binary	NAF	JSF
1	0.5000	0.3333	—
2	0.7500	0.5555	0.5000
3	0.8750	0.7037	0.5897
4	0.9375	0.8025	0.6425
5	0.9688	0.8683	0.6727

map such that

$$\phi(x, y) = (x^2, y^2) \quad \text{and} \quad \phi(\mathcal{O}) = \mathcal{O}. \tag{6.22}$$

Frobenius maps are obviously very cheap if compared to point operations discussed in Sec. 6.2 because only two or three squarings are required depending on the coordinate system. Squarings are efficient especially in normal basis where they are simple bitvector rotations as discussed in Ch. 4. Successive applications of Frobenius maps are simply $\phi^t(x, y) = (x^{2^t}, y^{2^t})$.

It can be shown that the following equation stands for all $P \in E_{a_2}(\mathbb{F}_{2^m})$:

$$\mu\phi(P) - \phi^2(P) = 2P \tag{6.23}$$

where $\mu = (-1)^{1-a_2}$. Thus, the Frobenius map can be seen as a complex number τ satisfying

$$\mu\tau - \tau^2 = 2 \tag{6.24}$$

which has two solutions, and it is chosen that

$$\tau = \frac{\mu + \sqrt{-7}}{2}. \tag{6.25}$$

It is now possible to multiply points in $E_{a_2}(\mathbb{F}_{2^m})$ with elements of a ring of polynomials in τ with integer coefficients, given by

$$k = \sum_{i=0}^{\ell-1} k_i \tau^i \tag{6.26}$$

where k_i are integers. Obviously, it is of interest to find such a representation where the length, ℓ, and the nonzero digits, k_i, are small. [148]

It follows directly from (6.24) that every element k of the ring can be expressed in a canonical form as

$$k = k_0 + k_1\tau. \tag{6.27}$$

If an integer, k, is represented with a binary expansion, each bit k_i requires point doubling. Point addition is computed if $k_i = 1$, as discussed previously. Similarly, if k is represented with a τ-adic expansion of (6.26) where $k_i \in \{0, 1\}$, each k_i requires a Frobenius map and $k_i = 1$ yields point addition. Such representation can be found by repeatedly dividing the integer k by τ. The digits $k_i \in \{0, 1\}$ are the remainders of the divisions. This procedure is analogous with the computation of binary representation where divisions by 2 are applied. [148]

Analogously with the NAF, it is possible to represent k in τ-adic Non-Adjacent Form (τNAF) where $k_i \in \{0, \pm 1\}$ and $k_i k_{i+1} = 0$ for all i, *i.e.*, there are no adjacent nonzero digits. Algorithms for computing τNAF and width-w τNAF, or τNAF_w for short, were presented by Solinas in [264, 266, 267]. The average density of τNAF is the same as for binary NAF, *i.e.*, 1/3. The problem is, however, that the length of an expansion, ℓ, doubles to approximately $2m$ if the τNAF conversion algorithm from [264, 266, 267] is applied directly to an integer.

It is possible to circumvent this problem because the following equation stands for all $P \in E_{a_2}(\mathbb{F}_{2^m})$ [187]:

$$(\phi^m - 1)P = \phi^m P - P = P - P = \mathcal{O}. \tag{6.28}$$

This implies that if $k' \equiv k \pmod{\tau^m - 1}$, then $k'P = kP$. τNAF of the remainder of k divided by $\tau^m - 1$ has a length of approximately m which is only one half of the length of τNAF of k [264, 266, 267]. It can be proven that $k'P = kP$ stands for all $P \in E_{a_2}(\mathbb{F}_{2^m})$ also in the case $k' \equiv k \pmod{\delta}$ where $\delta = (\tau^m - 1)/(\tau - 1)$ [266, 267]. The strategy is to find $k' = k_0 + k_1\tau \equiv k \pmod{\delta}$ with the smallest possible norm and then convert it to τNAF. The result has a length of approximately m. Finding $k' \equiv k \pmod{\delta}$ can be expensive, although an efficient partial reduction algorithm was proposed in [267].

Another approach is to first convert an integer k to τNAF and then factor $(\tau^m - 1)$ out of the result. The problems in this approach are that the τNAF conversion takes longer and the non-adjacency is lost in the reduction and its recovery may be computationally demanding. This approach was outlined at least in [177]. This method yields efficient hardware implementations as shown in **VIII**.

If reduced τNAFs are used, the average computational cost of (6.3) is only $m/3$ point additions because point doublings are replaced by Frobenius maps which are almost free. This is a significant reduction compared to computations on general curves. Hence, Koblitz curves offer major benefits over general curves.

However, before fast point multiplications can be utilized, conversions need to be applied to integers which require time and resources. In order to avoid these conversions, random τ-adic expansions can be generated in a simple manner and they have been shown to be well-distributed [162]. Such representations are troublesome in certain applications including digital signatures because the binary equivalent is required which yields a need for a conversion from τ-adic to binary expansion. An efficient algorithm for the conversion from τ-adic to binary expansion and its hardware implementation were presented in **IX**.

Many methods, which were discussed earlier, can be adapted to Koblitz curves and, in some cases, even further improvements are gained compared to their general curve equivalents. τNAFs can be compressed similarly as NAFs and the method for selecting random NAFs gives almost perfectly distributed τNAFs, as well [141]. Adaptations of DBNS-related expansions to Koblitz curves were presented in **X** and in [15, 16], but practical significance of such methods was first demonstrated in **X**. Point halving gives computational advantages also on Koblitz curves even though there are no point doublings to be replaced [17, 19]. Computing one point halving reduces the number of required point additions from $m/3$ to $m/4$ [19]. Unfortunately, Montgomery point multiplication has not been adapted to Koblitz curves. Window methods, on the other hand, can be computed without requirements for storage space as presented in [216, 284] by exploiting the inexpensiveness of Frobenius maps. The technique operates so that an entire k is scanned once for every precomputed point and, hence, it requires storage only for the point currently at hand [216, 284]. The inexpensiveness of Frobenius maps was used also in **IV** allowing efficient parallelization of point multiplication. Multiple point multiplications and the idea of JSF can be used also for τ-adic representations. An algorithm for obtaining τ-adic Joint Sparse Form (τJSF) for a pair of integers was proposed by Ciet *et al.* in [65] and it was generalized for n integers by Brumley in [39]. Also τJSFs have the probabilities of nonzero columns presented in Table 6.2.

6.5 Literature Review of ECC Implementations

This section reviews published implementations of ECC on both ASICs and FPGAs in chronological order. The FPGA-based implementations are considered in more detail because of the focus of this thesis. Details of implementations discussed in the following are collected to Table 6.3. The table includes the design which is the most comparable with the implementations of the publications of this thesis when several implementations are presented. Other reviews of ECC implementations are available in [26, 193].

The first hardware implementation of ECC was published by Agnew *et al.* in [2] in 1993. They presented a processor architecture with a bit-serial normal basis multiplier for $\mathbb{F}_{2^{155}}$. They provided performance estimates for both supersingular and non-supersingular curves. The first FPGA-based ECC architecture was presented by Rosner in [243] in 1998. Both of these implementations use parameters which are nowadays considered insecure, *i.e.*, composite fields are considered in [2, 243] and supersingular curves are used in [2]. Nevertheless, at least [2] is commonly considered one of the landmark works in the field of cryptographic hardware implementation. It was also considerably ahead of its time because implementations of ECC did not start to gain serious interest in academia before the turn of the millennium.

In 1999, Gao *et al.* presented ECC implementations on a low-cost Xilinx FPGA in [102]. However, the very small field sizes ($m \leq 53$) used in [102] were insecure already in 1999. Orlando and Paar [219] also presented performance estimates of ECC with their super-serial multiplier at the same conference, IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM) 1999.

In 2000, Leung *et al.* presented a microcoded processor architecture for FPGAs in [168]. The paper also presented an automated design generator which facilitated generation of several designs with different design parameters. The microcoded architecture provided easily modifiable point operations. This work was later continued by Leong and Leung in [167] which provided further studies on the architecture as well as the effects of coordinate selection, field size, and parallelism. In [106, 107], Goodman and Chandrakasan presented an ASIC implementation targeting to low-power applications. The architecture was reconfigurable in the sense that it supported several irreducible polynomials and implemented support for both $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. The architecture also performed SHA-1 hashing and RSA and, hence, supported practically all of the IEEE P1363 standard [132]. The papers concluded that ECC is both faster and more energy-efficient than RSA. Energy-efficiency comparisons to FPGA and software based implementations were also provided and the ASIC implementation was claimed to be 30-180 times more energy-efficient than FPGA and over three orders of magnitude better than software. Okada *et al.* presented an ECC processor based on a novel field multiplier architecture in [214]. The multiplier can be seen as a generalization of the super-serial multiplier presented in [219] and it supports arbitrary irreducible polynomials. This was also the first paper considering FPGA-based implementation of Koblitz curves. Koblitz curves were found to be nearly twice as fast as general curves. The architecture did not, however, include circuitry for τ-adic conversions. Orlando and Paar presented an FPGA-based processor optimized for a fixed irreducible polynomial in [220]. Irreducible polynomials could be changed by reprogramming the

FPGA. They provided comparisons between Montgomery point multiplication and binary method and concluded that Montgomery point multiplication is superior. Notice that [106, 214, 220] were all published in CHES 2000.

In 2001, Orlando and Paar [221] published an FPGA-based ECC processor for $\mathbb{F}_p$ using Montgomery multiplier with precomputed LUTs stored in embedded memory. Ernst *et al.* [94] discussed implementation of ECC by using an automated VHDL generator. Again, the generator provided multiple designs with various design parameters in a short time.

In 2002, Kerins *et al.* [144] presented an FPGA-based processor optimized for $a(x)b(x)/c(x) \mod p(x)$ in $\mathbb{F}_{2^m}$. The processor was designed to support point multiplication in $\mathcal{A}$, where the above computation is useful. Comprehensive comparisons of various aspects of ECC implementations were given by Bednara *et al.* in [27, 28]. Polynomial and normal bases were compared and polynomial basis was found to give better results. Coordinate systems and point multiplication algorithms were compared and Montgomery point multiplication was shown to provide the best results. Choosing the number of multipliers in the processor was also briefly discussed. Ernst *et al.* [93] presented a low-cost implementation for computing kP on $E(\mathbb{F}_{2^{133}})$ by using an Atmel FPGA board. In [114], Gura *et al.* presented an ECC implementation integrated into OpenSSL. Computations were optimized for $\mathbb{F}_{2^m}$ with certain fixed polynomials but support for arbitrary polynomials was also included. The fixed polynomials were shown to provide ten times faster performance. The architecture was improved by introducing partial reduction for arbitrary polynomials in [115] which reduced the gap to six times. Even further improvements were shown to be possible by Eberle *et al.* in [87] where a new multiplier was introduced reducing the difference to only two times. This is very impressive considering the complexity of polynomial reduction with an arbitrary irreducible polynomial. Schroeppel *et al.* [259] published a low-power ASIC implementation of ElGamal signatures on elliptic curves. The implementation utilized point halving, composite field $\mathbb{F}_{2^{178}}$ which was claimed to be secure, and included logic for performing all operations required in signature generation and verification, *i.e.*, also SHA-1 was included. Potgieter and van Dyk [227] presented a scalable and flexible processor architecture for $\mathbb{F}_{2^m}$. Their architecture was based on a multiplier combining the classical and Montgomery multipliers.

In 2003, Satoh and Takano [255] presented a processor architecture implementing support for both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ with different irreducible polynomials and primes. It was concluded that $\mathbb{F}_{2^m}$ is 3-6 times faster than $\mathbb{F}_p$ and support for $\mathbb{F}_{2^m}$ was achieved with no additional cost compared to $\mathbb{F}_p$. Nguyen *et al.* [212] studied implementation of ECC on a reconfigurable computer. Their study concentrated on partitioning between software and hardware (FPGA). They concluded that the entire point multiplication should be implemented in hardware, but it suffices that only the finite field arithmetic is implemented in VHDL. Örs *et al.* [222] described an ECC processor over $\mathbb{F}_p$. They used a multiplier based on systolic array architecture and Montgomery representation. The processor is capable of computing RSA as well.

In 2004, Lutz and Hasan [178] (based on Lutz's Master's thesis [177]) presented a processor architecture computing point multiplication on both general and Koblitz curves. The implementation was optimized for Koblitz curves so that all coordinates of Frobenius maps were computed in parallel with squarers attached to registers. The implementation did not include a τ-adic converter.

Table 6.3: Published ECC implementations

Reference	Year	Curve	Device	Area requirements[1]	Features[2]	t (μs)	T (ops)
Agnew [2]	1993	$E(\mathbb{F}_{2^{155}})$, NB	Gate array	∼11,000 gt.		——	——
Ansari [12]	2006	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-II 2000	8,300 LUT, 1,100 FF, 7 BRAM		41	24,000
Ansari [13]	2007	$E(\mathbb{F}_{2^{163}})$, PB	0.18 μm CMOS	36000 gt., 1KB RAM		62	19,000
Bajracharya [20]	2004	$E(\mathbb{F}_{2^{233}})$, NB	Virtex-II 6000	59% of sl.		201	5,000
Batina [24]	2005	$E(\mathbb{F}_{2^{179}})$, PB	Virtex 800	11,811 sl.	C	557	1,800
Batina [25]	2006	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-II Pro	8769 sl.	H	667	1,500
Bednara [28]	2002	$E(\mathbb{F}_{2^{191}})$, PB	Virtex 1000-4	——	O	2270	440
Benaissa [29]	2006	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000	6,010 LUT, 504 FF, 12 BRAM	F	804	1,200
Chelton [57]	2008	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-4 L200	16209 sl.	O	19.55	51,120
Chen [58]	2007	$E(\mathbb{F}_p)$, 256-bit	0.13 μm CMOS	122000 gt.		1010	990
Cheung [59]	2005	$E(\mathbb{F}_{2^{162}})$, NB	Virtex-II 6000	——	O	100	10,000
Daly [72]	2004	$E(\mathbb{F}_p)$, 160-bit	Virtex-E 2000-6	3434 sl.	O	——	——
Daneshbeh [73]	2004	——	0.18 μm CMOS	35,000 gt.	F	——	——
Eberle [87]	2003	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000-7	20,068 LUT, 6321 FF	F	302.29	3,308
Ernst [94]	2001	$E(\mathbb{F}_{2^{155}})$, NB	XC4085XLA	63% of CLB		1290	775
Ernst [93]	2002	$E(\mathbb{F}_{2^{113}})$, PB	AT94K40	96% of resources		1400	700
Gao [102]	1999	$E(\mathbb{F}_{2^{53}})$, NB	XC4044XL	1626 CLB	O	370	2,670
Goodman [107]	2001	$E(\mathbb{F}_{2^{176}})$, PB	0.25 μm CMOS	2.9×2.9 mm^2	FI	6950	140
Gura [114]	2002	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000-7	20,068 LUT, 6321 FF	(F)I	143.81	6,987
Gura [114]	2002	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000-7	20,068 LUT, 6321 FF	FI	1554	644
Gura [115]	2002	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000-7	——	F	930	1,075
Kerins [144]	2002	$E(\mathbb{F}_{2^{151}})$, PB	Virtex-E 2000-6	4,048 sl.	F	5100	200
Leong [167]	2002	$E(\mathbb{F}_{2^{113}})$, NB	Virtex 1000-6	1,410 sl.	O	4300	230
Leung [168]	2000	$E(\mathbb{F}_{2^{113}})$, NB	Virtex 300-4	1,290 sl.	O	3700	270
Liu [173]	2006	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-II 2000	910 CLB and 1 BRAM		1900	530
Lutz [177, 178]	2004	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000	10,017 LUT, 1,930 FF		233	4,300
Lutz [177, 178]	2004	$E_1(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000	10,017 LUT, 1,930 FF		75	13,300
McIvor [184]	2006	$E(\mathbb{F}_p)$, $< 2^{256}$	Virtex-II Pro 125-7	15,755 sl., 256 mult.		3860	260
Mentens [191]	2004	$E(\mathbb{F}_{2^{160}})$, PB	Virtex 800-4	150,678 equiv. gt.	O	3801	260
Morales-S. [199]	2006	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-II 4000	7342 sl.	O	1020	980

[1] sl. = slice, gt. = gate, BRAM = BlockRAM, FF = Flip-Flop, mult. = embedded multiplier, M4K and M512 = Stratix II embedded memory blocks

[2] F = Flexible, O = Other designs also included in the paper, C = Countermeasures against side-channel attacks, H = HECC, B = Both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$, M = Multiple point multiplications, I = Includes other operations, *e.g.*, SHA-1, T = τ-adic converter

Continued on next page...

Reference	Year	Curve	Device	Area requirements[1]	Features[2]	t (μs)	T (ops)
Nguyen [212]	2003	$E(\mathbb{F}_{2^{233}})$, PB	Virtex-II 6000	39% of 33,792 sl.	O	2979	336
Okada [214]	2000	$E(\mathbb{F}_{2^{163}})$, PB	Flex 10K 250-2	——	F	80300	12
Okada [214]	2000	$E_1(\mathbb{F}_{2^{163}})$, PB	Flex 10K 250-2	——	F	45600	22
Orlando [220]	2000	$E(\mathbb{F}_{2^{167}})$, PB	Virtex-E 400-8	3002 LUT, 1769 FF, 10 BRAM	O	210	4,800
Orlando [221]	2001	$E(\mathbb{F}_p)$, P-192	Virtex-E 1000-8	11,416 LUT, 5,735 FF, 35 BRAM		3000	330
Öztürk [224]	2004	$E(\mathbb{F}_p)$, $(2^{167}+1)/3$	0.13 μm CMOS	34,390 gt.	O	3100	320
Örs [222]	2003	$E(\mathbb{F}_p)$, 160-bit	Virtex-E 1000-8	11,227 LUT, 6,959 FF		14414	69
Peter [225]	2007	$E(\mathbb{F}_{2^{163}})$, PB	0.25 μm CMOS	1.0 mm^2	(F)O	83	12,000
Potgieter [227]	2002	$E(\mathbb{F}_{2^{163}})$, PB	Spartan-II 200	——	F	3776	260
Rodríguez-H. [241]	2004	$E(\mathbb{F}_{2^{191}})$, PB	Virtex-E 2600	17630 sl.		63	16,000
Rosner [243]	1998	$E(\mathbb{F}_{(2^8)^{21}})$, PB	XC4062XL-1	1810 sl.	O	4470	220
Sakiyama [249]	2006	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-II Pro 30	8450 sl.	FOH	280	3,600
Sakiyama [251]	2007	$E(\mathbb{F}_p)$, 256-bit	Spartan-III 5000-5	27597 sl.	O	17700	56
Sakiyama [250]	2007	$E(\mathbb{F}_{2^{163}})$, PB	0.13 μm CMOS	115000 gt.	(F)OH	29	34,000
Saqib [252]	2004	$E(\mathbb{F}_{2^{191}})$, PB	Virtex-E 3200	18314 sl., 24 BRAM		56.44	17,700
Satoh [255]	2003	$E(\mathbb{F}_{2^{160}})$, PB	0.13 μm CMOS	106,659 gt.	FB	190	5,300
Satoh [255]	2003	$E(\mathbb{F}_p)$, 192-bit	0.13 μm CMOS	106,659 gt.	FB	1440	700
Schroeppel [259]	2002	$E(\mathbb{F}_{(2^{89})^2})$, PB	0.5 μm CMOS	191,000 gt.	I	4400	230
Shu [262]	2005	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-E 2000-7	25,763 LUT, 7,467 FF	O	48	21,000
Sozzani [270]	2005	$E(\mathbb{F}_{2^{163}})$, PB	0.13 μm CMOS	0.66 mm^2		27	36,805
Zhou [310]	2007	$E(\mathbb{F}_{2^{163}})$, PB	Stratix II S15C3	6589 ALUT, 5777 FF, memory	I	750	1,300
III	2004	$E(\mathbb{F}_{2^{163}})$, PB	Virtex-II 8000-5	18,076 sl.	O	106	9,400
IV	2007	$E(\mathbb{F}_{2^{163}})$, NB	Stratix II S180C3	11,800 ALM, several M4K	O	48.88	20,458
IV	2007	$E_1(\mathbb{F}_{2^{163}})$, NB	Stratix II S180C3	13,472 ALM, several M4K	OT	25.81	49,318
V	2007	$E_1(\mathbb{F}_{2^{163}})$, NB	Stratix II S180C3	67,467 ALM, 305 M4K, 240 M512	MT	114.2	166,000
VI	2007	$E_1(\mathbb{F}_{2^{163}})$, PB	Stratix II S180C3	26,148 ALM	O	4.91	203,670
VII	2008	$E_1(\mathbb{F}_{2^{163}})$, PB	Stratix II S180C3	16,930 ALM, 21 M4K	MT	16.36	161,290
X	2006	$E_1(\mathbb{F}_{2^{163}})$, NB	Virtex-II 2000-6	8,745 sl., 10 BRAM, 2 mult.	T	35.75	28,000
XI	2008	$E_1(\mathbb{F}_{2^{163}})$, NB	Stratix II S180C3	8799 ALM, 36 M4K, 20 M512, 4 DSP	T	35.04	28,500
XI	2008	$E_1(\mathbb{F}_{2^{163}})$, NB	Stratix II S180C3	28328 ALM, 66 M4K, 52 M512, 4 DSP	T	13.38	74,700

[1] sl. = slice, gt. = gate, BRAM = BlockRAM, FF = Flip-Flop, mult. = embedded multiplier, M4K and M512 = Stratix II embedded memory blocks

[2] F = Flexible, O = Other designs also included in the paper, C = Countermeasures against side-channel attacks, H = HECC, B = Both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$, M = Multiple point multiplications, I = Includes other operations, *e.g.*, SHA-1, T = τ-adic converter

Mentens *et al.* [191] presented an implementation for $\mathbb{F}_p$ using Montgomery multiplier. Rodríguez-Henríquez *et al.* [241] presented an implementation computing Montgomery point multiplication with Karatsuba multipliers over $\mathbb{F}_{2^{191}}$. The paper also studied parallelization and concluded that up to four multipliers can be used in speeding up Montgomery point multiplication. Slightly improved results were presented in [252]. Bajracharya *et al.* [20] discussed implementation on a reconfigurable computer similarly as in [212] with different design parameters and considerably improved results. Öztürk *et al.* [224] presented a low-power ASIC implementation for ECC over $\mathbb{F}_p$. The cost of modular operations was reduced by introducing modulus scaling techniques and a new inversion algorithm together with an efficient hardware implementation. Daneshbeh and Hasan presented an ASIC-based implementation utilizing combined multiplier and inverter in [73]. The implementation handled different irreducible polynomials and field sizes. Daly *et al.* [72] introduced an FPGA-based ALU supporting AB/C operation in $\mathbb{F}_p$ using Montgomery representation.

In 2005, Sozzani *et al.* [270] discussed ASIC implementation of ECC and presented an implementation capable of computing both Montgomery point multiplication and binary method at the same time. This was achieved by prioritizing multiplier for multiplication-intensive Montgomery point multiplication and inverter for inversion-intensive binary method in $\mathcal{A}$. ECC was also compared to RSA and it was concluded that ECC is both faster and more area efficient. Batina *et al.* [24] presented an implementation which takes side-channel attacks into account on all design levels. Cheung *et al.* [59] introduced a customizable architecture and provided several implementations which were generated with an automated design generator. They also independently showed how Montgomery point multiplication can be computed with four parallel multipliers similarly as in [241]. Also the problem of comparing designs on different FPGAs was addressed by studying the implementations on different Xilinx FPGAs. Shu *et al.* [262] presented high-speed implementations utilizing parallel multipliers specialized for certain operations.

In 2006, Batina *et al.* [25] compared ECC and HECC. They concluded that HECC is faster but requires more area. However, they used binary method in $\mathcal{P}$ for ECC which is not the most efficient method. Liu *et al.* [173] proposed a method for computing point multiplication with shared LUTs in multipliers suggested in [123]. Benaissa *et al.* [29] presented a new word-level multiplier for arbitrary irreducible polynomials and provided point multiplication timings with a processor using the multiplier. McIvor *et al.* [184] studied implementation of ECC over $\mathbb{F}_p$. Their processor included a new combined modular inverter and bit-parallel multiplier. Morales-Sandoval and Feregrino-Uribe evaluated different implementation techniques and parallelization alternatives in [199]. They targeted implementations especially to mobile applications. Sakiyama *et al.* [249] presented a processor for both ECC and HECC. They exploited parallelism in operations and provided comparisons between ECC and HECC and concluded that ECC is faster. Another study from the same authors [250], continuing the work of [249], described a parallelized ASIC implementation of ECC and HECC and also preferred ECC. Ansari and Hasan [12] introduced a carefully implemented processor where the multiplier is always kept busy with no idle cycles. The processor implemented Montgomery point multiplication with polynomial basis.

In 2007, Sakiyama *et al.* [251] presented a reconfigurable ALU supporting

RSA and ECC over 256-bit $\mathbb{F}_p$. They concluded that ECC outperforms RSA by about factor of two. Chen *et al.* [58] presented a systolic array based architecture for $\mathbb{F}_p$ arithmetic and used it for ECC. Ansari and Wu [13] presented an ECC implementation employing an efficient digit-serial multiplier in polynomial basis and utilized parallelism in point operations which resulted in a fast ECC implementation. Peter *et al.* [225] studied the effects of supporting several irreducible polynomials in ECC implementations. They concluded that supporting a few irreducible polynomials can be done without major decrease in speed and increase in area. However, supporting arbitrary irreducible polynomials is costly but, nonetheless, considerably more efficient than software. Zhou *et al.* [310] presented an implementation on a Nios II soft-core processor utilizing custom functions for multiplication and inversion. Hence, the implementation falls to the class (e) in the classification of Fig. 3.4. They concluded that considerable speedup is achieved with the approach compared to pure software on Nios II. In 2008, Chelton and Benaissa [57] presented an implementation which appears to be the fastest FPGA-based implementation using general curves. Their processor utilized a carefully pipelined multiplier and supported custom instructions: multiply-accumulate, multiply-square, and repeated-square. The processor achieved point multiplication times of 19.55 μs and 33.05 μs in a Virtex-4 and Virtex-E FPGAs, respectively.

As can be seen above, a large number of ECC implementations have been published. The implementations target to various applications with different requirements and introduce numerous different implementation techniques. Hence, the following discussion is presented in order to provide an overview of the implementations and to ensure fairer comparison in Sec. 6.5.5.

6.5.1 Typical Structure of ECC Processors

The most typical structure of an ECC implementation is depicted in Fig. 6.2. Such a structure can be identified from a large majority of publications discussed above. It can be found also from implementations presented in **III**–**V**, **X**, and **XI**. The structure includes a Field Arithmetic Processor (FAP) which supports arithmetic in $\mathbb{F}_q$. FAPs always include dedicated processing units at least for multiplication and addition and memory or registers for storing elements of $\mathbb{F}_q$. An FAP implements the lowest level of the hierarchy of Fig. 6.1 and higher levels are implemented in control logic with a Finite State Machine (FSM) and/or microcode. If both FSM and microcode are used, the FSM implements the highest level of the hierarchy while point operation level is implemented with the microcode.

Depending on $\mathbb{F}_q$, other operations besides multiplication and addition are also commonly supported. Squaring is typically supported for $\mathbb{F}_{2^m}$ whereas $\mathbb{F}_p$ implementations usually include an inverter. The reasons are obvious: Squaring is very cheap in $\mathbb{F}_{2^m}$, especially in normal basis or with a fixed irreducible polynomial, and inversion in $\mathbb{F}_p$ would be very time consuming without a dedicated inverter. If an inverter is not available, inversion is computed with Fermat's Little Theorem, (4.19). However, inverters are commonly included also for $\mathbb{F}_{2^m}$, especially, with polynomial basis. Other operations that may be supported include shifts, comparisons, and custom-instructions, such as multiply-and-accumulate.

Control logic for an FAP is usually, but not always (see *e.g.* [115]), included in implementations presented in the papers. FAPs which do not include control

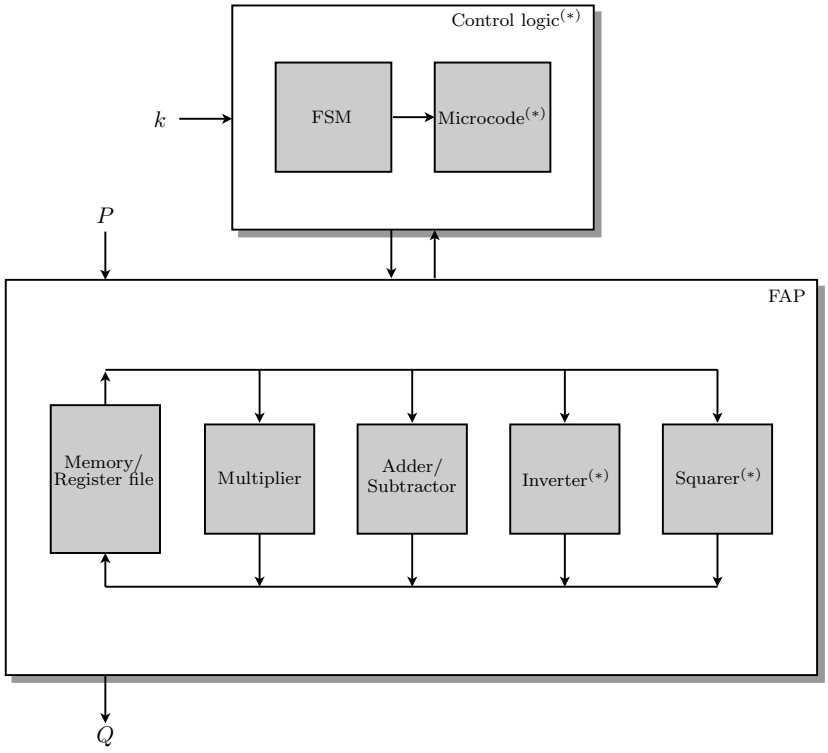

Figure 6.2: Typical structure of an ECC implementation. (*) not necessarily available in all FAPs.

logic can be used only in a close relationship with a controlling processor, *i.e.*, in the classification of Fig. 3.4, they can be used as coprocessors (c), reconfigurable function units (d), or embedded processing units (e). All implementations presented in the publications of this thesis include control logic and, therefore, they can be used in all roles given in Fig. 3.4, although their sizes probably prevent their use as a reconfigurable function unit.

6.5.2 Parallelism in ECC Processors

The possibility of using parallelism is the key factor that gives advantage for FPGA (and ASIC) implementations, but the use of parallelism in ECC is far from trivial. Because multiplications dominate in costs of point operations (if projective coordinates are in use), efficient computation of multiplications is emphasized in almost all published implementations. Use of parallelism is easy in finite field operations as discussed in Ch. 4. However, highly parallelized multipliers, *i.e.*, bit-parallel or digit-serial with large digit size, have poor area-latency efficiencies which degrade their feasibility in practice. Besides, when an area of a multiplier grows, routing in the place & route becomes much harder which degrades results even further. These aspects were studied in **IV**.

Another approach is to reduce the number of multiplications on the critical path by using multiple multipliers. This utilizes parallelism in point operations.

However, data dependencies prevent achieving full multiplier utilization with parallel multipliers which leads to poor area-latency efficiency. For example, point addition and point doubling of Montgomery point multiplication can be computed with critical paths of six, three, or two multiplications with one, two, or four multipliers, respectively [59, 241]. Hence, full multiplier utilization can be achieved only with one or two multipliers.

The situation is even worse when parallelism is utilized on the highest level of the hierarchy of Fig. 6.1. Point multiplication algorithms are recursive and, thus, contain only limited amounts of intrinsic parallelism. For example, point operations in Alg. 6.1 must be computed recursively because the result of a point addition is needed in point doublings, and vice versa. However, point addition and point doubling can be computed in parallel in Montgomery point multiplication, Alg. 6.2, [241, 262] and right-to-left binary method [212]. Beyond that, consecutive point operations must be computed recursively, which limits the amount of exploitable parallelism. However, this problem can be circumvented with Koblitz curves by exploiting the inexpensiveness of Frobenius maps as shown in **IV**. It is also possible to use more parallelism in point operations by interleaving successive point additions as shown in **VI**. Sozzani *et al.* [270] took another approach by computing two point multiplications simultaneously with different algorithms, one of which is multiplication-intensive and the other which is inversion-intensive.

6.5.3 Flexibility of ECC Processors

In many practical applications, implementations must handle elliptic curve operations with various different parameters, such as curve, field size, *etc.* In such applications, support for all parameters must be implemented. This support is referred to as flexibility. It is one of the most important features determining speed and area requirements of an implementation and, therefore, comparing designs with different levels of flexibility is difficult.

Many implementations having the structure of Fig. 6.2 provide inherent flexibility on the two highest levels of the hierarchy of Fig. 6.1, *e.g.*, point operations can be modified easily by uploading a new microcode. On the other hand, the same point multiplication algorithms and even point operation formulae can be used for a variation of curves which limits the need for such flexibility. However, if an FAP fails to support different finite fields, an implementation using that FAP is always limited to only few elliptic curves, and it cannot be considered flexible. Hence, support for various finite fields is the feature that has the most crucial effect on flexibility.

At simplest, limited field flexibility can be achieved by including specific circuitries for a small number of fields. For example, support for a few binary fields with polynomial basis can be achieved with specific reduction circuitries for a few irreducible polynomials. This approach provides adequate flexibility for many applications without major decrease in speed. However, the approach is not viable if a large number of fields must be supported.

Reduction with an arbitrary irreducible polynomial is iterative which makes it inherently slow. However, it can be computed fairly efficiently with methods such as the partial reduction [115]. On the other hand, fixed reduction circuitry can perform an entire reduction in one clock cycle with a high clock frequency with a few XORs if the polynomial is sparse (*e.g.*, trinomial or pentanomial).

Combination of the two above approaches, *i.e.*, combining the support for both arbitrary and specific irreducibles as shown in [87, 114, 115], is a viable solution in many practical applications. Reduction circuitries for specific irreducible polynomials provide fast computation for a few frequently used fields while arbitrary fields are supported by a slower generic reduction circuitry. However, even this approach restricts to binary fields with polynomial basis.

Normal bases do not suit well for flexible designs. Although it is easy to design flexible adders and squarers for normal bases, multipliers supporting arbitrary fields cannot be implemented efficiently, because they would need to support multiple F-functions; see Sec. 4.3.2. Therefore, an implementation should include circuitry for computing different F-functions on-the-fly and it should be able to update the F-function circuitry accordingly. At least to the author's knowledge, such multipliers have not been published, and it is hard to see how such multipliers could be devised without serious reductions in speed and increases in area.

Thus far, only flexibility of $\mathbb{F}_{2^m}$ has been discussed. Support for both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ can be added by utilizing techniques discussed in Sec. 4.5. It should be noted that $\mathbb{F}_p$ is slower than $\mathbb{F}_{2^m}$, and speedups compared to software are also considerably lower.

Flexibility is more crucial for ASICs than for FPGAs because FPGAs can provide flexibility through reprogramming; see the advantages provided by reprogrammability from Sec. 3.2. Of course, if parameters need to be changed frequently, reprogramming times start to dominate which makes this approach unpractical and, thus, flexible FAPs may be needed in FPGAs as well. Anyhow, design flexibility is undoubtedly more important in ASIC implementations where designs cannot be changed after manufacturing.

6.5.4 Optimizations for Specific Curves or Algorithms

The structure of Fig. 6.2 can be applied in elliptic curve point multiplication over all sets of parameters, *e.g.*, coordinate systems. However, structures optimized for certain operations have been presented and they provide considerable speed increases with the expense of reduced generality. A specific structure for Montgomery point multiplication was presented in [241]. Point addition and point doubling are computed in parallel with processing units build around one multiplier and several registers. Control logic is very simple: Only the bit k_i is used for selecting the inputs of point addition and point doubling processing units, see Alg. 6.2. However, the structure does not include logic for (6.12), *i.e.*, it outputs only X and Z coordinates of the result point, Q. Similar approach was presented in [262], but their implementation included a specific processing unit also for coordinate conversions.

The structure of Fig. 6.2 can be optimized for Koblitz curves by attaching squarers to the registers holding Q [178]. This enhances the performance on Koblitz curves without sacrificing generality. The architecture of **VI** can be seen as an adaptation of the ideas of [241, 262] to Koblitz curves, although the architecture also utilizes point operation interleaving allowing even further improvements in speed. The structure of **VII** builds upon **VI** and introduces specific units for precomputations, τ-adic conversions, for-loop computation, and coordinate conversion.

Complex instructions, *e.g.*, $A(B+D)+C$ in [250] or $a(x)b(x)/c(x) \mod p(x)$

in [144], can be seen as optimizations for specific algorithms. For example, $a(x)b(x)/c(x) \mod p(x)$ is useful for point operations in $\mathcal{A}$, but it has little use with projective coordinates.

6.5.5 Comparisons

Details of published ECC designs have been collected into Table 6.3. Fair comparison is extremely difficult (confer AES implementations) and all issues discussed in Sec. 5.3.3 cause difficulties also for ECC comparisons. However, comparing ECC designs is even more difficult because the variety of implemented algorithms is larger and this variety has great impacts on implementation results. For instance, curves and field sizes have major influences on both speed and area requirements, as can be seen, for example, from the tables in **VI** which list implementation results with different field sizes. Nevertheless, certain implementations that represent the state-of-the-art are given in the following together with estimates of how designs presented in **III**–**VII**, **X**, and **XI** compare with other published designs.

To the author's knowledge, the fastest design for general curves was presented by Chelton and Benaissa in [57] where point multiplication time 19.55 μs was achieved on NIST B-163 with a Virtex-4 FPGA. The fastest design for Koblitz curves is given in **VI** for NIST K-163 on which point multiplication takes 4.91 μs in a Stratix II FPGA excluding τ-adic conversions. When throughput is considered instead of computation time, the implementations presented in **V** and **VII** outperform other published designs, because they are the only ones optimized for high throughput, with the exception of [270], *i.e.*, throughput is simply the inverse of computation time for other designs. The most compact implementations given in Table 6.3 use parameters which are insecure, but compact implementations using secure parameters are given, for example, by Liu *et al.* in [173] and Orlando and Paar in [220]. Compact implementations for both general and Koblitz curves are listed also in **IV**. However, very compact FPGA-based implementations are still missing from the literature. Support for arbitrary irreducible polynomials was achieved efficiently by Eberle *et al.* in [87] with a technique called partial reduction. They also included fixed reduction circuitries for specific irreducible polynomial in order to accelerate their computation [87, 114, 115]. When support for both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ is needed, a good option is the implementation presented in [255].

Table 6.3 shows that the designs of this thesis compare favorably with other designs from the literature. The implementations are faster than most other published implementations and, in fact, **VI** and **VII** utilizing Koblitz curves represent the fastest published ECC implementations. The fastest general curve implementation, which was presented in [57], outperforms the implementations presented in **IV**, but it used polynomial basis whereas **IV** uses a normal basis. Table 6.3 also shows that Koblitz curves give considerable enhancements in speed compared to general curves. They are more than twice as fast as general curves even including the time of τ-adic conversions. Koblitz curves are therefore highly feasible alternatives in applications requiring very fast computation.

The area requirements of implementations of this thesis are quite large. They prevent using the implementations in constrained applications, but the area requirements are in line with other published implementations. Besides, the values given in Table 6.3 reflect only the area requirements of the fastest implementa-

tions and smaller implementations were also presented in the publications. All architectures presented in the publications are also scalable in the sense that speed can be traded off for smaller area. In the publications, the target has been in high speed which has resulted in large area requirements but, for example, Tables VII and VIII of **IV** show that compact designs can be realized with the same architectures.

The use of fixed parameters, especially, fixed fields, give a large speed advantage compared to flexible designs and, hence, the implementations of this thesis are not comparable with flexible designs. It has been shown that fixed designs are at least about twice as fast as flexible designs [87]. However, because the implementations of this thesis use FPGAs, the disadvantages of fixed designs are reduced as flexibility can be achieved in many cases by reprogramming the FPGA.

Chapter 7

Results

THIS CHAPTER summarizes the main research results of this thesis and identifies their significance to the research field of cryptographic implementation. Details, such as exact computation times, *etc.*, are not considered in this chapter but they are available in the appended publications. The contributions of this thesis are twofold as they relate to either AES or ECC, and they are discussed separately in Secs. 7.1 and 7.2, respectively.

7.1 AES-related Contributions

AES was discussed in **I**, which surveyed and compared AES implementations, and **II**, which presented a fast AES implementation for FPGAs. The following two sections discuss their results.

7.1.1 AES Surveys

The survey provided in **I** gave an overview of secret-key cryptographic implementations using FPGAs, as well as hash algorithms. The emphasis was on AES and the survey was the first review concentrating to high-speed implementations; to the author's knowledge, FPGA implementations of AES had been reviewed only in [293] where the emphasis was on security questions of FPGAs as implementation platforms. The implementations were compared both in terms of throughput per slice and throughput per area which consisted of the slice and BlockRAM utilization. A fairer comparison was provided than in other comparisons which had only considered slice count. However, even the new metric cannot provide an unconditionally fair comparison as discussed in Sec. 5.3.3. Nonetheless, the survey provided valuable information of the state of FPGA-based AES implementations at the time of publication.

The survey presented in Ch. 5 should be considered as a contribution of this thesis as well because, to the author's knowledge, it is the most comprehensive and up-to-date review of published ASIC and FPGA implementations available in the literature at the time of writing this thesis. Thus, it complements other reviews previously presented in the literature, such as **I** and [118, 120, 293].

7.1.2 Fast AES Implementation

The main contribution of the implementation of **II** was its speed. In retrospective, the approach taken in **II** is not the most efficient because it computes the entire encryption in $\mathbb{F}_{(2^4)^2}$. The design could be improved also in many other aspects; for instance, the pipeline should be balanced better and the isomorphisms should be optimized. Nonetheless, the implementation of **II** was the fastest published implementation of AES-128 encryption at the time of publication and presented the first published FPGA implementation utilizing composite fields. Hence, despite it weaknesses from the current point-of-view, it represented the state-of-the-art in achieved throughput at the time of publication, and it is still cited many recent publications.

7.2 ECC-related Contributions

The main contributions of this thesis relate to ECC, and ECC was the subject in **III**–**XI**. The focus was on high-speed implementation using parallelism. Operations of ECC are recursive in nature which restricts the use of parallelism. In this thesis, methods enabling increased parallelism were studied and developed. They resulted in faster implementations than existing methods, as shown in the comparison of Sec. 6.5.5. The studies considered both general and Koblitz curves, the emphasis being in Koblitz curves.

The implementations presented in **IV**–**XI** use curves specified in [205], *i.e.*, the NIST curves, and **III** uses curves from [52] recommended by SECG. Despite this fact, the contributions of this thesis are curve independent in the sense that they can be easily adapted to curves defined in different standards, or even to non-standardized curves. Of course, the methods developed specifically for Koblitz curves set restrictions for curve selection but even they are not restricted to any particular Koblitz curves.

7.2.1 ECC Implementations for General Curves

General curves were studied in **III** and **IV**. A fast and scalable architecture suitable for an automated VHDL generator [137] was presented in **III**. The architecture was optimized for the 4-to-1-bit LUT structure of the target FPGA. It was shown to result in very fast point multiplication times, including some of the fastest results available in the literature at the time of publication. In retrospective, the weakest link in the implementation was the use of a large control FSM which resulted in a large circuitry.

One of the key publications of this thesis, **IV**, also considered general curves. The main contribution of **IV** concerning general curves was that it illuminated the effects of parallelization in ECC implementations which was not adequately addressed in the existing literature. A generic architecture was presented and tools were provided for analyzing parallelization on different levels of the hierarchy of Fig. 6.1. The paper concluded that large field multipliers are inefficient and better results are obtained with several smaller multipliers in parallel.

7.2.2 Utilizing Parallelism with Koblitz Curves

Koblitz curves were considered in **IV**–**XI**. The main contribution of **IV**, especially useful for Koblitz curves, was the method to parallelize point multiplication for several FAPs. The method resulted in implementations which were faster and required less area than traditional implementations utilizing either one large field multiplier or several smaller ones.

Throughput maximization with parallel FAPs was studied in **V**. It was concluded that allowing slightly longer computation times for single operations results in a considerably higher throughput because more parallel FAPs fit into an FPGA. This conclusion is, again, due to the fact that multipliers with large digit sizes are inefficient. Improvements to precomputation algorithms related to three-term multiple point multiplications were also developed and they were shown to provide major enhancements compared to a straightforward approach. To the author's knowledge, **V** was the first paper presenting ECC implementations which prioritize throughput over computation time of a single point multiplication. Arguably, the focus in high-speed ECC implementations will shift to high throughput in the future. The reasons are that the computation times achievable with current knowledge are adequate for most applications, but there is still a need for higher throughput. Furthermore, increasing throughput by shortening computation time increases area considerably, whereas parallel processing leads to high throughput with reasonable area, as shown in **V**.

The parallelization methods presented in **IV** and **V** utilized parallelism on the highest level of the hierarchy of Fig. 6.1. Data dependencies restrict efficient use of parallelism on the middle level of the hierarchy, but observations made in **VI** helped to circumvent this problem by interleaving consecutive point additions. Hence, the method utilized parallelism on both the highest and middle levels of the hierarchy. The proposed method was shown to be highly feasible and scalable in practice. It resulted in the shortest point multiplication times available in the literature today. The same idea was used in **VII**, which introduced also further optimizations by utilizing similar structures of window methods and multiple point multiplications. An FPGA implementation with both short computation time and high throughput was achieved by designing optimized processing units for different parts of the algorithms.

These results prove that highly specialized implementations, which use large amounts of parallelism and features of specific curves, can provide very large speedups compared to more general implementations. Such specializations can be done in FPGAs without major reductions in generality of the system because an FPGA can be reprogrammed if other parameters are needed. The results also demonstrate the efficiency of Koblitz curves in hardware implementations. Prior to this thesis, only few publications had discussed implementations using Koblitz curves, but this thesis shows that Koblitz curves give major speed improvements even with τ-adic conversions. Hence, the results of this thesis can improve the attractiveness of Koblitz curves in hardware implementations and, as a consequence, also their use in practical systems.

7.2.3 Efficient Converters for Koblitz Curves

Because point multiplication times have been reduced to only a few microseconds, the traditional approach to compute τ-adic conversions in host processors

as in [178, 214] is not a viable solution as it becomes the bottleneck. Hence, efficient hardware implementations of τ-adic conversions are necessary. Hardware implementations of τ-adic conversions were not presented in the literature prior to **VIII** and **IX** and, thus, they delivered undeniable novelty. Algorithm modifications and an efficient implementation for converting integers into the τNAF were presented in **VIII**. The implementation was later used in **IV**, **V**, and **VII**. The converter from **IX** performs conversions to the other direction, from τ-adic representation to binary integer. Such conversions are useful in applications where random τ-adic expansions are used but also their integer equivalents are needed. Such applications include, for example, ECDSA with Koblitz curves. They could be realized with the converter of **VIII** as well, but the conversions of **IX** provide performance improvements because they can be computed in parallel with point multiplications.

7.2.4 Adaptation of the DBNS to Koblitz Curves

An adaptation of the DBNS to τ-adic representations was proposed in **X** where k was represented as $\sum_{i,j} k_{i,j}\tau^i(\tau-1)^j$. The representation was considerably sparser than the τNAF and, as a result, achieved point multiplication times were shorter. The paper also presented a variation using three bases which has serious theoretical significance because it results in the first provably sublinear point multiplication algorithm for Koblitz curves. Different adaptations with the form $\sum_{i,j} k_{i,j}\tau^i 3^j$ were proposed almost at the same time with **X** in [15, 16], but their practical usefulness was not verified. However, the results of the FPGA implementation proved that the representation suggested in **X** was indeed practical. The work of **X** was extended to a journal paper, **XI**, including also a study of the inherent parallelism included in the DBNS. It was shown that exploiting this parallelism leads to a very fast FPGA implementation.

Chapter 8

Conclusions

THIS THESIS studied implementation of cryptographic algorithms. The subject has importance in many practical systems which use cryptographic algorithms for improving security. Cryptographic algorithms are omnipresent in modern communication in which information security, such as confidentiality of communication or reliable authentication, are absolute necessities. The thesis studied techniques enabling fast computation of cryptographic algorithms which cannot be achieved with software-based solutions. Hardware acceleration is commonly required in important applications, such as heavily-loaded network servers. Efficient implementation techniques are necessary in order to provide high-security cryptographic algorithms to such applications and, hence, the subject has been studied intensively in both academia and industry. The focus of the thesis was on a secret-key cryptographic algorithm, AES, which was studied in **I** and **II**, and public-key cryptography algorithms, ECC, which were the topic of **III**–**XI**. The most important contributions of the thesis consider ECC.

Because cryptographic algorithms are used in a variety of applications ranging from low cost environments, such as RFID tags, to high speed systems, such as optical networks, the requirements set for cryptographic implementations differ. The most important requirements in low cost environments are low resource utilization and power consumption whereas high speed systems require short computation times and high throughputs. These requirements are usually out of reach with general-purpose microprocessors which are too slow and power consuming. On the other hand, ASICs are often too expensive and they lack flexibility which is essential for many cryptosystems. All presented implementations of this thesis targeted to FPGAs. However, many of the ideas and architectures can be adapted to ASICs as well. The focus of implementations was on accelerating computation and, hence, the target applications are environments which require very high computation speed.

In the introduction, Ch. 1, the scope of this thesis was described as finding answers to certain questions. The first ones asked which algorithms are the most suitable for hardware and how should they be improved in order to increase suitability. As shown in this thesis, the answers to these questions depend much on the implementation platform and target application. For instance, in

AES, both memory-based and combinatorial approaches, discussed in Sec. 5.2.1 and 5.2.2, respectively, have merits. Their mutual superiority depends on how much memory is available and whether operations are pipelined, *etc.* On the ECC side, this thesis showed that Koblitz curves are highly suitable also for hardware implementations and they result in significant speedups compared to general curves. It was also asked what kind of implementations provide the best results. This thesis has answered this question in many ways. For instance, although it is obvious that parallelism can improve the speed of implementations, this thesis showed that it must be carefully studied where to focus parallelism in order to maximize efficiency.

The thesis presented improvements both to algorithms and techniques for implementing them. Improvements to algorithms included modifications that made them more suitable for hardware implementations and reorganizations that enabled increased amount of parallelism. Practical implementations were presented in all publications and they exploited the results of the algorithm modifications. Many of the implementations were the fastest ones available in the literature at the time of publication which verifies the efficiency of both implementations and techniques used in them.

Next, the contributions of this thesis are shortly revisited, and certain topics for future research are pointed out, first, for AES and, then, for ECC. Efficient implementation techniques of AES have been studied extensively in the literature and they have considerable importance in many practical systems because of the wide distribution of AES. AES-related contributions of this thesis were the surveys and comparisons of high-speed FPGA-based implementations (**I**, Ch. 5) and an efficient combinatorial implementation (**II**). It can be concluded, based on the discussion of Ch. 5, that hardware implementation of AES is a mature field of research and it is unlikely that any new techniques considerably improving implementations will emerge. This conclusion is supported by the facts that techniques to implement AES have been extensively studied already for several years and methods to implement AES in practically all applications exist. However, introduction of some new features in FPGAs can produce a small renaissance to the field because they can open new directions for implementations. Indications of such possibilities can be found from the very recent papers [42, 86] which utilized the new Virtex-5 architecture. However, as stated in [42], such improvements arise directly from the new architecture and do not necessarily include any theoretical novelty.

As mentioned, the most important contributions relate to ECC. They include techniques increasing speed of ECC computations, especially studies on parallelization of ECC operations, and issues related to Koblitz curves. Several methods and architectures were presented for Koblitz curves including new methods to utilize parallel processing (**IV**, **V**), a method to interleave point additions (**VI**, **VII**), architectures for conversions needed for Koblitz curves (**VIII**, **IX**), and an adaptation of the DBNS which was shown to be highly feasible with an FPGA implementation (**X**, **XI**). There exist a plethora of published ECC designs as well. Despite the amount of work that has been done, the field of ECC implementations has not been fully matured and certain open problems still exist. Although basic solutions for implementing ECC are available and well-known, means to include ECC in very demanding environments requiring, *e.g.*, very low cost or high throughput, are not fully known. Minimizing area requirements is indispensable in order to provide ECC, or any kind of high-

security public-key cryptography for that matter, to low-cost environments. It has started to gain interest in the community [23, 116] and will probably be an important research topic in the near future. Combining low area requirements with efficient side-channel countermeasures, which are essential because of the viability of side-channel attacks in low-cost environments, is a challenging task that requires more research. On the other hand, the focus in high-speed implementations will probably shift to high throughput in the future. The reasons are that, arguably, adequately short computation times are achievable with current knowledge, but there would still be need for higher throughput. Naturally, throughput can by improved by shortening computation times but such an approach easily leads to poor area efficiency, whereas parallel processing provides more area efficient solutions, as shown in **V**. To the author's knowledge, the first studies focusing on maximizing throughput were presented in **V** and **VII**. New algorithms and curves providing more efficient computations have been introduced even recently, *e.g.*, [31], which naturally increases also the need for hardware implementation related research. Another topic, which will probably attain interest in the community, is implementation of hash algorithms. When NIST decides the finalists of the new hash algorithm competition, their hardware implementation will likely be an active research topic.

To conclude, this thesis has increased knowledge of how to accelerate computation of mainstream cryptographic algorithms with hardware accelerators, implemented specifically in FPGAs. The most important results are those enabling faster computation of ECC by using increased parallelism because they show that high-security public-key cryptography can be used also in computationally demanding systems. The publications of this thesis include the fastest reported ECC implementations currently available in the literature. The contributions of the thesis can be directly adapted to practical systems. Consequently, they can—in their part—enable increasing use of high-security cryptographic algorithms in various practical systems and, thus, improve the quality of such systems.

Bibliography

[1] Agnew, G.B., Mullin, R.C., Onyszchuk, I.M., and Vanstone, S.A., "An implementation for a fast public-key cryptosystem," *Journal of Cryptology*, vol. 3, no. 2, Jan. 1991, pp. 63–79.

[2] Agnew, G.B., Mullin, R.C., and Vanstone, S.A., "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, Jun. 1993, pp. 804–813.

[3] Al-Daoud, E., Mahmod, R., Rushdan, M., and Kilicman, A., "A new addition formula for elliptic curves over $GF(2^n)$," *IEEE Transactions on Computers*, vol. 51, no. 8, Aug. 2002, pp. 972–975.

[4] Alam, M., Ray, S., Mukhopadhayay, D., Ghosh, S., RoyChowdhury, D., and Sengupta, I., "An area optimized reconfigurable encryptor for AES-Rijndael," in *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe, DATE 2007*, Nice, France, Apr. 16–20, 2007, pp. 1116–1121.

[5] Altera Corporation, "Nios II processor reference handbook," Manual, ver. 7.2.0, Oct. 2007, http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf (Oct. 9, 2008).

[6] ———, "Stratix II device handbook," Data sheet, May 2007, http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf (Oct. 9, 2008).

[7] ———, "Stratix III device handbook," Data sheet, Jul. 2008, http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf (Oct. 9, 2008).

[8] American National Standards Institute (ANSI), "Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)," *ANSI X9.62*, 1999.

[9] ———, "Public key cryptography for the financial services industry: Key agreement and key transport using elliptic curve cryptography," *ANSI X9.63*, 2001.

[10] Anderson, R., Bond, M., Clulow, J., and Skorobogatov, S., "Cryptographic processors—a survey," *Proceedings of the IEEE*, vol. 94, no. 2, Feb. 2006, pp. 357–369.

[11] Anderson, R.J., "Why cryptosystems fail," *Communications of the ACM*, vol. 37, no. 11, Nov. 1994, pp. 32–40.

[12] Ansari, B. and Anwar Hasan, M., "High performance architecture of elliptic curve scalar multiplication," *Techical Report CORR 2006-01*, University of Waterloo, Canada, 2006.

[13] Ansari, B. and Wu, H., "Efficient finite field processor for $GF(2^{163})$ and its VLSI implementation," in *Proceedings of the 4th International Conference on Information Technology: New Generations, ITNG 2007*, Las Vegas, Nevada, USA, Apr. 2–4, 2007, pp. 1021–1026.

[14] Ash, D.W., Blake, I.F., and Vanstone, S.A., "Low complexity normal bases," *Discrete Applied Mathematics*, vol. 25, no. 3, Nov. 1989, pp. 191–210.

[15] Avanzi, R., Dimitrov, V.S., Doche, C., and Sica, F., "Extending scalar multiplication using double bases," in *Proceedings of the 12th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology — ASIACRYPT 2006*, Shanghai, China, Dec. 3–7, 2006, Lecture Notes in Computer Science, vol. 4284, Springer, pp. 130–144.

[16] Avanzi, R. and Sica, F., "Scalar multiplication on Koblitz curves using double bases," in *Revised Selected Papers of the 1st International Conference on Cryptology in Vietnam, Progress in Cryptology — VIETCRYPT 2006*, Hanoi, Vietnam, Sep. 25–28, 2006, Lecture Notes in Computer Science, vol. 4341, Springer, pp. 131–146.

[17] Avanzi, R.M., Ciet, M., and Sica, F., "Faster scalar multiplication on Koblitz curves combining point halving with the Frobenius endomorphism," in *Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography, PKC 2004*, Singapore, Mar. 1–4, 2004, Lecture Notes in Computer Science, vol. 2947, Springer, pp. 28–40.

[18] Avanzi, R.M. and Cohen, H., "Compositeness and primality testing—factoring," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography* (H. Cohen and G. Frey, eds.), chap. 25, Chapman & Hall/CRC, 2006, pp. 591–614.

[19] Avanzi, R.M., Heuberger, C., and Prodinger, H., "Minimality of the Hamming weight of the τ-NAF for Koblitz curves and improved combination with point halving," in *Revised Selected Papers of the 12th International Workshop on Selected Areas on Cryptography, SAC 2005*, Kingston, Ontario, Canada, Aug. 11–12, 2005, Lecture Notes in Computer Science, vol. 3897, Springer, pp. 332–344.

[20] Bajracharya, S., Shu, C., Gaj, K., and El-Ghazawi, T., "Implementation of elliptic curve cryptosystems over $GF(2^n)$ in optimal normal basis on

a reconfigurable computer," in *Proceedings of the International Conference on Field Programmable Logic and Application, FPL 2004*, Antwerp, Belgium, Aug. 29–Sep. 1, 2004, Lecture Notes in Computer Science, vol. 3203, Springer, pp. 1001–1005.

[21] Barreto, P.S.L.M. and Rijmen, V., "The Whirlpool hashing function," Online document, May 24, 2003, http://planeta.terra.com.br/informatica/paulobarreto/whirlpool.zip (Oct. 9, 2008).

[22] Barrett, P., "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology — CRYPTO '86*, Santa Barbara, California, USA, Aug. 11–15, 1986, Lecture Notes in Computer Science, vol. 263, Springer, pp. 311–323.

[23] Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., and Verbauwhede, I., "Public-key cryptography for RFID-tags," in *Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications, PerCom 2007*, White Plains, New York, USA, Mar. 19–23, 2007, pp. 217–222.

[24] Batina, L., Mentens, N., Preneel, B., and Verbauwhede, I., "Side-channel aware design: Algorithms and architectures for elliptic curve cryptography over $GF(2^n)$," in *Proceedings of the 16th International Conference on Application-Specific Systems, Architecture and Processors, ASAP 2005*, Samos, Greece, Jul. 23–25, 2005, pp. 350–355.

[25] ———, "Flexible hardware architectures for curve-based cryptography," in *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006*, Island of Kos, Greece, May 21–24, 2006, pp. 4839–4842.

[26] Batina, L., Örs, S.B., Preneela, B., and Vandewalle, J., "Hardware architectures for public key cryptography," *Integration, the VLSI Journal*, vol. 34, no. 1-2, 2003, pp. 1–64.

[27] Bednara, M., Daldrup, M., Teich, J., von zur Gathen, J., and Shokrollahi, J., "Tradeoff analysis of FPGA based elliptic curve cryptography," in *Proceedings of the 2002 IEEE International Symposium on Circuits and Systems, ISCAS 2002*, Phoenix-Scottdale, Arizona, USA, May 26–29, 2002, vol. 5, pp. 797–800.

[28] Bednara, M., Daldrup, M., von zur Gathen, J., Shokrollahi, J., and Teich, J., "Reconfigurable implementation of elliptic curve crypto algorithms," in *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS 2002, Reconfigurable Architectures Workshop, RAW 2002*, Ft. Lauderdale, Florida, USA, Apr. 15–19, 2002, pp. 157–164.

[29] Benaissa, M. and Lim, W.M., "Design of flexible $GF(2^m)$ elliptic curve cryptography processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 6, Jun. 2006, pp. 659–662.

[30] Berlekamp, E.R., *Algebraic Coding Theory*, McGraw-Hill, 1968.

[31] Bernstein, D.J., Lange, T., and Farashahi, R.R., "Binary Edwards curves," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2008*, Washington, DC, USA, Aug. 10–13, 2008, Lecture Notes in Computer Science, vol. 5154, Springer, pp. 244–265.

[32] Blake, I., Seroussi, G., and Smart, N., *Elliptic Curves in Cryptography*, London Mathematical Society Lecture Notes Series, vol. 265, Cambridge University Press, 1999.

[33] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., and Vikkelsoe, C., "PRESENT: An ultra-lightweight block cipher," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2007*, Vienna, Austria, Sep. 10–13, 2007, Lecture Notes in Computer Science, vol. 4727, Springer, pp. 450–466.

[34] Bouldin, D., "Enhancing electronic systems with reconfigurable hardware," *IEEE Circuits and Devices Magazine*, vol. 22, no. 3, May–Jun. 2006, pp. 32–36.

[35] Brauer, A., "On addition chains," *Bulletin of the American Mathematical Society*, vol. 45, no. 10, Oct. 1939, pp. 736–739.

[36] Brickell, E.F., Gordon, D.M., McCurley, K.S., and Wilson, D.B., "Fast exponentiation with precomputation," in *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques, Advances in Cryptology — EUROCRYPT '92*, Balatonfüred, Hungary, May 24–28, 1992, Lecture Notes in Computer Science, vol. 658, Springer, pp. 200–207.

[37] Brokalakis, A., Kakarountas, A.P., and Goutis, C.E., "A high-throughput area efficient FPGA implementation of AES-128 encryption," in *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation, SiPS 2005*, Athens, Greece, Nov. 2–4, 2005, pp. 116–121.

[38] Brumley, B.B., "Efficient three-term simultaneous elliptic scalar multiplication with applications," in *Proceedings of the 11th Nordic Workshop on Secure IT Systems, NordSec 2006*, Linköping, Sweden, 2006, pp. 105–116.

[39] ———, "Left-to-right signed-bit τ-adic representations of n integers," in *Proceedings of the 8th International Conference on Information and Communications Security, ICICS 2006*, Raleigh, North Carolina, USA, Dec. 4–7, 2006, Lecture Notes in Computer Science, vol. 4307, Springer, pp. 469–478.

[40] ———, "Implementing cryptography for packet level authentication," in *Proceedings of the 2008 International Conference on Security and Management, SAM 2008*, Las Vegas, Nevada, USA, Jul. 14–17, 2008, CSREA Press, pp. 475–480.

[41] Brumley, B.B. and Järvinen, K.U., "Fast point decompression for standard elliptic curves," in *Proceedings of the 5th European PKI Workshop, EuroPKI 2008*, Trondheim, Norway, Jun. 16–17, 2008, Lecture Notes in Computer Science, vol. 5057, Springer, pp. 134–149.

[42] Bulens, P., Standaert, F.X., Quisquater, J.J., Pellegrin, P., and Rouvroy, G., "Implementation of the AES-128 on Virtex-5 FPGAs," in *Proceedings of the 1st International Conference on Cryptology in Africa, Progress in Cryptology — AFRICACRYPT 2008*, Casablanca, Morocco, Jun. 11–14, 2008, Lecture Notes in Computer Science, vol. 5023, Springer, pp. 16–26.

[43] Bundesamt für Sicherheit in der Informationstechnik (BSI), "Elliptic curve cryptography based on ISO 15946," *Technical Guideline TR-03111, ver. 1.00*, Feb. 14, 2007, http://www.bsi.de/literat/tr/tr03111/BSI-TR-03111.pdf (Oct. 9, 2008).

[44] ———, "Advanced security mechanisms for machine readable travel documents — extended access control (EAC)," *Technical Guideline TR-03110, ver. 1.11*, Feb. 21, 2008, http://www.bsi.de/fachthem/epass/TR-03110_v111.pdf (Oct. 9, 2008).

[45] Burr, W.E., "Selecting the advanced encryption standard," *IEEE Security and Privacy*, vol. 1, no. 2, Mar.–Apr. 2003, pp. 43–52.

[46] Candolin, C., *Securing Military Decision Making in a Network-Centric Environment*, Ph.D. thesis, Helsinki University of Technology, 2005.

[47] Candolin, C., Lundberg, J., and Kari, H., "Packet level authentication in military networks," in *Proceedings of the 6th Australian Information Warfare & IT Security Conference*, Geelong, Australia, Nov. 2005.

[48] Canright, D., "A very compact S-box for AES," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2005*, Edinburgh, Scotland, UK, Aug. 29–Sep. 1, 2005, Lecture Notes in Computer Science, vol. 3659, Springer, pp. 441–456.

[49] Carlier, V., Chabanne, H., Dottax, E., and Pelletier, H., "Electromagnetic side channels of an FPGA implementation of AES," Cryptology ePrint Archive, Report 2004/145, 2004, http://eprint.iacr.org/ (Oct. 9, 2008).

[50] Certicom, Corp., "The Certicom ECC challenge," Web page, http://www.certicom.ca/index.php/the-certicom-ecc-challenge (Oct. 9, 2008).

[51] Certicom Research, "SEC 1: Elliptic curve cryptography," *Standards for Efficient Cryptography*, Sep. 20, 2000.

[52] ———, "SEC 2: Recommended elliptic curve domain parameters," *Standards for Efficient Cryptography*, Sep. 20, 2000.

[53] ———, "Certicom patent letter," Online document, Feb. 10, 2005, http://www.secg.org/download/aid-398/certicom_patent_letter_SECG.pdf (Oct. 9, 2008).

[54] Chang Shantz, S., "From Euclid's GCD to Montgomery multiplication to the great divide," *Techical Report SMLI TR-2001-95*, Sun Microsystems, Inc., Jun. 2001.

[55] Charot, F., Yahya, E., and Wagner, C., "Efficient modular-pipelined AES implementation in counter mode on Altera FPGA," in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications, FPL 2003*, Lisbon, Portugal, Sep. 1–3, 2003, Lecture Notes in Computer Science, vol. 2778, Springer, pp. 282–291.

[56] Chaves, R., Kuzmanov, G., Vassiliadis, S., and Sousa, L., "Reconfigurable memory based AES co-processor," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, the 13th Reconfigurable Architectures Workshop, RAW 2006*, Rhodes Island, Greece, Apr. 25–26, 2006.

[57] Chelton, W.N. and Benaissa, M., "Fast elliptic curve cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, Feb. 2008, pp. 198–205.

[58] Chen, G., Bai, G., and Chen, H., "A high-performance elliptic curve cryptographic processor for general curves over $GF(p)$ based on a systolic arithmetic unit," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, vol. 54, no. 5, May 2007, pp. 412–416.

[59] Cheung, R.C.C., Telle, N.J., Luk, W., and Cheung, P.Y.K., "Customizable elliptic curve cryptosystem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, Sep. 2005, pp. 1048–1059.

[60] Chevallier-Mames, B., Ciet, M., and Joye, M., "Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity," *IEEE Transactions on Computers*, vol. 53, no. 6, Jun. 2006, pp. 760–768.

[61] Chodowiec, P. and Gaj, K., "Very compact FPGA implementation of the AES algorithm," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2003*, Cologne, Germany, Sep. 8–10, 2003, Lecture Notes in Computer Science, vol. 2779, Springer, pp. 319–333.

[62] Chodowiec, P., Khuon, P., and Gaj, K., "Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining," in *Proceedings of the 2001 ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays*, Monterey, California, USA, Feb. 11–13, 2001, pp. 94–102.

[63] Chou, W., "Inside SSL: Accelerating secure transactions," *IEEE IT Professional*, vol. 4, no. 5, Sep.–Oct. 2002, pp. 37–41.

[64] Chung, J. and Hasan, M.A., "Low-weight polynomial form integers for efficient modular multiplication," *IEEE Transactions on Computers*, vol. 56, no. 1, Jan. 2007, pp. 44–57.

[65] Ciet, M., Lange, T., Sica, F., and Quisquater, J.J., "Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Advances in Cryptology — EUROCRYPT 2003*, Warsaw, Poland, May 4–8, 2003, Lecture Notes in Computer Science, vol. 2656, Springer, pp. 388–400.

[66] Ciet, M. and Sica, F., "An analysis of double base number systems and a sublinear scalar multiplication algorithm," in *Proceedings of the 1st International Conference on Cryptology in Malaysia, Progress in Cryptology — Mycrypt 2005*, Kuala Lumpur, Malaysia, Sep. 28–30, 2005, Lecture Notes in Computer Science, vol. 3715, Springer, pp. 171–182.

[67] Cilardo, A., Coppolino, L., Mazzocca, N., and Romano, L., "Elliptic curve cryptography engineering," *Proceedings of the IEEE*, vol. 94, no. 2, Feb. 2006, pp. 395–406.

[68] Cohen, H. and Frey, G. (eds.), *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman & Hall/CRC, 2006.

[69] Compton, K. and Hauck, S., "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, Jun. 2002, pp. 171–210.

[70] Crandall, R.E., "Method and apparatus for public key exchange in a cryptographic system," United States Patent 5,159,632, Oct. 27, 1992.

[71] Daemen, J. and Rijmen, V., *The Design of Rijndael: AES — The Advanced Encryption Standard*, Springer, 2002.

[72] Daly, A., Marnane, W., Kerins, T., and Popovici, E., "An FPGA implementation of a $GF(p)$ ALU for encryption processors," *Microprocessors and Microsystems*, vol. 28, no. 5-6, Aug. 2004, pp. 253–260.

[73] Daneshbeh, A.K. and Hasan, M.A., "Area efficient high speed elliptic curve cryptoprocessor for random curves," in *Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2004*, Las Vegas, Nevada, USA, Apr. 5–7, 2004, vol. 2, pp. 588–592.

[74] Dhem, J.F., "Efficient modular reduction algorithm in $\mathbb{F}_q[x]$ and its application to "left to right" modular multiplication in $\mathbb{F}_2[x]$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2003*, Cologne, Germany, Sep. 8–10, 2003, Lecture Notes in Computer Science, vol. 2779, Springer, pp. 203–213.

[75] Dhem, J.F., Koeune, F., Leroux, P.A., Mestré, P., Quisquater, J.J., and Willems, J.L., "A practical implementation of the timing attack," in *Proceedings of the 3rd International Conference on Smart Card Research and Applications, CARDIS '98*, Louvain-la-Neuve, Belgium, Sep. 14–16, 1998, Lecture Notes in Computer Science, vol. 1820, Springer, pp. 167–182.

[76] Diffie, W. and Hellman, M.E., "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, Nov. 1976, pp. 644–654.

[77] Dimitrov, V., Imbert, L., and Mishra, P.K., "Efficient and secure elliptic curve point multiplication using double-base chains," in *Proceedings of the 11th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology — ASIACRYPT 2005*, Chennai, India, Dec. 4–8, 2005, Lecture Notes in Computer Science, vol. 3788, Springer, p. 59–78.

[78] ———, "The double-base number system and its application to elliptic curve cryptography," *Mathematics of Computation*, vol. 77, no. 262, Apr. 2008, pp. 1075–1104.

[79] Dimitrov, V.S., Jullien, G.A., and Miller, W.C., "An algorithm for modular exponentiation," *Information Processing Letters*, vol. 66, no. 3, May 1998, pp. 155–159.

[80] ———, "Theory and applications of the double-base number system," *IEEE Transactions on Computers*, vol. 48, no. 10, Oct. 1999, pp. 1098–1106.

[81] Dobbertin, H., Bosselaers, A., and Preneel, B., "RIPEMD-160: A strengthened version of RIPEMD," in *Proceedings of the 3rd International Workshop on Fast Software Encryption, FSE 1996*, Cambridge, UK, Feb. 21–23, 1996, Lecture Notes in Computer Science, vol. 1039, Springer, pp. 71–82.

[82] Doche, C., "Exponentiation," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography* (H. Cohen and G. Frey, eds.), chap. 9, Chapman & Hall/CRC, 2006, pp. 145–168.

[83] ———, "Finite field arithmetic," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography* (H. Cohen and G. Frey, eds.), chap. 11, Chapman & Hall/CRC, 2006, pp. 201–237.

[84] Doche, C. and Lange, T., "Arithmetic of elliptic curves," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography* (H. Cohen and G. Frey, eds.), chap. 13, Chapman & Hall/CRC, 2006, pp. 267–302.

[85] Doche, C. and Lubicz, D., "Algebraic background," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography* (H. Cohen and G. Frey, eds.), chap. 2, Chapman & Hall/CRC, 2006, pp. 19–37.

[86] Drimer, S., Güneysu, T., and Paar, C., "DSPs, BRAMs and a pinch of logic: New recipes for AES on FPGAs," in *Proceedings of the 16th Annual IEEE Symposium on Field-programmable Custom Computing Machines, FCCM 2008*, Stanford, California, USA, Apr. 14–15, 2008, to appear.

[87] Eberle, H., Gura, N., and Chang-Shantz, S., "A cryptographic processor for arbitrary elliptic curves over $GF(2^m)$," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors, ASAP 2003*, The Hague, The Netherlands, Jun. 24–26, 2003, pp. 444–454.

[88] Eberle, H., Gura, N., Chang Shantz, S., Gupta, V., Rarick, L., and Sundaram, S., "A public-key cryptographic processor for RSA and ECC," in *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2004*, Galveston, Texas, USA, Sep. 27–29, 2004, pp. 98–110.

[89] Eberle, H., Shantz, S., Gupta, V., Gura, N., Rarick, L., and Spracklen, L., "Accelerating next-generation public-key cryptosystems on general-purpose CPUs," *IEEE Micro*, vol. 25, no. 2, Mar.–Apr. 2005, pp. 52–59.

[90] Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., and Uhsadel, L., "A survey of lightweight-cryptography implementations," *IEEE Design and Test of Computers*, vol. 24, no. 6, Nov.–Dec. 2007, pp. 522–533.

[91] Elbirt, A.J., Yip, W., Chetwynd, B., and Paar, C., "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, Aug. 2001, pp. 545–557.

[92] ElGamal, T., "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, Jul. 1985, pp. 469–472.

[93] Ernst, M., Jung, M., Madlener, F., Huss, S., and Blümel, R., "A reconfigurable system on chip implementation for elliptic curve cryptography over $GF(2^n)$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 381–399.

[94] Ernst, M., Klupsch, S., Hauck, O., and Huss, S.A., "Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems," in *Proceedings of the 12th International Workshop on Rapid System Prototyping, RSP 2001*, Monterey, California, USA, Jun. 25–27, 2001, pp. 24–29.

[95] Feldhofer, M., Dominikus, S., and Wolkerstorfer, J., "Strong authentication for RFID systems using the AES algorithm," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2004*, Cambridge, Massachusetts, USA, Aug. 10–13, 2004, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 357–370.

[96] Feldhofer, M., Wolkerstorfer, J., and Rijmen, V., "AES implementation on a grain of sand," *IEE Proceedings: Information Security*, vol. 152, no. 1, Oct. 2005, pp. 13–20.

[97] Fischer, V. and Drutarovský, M., "Two methods of Rijndael implementation in reconfigurable hardware," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 77–92.

[98] Fischer, V., Drutarovský, M., Chodowiec, P., and Gramain, F., "InvMixColumn decomposition and multilevel resource sharing in AES implementations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 8, Aug. 2005, pp. 989–992.

[99] Frey, G. and Lange, T., "Algebraic realizations of DL systems," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography* (H. Cohen and G. Frey, eds.), chap. 23, Chapman & Hall/CRC, 2006, pp. 547–572.

[100] Gaj, K. and Chodowiec, P., "Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays," in *Proceedings of the Cryptographers' Track at*

the RSA Conference 2001, Topics in Cryptology — CT-RSA 2001, San Francisco, California, USA, Apr. 8–12, 2001, Lecture Notes in Computer Science, vol. 2020, Springer, pp. 84–99.

[101] Gallant, R.P., Lambert, R.J., and Vanstone, S.A., "Faster point multiplication on elliptic curves with efficient endomorphisms," in *Proceedings of the 21st Annual International Cryptology Conference, Advances in Cryptology — CRYPTO 2001*, Santa Barbara, California, USA, Aug. 19–23, 2001, Lecture Notes in Computer Science, vol. 2139, Springer, pp. 190–200.

[102] Gao, L., Shrivastava, S., Lee, H., and Sobelman, G.E., "A compact fast variable key size elliptic curve cryptosystem coprocessor," in *Proceedings of the 7th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 1999*, Napa Valley, California, USA, Apr. 21–23, 1999, pp. 304–305.

[103] Good, T. and Benaissa, M., "AES on FPGA from the fastest to the smallest," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2005*, Edinburgh, Scotland, UK, Aug. 29–Sep. 1, 2005, Lecture Notes in Computer Science, vol. 3659, Springer, pp. 427–440.

[104] ———, "Very small FPGA application-specific instruction processor for AES," *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, vol. 53, no. 7, Jul. 2006, pp. 1477–1486.

[105] ———, "Pipelined AES on FPGA with support for feedback modes (in a multi-channel environment)," *IET Information Security*, vol. 1, no. 1, Mar. 2007, pp. 1–10.

[106] Goodman, J. and Chandrakasan, A., "An energy efficient reconfigurable public-key cryptography processor architecture," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Worcester, Massachusetts, USA, Aug. 17–18, 2000, Lecture Notes in Computer Science, vol. 1965, Springer, pp. 175–190.

[107] ———, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, Nov. 2001, pp. 1808–1820.

[108] Gordon, D.M., "A survey of fast exponentiation methods," *Journal of Algorithms*, vol. 27, no. 1, Apr. 1998, pp. 129–146.

[109] Grabbe, C., Bednara, M., Teich, J., von zur Gathen, J., and Shokrollahi, J., "FPGA designs of parallel high performance $GF(2^{233})$ multipliers," in *Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, ISCAS 2003*, Bangkok, Thailand, May 25–28, 2003, vol. 2, pp. 268–271.

[110] Grabner, P.J., Heuberger, C., and Prodinger, H., "Distribution results for low-weight binary representations for pairs of integers," *Theoretical Computer Science*, vol. 319, no. 1-3, Jun. 2004, pp. 307–331.

[111] Großschädl, J., "A bit-serial unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 202–219.

[112] Großschädl, J. and Savaş, E., "Instruction set extensions for fast arithmetic in finite fields $GF(p)$ and $GF(2^m)$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2004*, Cambridge, Massachusetts, USA, Aug. 10–13, 2004, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 133–147.

[113] Guajardo, J. and Paar, C., "Itoh-Tsujii inversion in standard basis and its application in cryptography and codes," *Designs, Codes and Cryptography*, vol. 25, no. 2, Feb. 2002, pp. 207–216.

[114] Gura, N., Chang Shantz, S., Eberle, H., Gupta, S., Gupta, V., Finchelstein, D., Goupy, E., and Stebila, D., "An end-to-end systems approach to elliptic curve cryptography," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 349–365.

[115] Gura, N., Eberle, H., and Chang Shantz, S., "Generic implementations of elliptic curve cryptography using partial reduction," in *Proceedings of the 9th ACM conference on Computer and Communications Security, CCS 2002*, Washington, DC, USA, Nov. 18–22, 2002, vol. 1, pp. 108–116.

[116] Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S., "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2004*, Cambridge, Massachusetts, USA, Aug. 10–13, 2004, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 119–132.

[117] Halbutogullari, A. and Koç, Ç.K., "Mastrovito multipliers for general irreducible polynomials," *IEEE Transactions on Computers*, vol. 49, no. 5, May 2000, pp. 503–518.

[118] Hämäläinen, P., *Cryptographic Security Design and Hardware Architectures for Wireless Local Area Networks*, Ph.D. thesis, Tampere University of Technology, 2006.

[119] Hämäläinen, P., Alho, T., Hännikäinen, M., and Hämäläinen, T.D., "Design and implementation of low-area and low-power AES encryption hardware core," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, DSD 2006*, Dubrovnik, Croatia, Aug. 30–Sep. 1, 2006, pp. 577–583.

[120] Hämäläinen, P., Hännikäinen, M., and Hämäläinen, T.D., "Review of hardware architectures for advanced encryption standard implementations considering wireless sensor networks," in *Proceedings of the 7th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2007*, Samos, Greece, Jul. 16–19, 2007, Lecture Notes in Computer Science, vol. 4599, Springer, pp. 443–453.

[121] Hankerson, D., Hernandez, J.L., and Menezes, A., "Software implementation of elliptic curve cryptography over binary fields," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Worcester, Massachusetts, USA, Aug. 17–18, 2000, Lecture Notes in Computer Science, vol. 1965, Springer, pp. 1–24.

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

[123] Hasan, M.A., "Look-up table-based large finite field multiplication in memory constrained cryptosystems," *IEEE Transactions on Computers*, vol. 49, no. 7, Jul. 2000, pp. 749–758.

[124] Higuchi, A. and Takagi, N., "A fast addition algorithm for elliptic curve arithmetic in $GF(2^n)$ using projective coordinates," *Information Processing Letters*, vol. 76, no. 3, Dec. 15 2000, pp. 101–103.

[125] Hodjat, A. and Verbauwhede, I., "A 21.54 Gbits/s fully pipelined AES processor on FPGA," in *Proceedings of 2004 IEEE Symposium on Field-programmable Custom Computing Machines, FCCM 2004*, Napa, California, USA, Apr. 20–23, 2004, pp. 308–309.

[126] ———, "Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors," *IEEE Transactions on Computers*, vol. 55, no. 4, Apr. 2006, pp. 366–372.

[127] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., and Chee, S., "HIGHT: A new block cipher suitable for low-resource device," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, Yokohama, Japan, Oct. 10–13, 2006, Lecture Notes in Computer Science, vol. 4249, Springer, pp. 46–59.

[128] Hsiao, S.F., Chen, M.C., Tsai, M.Y., and Lin, C.C., "System-on-chip implementation of the whole advanced encryption standard processor using reduced XOR-based sum-of-product operations," *IEE Proceedings: Information Security*, vol. 152, no. 1, Oct. 2005, pp. 21–30.

[129] Hsiao, S.F., Chen, M.C., and Tu, C.S., "Memory-free low-cost designs of advanced encryption standard using common subexpression elimination for subfunctions in transformations," *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, vol. 53, no. 3, Mar. 2006, pp. 615–626.

[130] Hwang, D.D., Tiri, K., Hodjat, A., Lai, B.C., Yang, S., Schaumont, P., and Verbauwhede, I., "AES-based security coprocessor IC in 0.18-μm CMOS with resistance to differential power analysis side-channel attacks," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, Apr. 2006, pp. 781–791.

[131] Ichikawa, T., Kasuya, T., and Matsui, M., "Hardware evaluation of the AES finalists," in *Proceedings of the 3rd Advanced Encryption Standard Candidate Conference*, New York, New York, USA, Apr. 13–14, 2000, pp. 279–285.

[132] Institute of Electrical and Electronics Engineers (IEEE), "IEEE standard specifications for public-key cryptography," *IEEE Std 1363-2000*, Jan. 30, 2002.

[133] ———, "IEEE standard specifications for public-key cryptography—amendment 1: Additional techniques," *IEEE Std 1363a-2004*, Jul. 22, 2004.

[134] International Organization for Standardization / International Electrotechnic Commission (ISO/IEC), "Encryption algorithms," *ISO/IEC 18033, Part 2*, 2002.

[135] ———, "Techniques based on elliptic curves," *ISO/IEC 15946, Parts 1-4*, 2002.

[136] Itoh, T. and Tsujii, S., "A fast algorithm for computing multplicative inverses in $GF(2^m)$ using normal bases," *Information and Computation*, vol. 78, no. 3, Sep. 1988, pp. 171–177.

[137] Järvinen, K., Tommiska, M., and Skyttä, J., "A VHDL generator for elliptic curve cryptography," in *Proceedings of the 14th International Conference on Field Programmable Logic and Application, FPL 2004*, Antwerp, Belgium, Aug. 29–Sep. 1, 2004, Lecture Notes in Computer Science, vol. 3203, Springer, pp. 1098–1100.

[138] ———, "A compact MD5 and SHA-1 co-implementation utilizing algorithm similarities," in *Proceedings of the 2005 International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA 2005*, Las Vegas, Nevada, USA, Jun. 27–30, 2005, pp. 48–54.

[139] ———, "Hardware implementation analysis of the MD5 hash algorithm," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS-38*, Big Island, Hawaii, USA, Jan. 3–6, 2005, p. 298 (abstract), full paper in the CD proceedings and IEEE Xplore.

[140] Johnson, D., Menezes, A., and Vanstone, S., "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, Aug. 2001, pp. 36–63.

[141] Joye, M. and Tymen, C., "Compact encoding of non-adjacent forms with applications to elliptic curve cryptography," in *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001*, Cheju Island, Korea, Feb. 13–15, 2001, Lecture Notes in Computer Science, vol. 1992, Springer, pp. 353–364.

[142] Kaliski, B.S., "The Montgomery inverse and its applications," *IEEE Transactions on Computers*, vol. 44, no. 8, Aug. 1995, pp. 1064–1065.

[143] Karatsuba, A. and Ofman, Y., "Multiplication of multidigit numbers on automata," *Doklady Akademii Nauk SSSR*, vol. 145, no. 2, Jul. 1962, pp. 293–294, English translation: *Soviet Physics — Doklady*, vol. 7, no. 7, Jan. 1963, pp. 595-596.

[144] Kerins, T., Popovici, E., Marnane, W., and Fitzpatrick, P., "Fully parameterizable elliptic curve cryptography processor over $GF(2^m)$," in *Proceedings of the 12th International Conference on Field Programmable Logic and Applications, FPL 2002*, Montpellier, France, Sep. 2–4, 2002, p. 750–759.

[145] Knudsen, E.W., "Elliptic scalar multiplication using point halving," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology — ASIACRYPT '99*, Singapore, Nov. 14–18, 1999, Lecture Notes in Computer Science, vol. 1716, Springer, pp. 135–149.

[146] Koblitz, N., "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, 1987, pp. 203–209.

[147] ———, "Hyperelliptic cryptosystems," *Journal of Cryptology*, vol. 1, no. 3, Oct. 1989, pp. 139–150.

[148] ———, "CM-curves with good cryptographic properties," in *Advances in Cryptology — CRYPTO '91*, Santa Barbara, California, USA, Aug. 11–15, 1991, Lecture Notes in Computer Science, vol. 576, Springer, pp. 279–287.

[149] Koç, Ç.K. and Acar, T., "Montgomery multiplication in $GF(2^k)$," *Designs, Codes and Cryptography*, vol. 14, no. 1, Apr. 1998, p. 57–69.

[150] Koç, Ç.K. and Sunar, B., "Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields," *IEEE Transactions on Computers*, vol. 47, no. 3, Mar. 1998, pp. 353–356.

[151] Kocher, P., Jaffe, J., and Jun, B., "Differential power analysis," in *Proceedings of the 19th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO '99*, Santa Barbara, California, USA, Aug. 15–19, 1999, Lecture Notes in Computer Science, vol. 1666, Springer, pp. 388–397.

[152] Kocher, P.C., "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proceedings of the 16th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO '96*, Santa Barbara, California, USA, Aug. 18–22, 1996, Lecture Notes in Computer Science, vol. 1109, Springer, pp. 104–113.

[153] Kosaraju, N.M., Varanasi, M., and Mohanty, S.P., "A high-performance VLSI architecture for advanced encryption standard (AES) algorithm," in *Proceedings of the 19th International Conference on VLSI Design, VLSID 2006*, Hyderabad, India, Jan. 3–7, 2006.

[154] Kotturi, D., Yoo, S.M., and Blizzard1, J., "AES crypto chip utilizing high-speed parallel pipelined architecture," in *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS 2005*, Kobe, Japan, May 23–26, 2005, vol. 5, pp. 4653–4656.

[155] Kravitz, D.W., "Digital signature algorithm," United States Patent 5,231,668, Jul. 27, 1993.

[156] Kumar, S., Wollinger, T., and Paar, C., "Optimum digit serial $GF(2^m)$ multipliers for curve-based cryptography," *IEEE Transactions on Computers*, vol. 55, no. 10, Oct. 2006, pp. 1306–1311.

[157] Kuo, H. and Verbauwhede, I., "Architectural optimization for a 1.82Gbits/sec VLSI implementation of the AES Rijndael algorithm," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 51–64.

[158] Kuon, I. and Rose, J., "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, Feb. 2007, pp. 203–215.

[159] Kwon, S., Gaj, K., Kim, C.H., and Hong, C.P., "Efficient linear array for multiplication in $GF(2^m)$ using a normal basis for elliptic curve cryptography," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2004*, Cambridge, Massachusetts, USA, Aug. 10–13, 2004, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 76–91.

[160] LaMacchia, B.A. and Manferdelli, J.L., "New Vistas in elliptic curve cryptography," *Information Security Technical Report*, vol. 11, no. 4, 2006, pp. 186–192.

[161] Lange, T., "A note on López-Dahab coordinates," Cryptology ePrint Archive, Report 2004/323, 2004, http://eprint.iacr.org/, (Oct. 9, 2008).

[162] ———, "Koblitz curve cryptosystems," *Finite Fields and Their Applications*, vol. 11, no. 2, Apr. 2005, pp. 200–229.

[163] Leander, G., Paar, C., Poschmann, A., and Schramm, K., "New lightweight DES variants," in *Revised Selected Papers of the 14th International Workshop on Fast Software Encryption, FSE 2007*, Luxemburg, Luxemburg, Mar. 26–28, 2007, Lecture Notes in Computer Science, vol. 4593, Springer, pp. 196–210.

[164] Lenstra, A.K. and Verheul, E.R., "Selecting cryptographic key sizes," *Journal of Cryptology*, vol. 14, no. 4, Dec. 2001, p. 255–293.

[165] Lenstra, H.W., "Factoring integers with elliptic curves," *The Annals of Mathematics*, vol. 126, no. 3, Nov. 1987, pp. 649–673.

[166] Leone, M., "A new low complexity parallel multiplier for a class of finite fields," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 160–170.

[167] Leong, P.H.W. and Leung, K.H., "A microcoded elliptic curve processor using FPGA technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 5, Oct. 2002, pp. 550–559.

[168] Leung, K.H., Ma, K.W., Wong, W.K., and Leong, P.H.W., "FPGA implementation of a microcoded elliptic curve cryptographic processor," in *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2000*, Napa Valley, California, USA, Apr. 17–19, 2000, pp. 68–76.

[169] Liberatori, M., Otero, F., Bonadero, J.C., and Castiñeira, J., "AES-128 cipher. high speed, low cost FPGA implementation," in *Proceedings of the 3rd Southern Conference on Programmable Logic, SPL 2007*, Mar del Plata, Argentina, Feb. 26–28, 2007, pp. 195–198.

[170] Lim, C.H. and Lee, P.J., "More flexible exponentiation with precomputation," in *Proceedings of the 14th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO '94*, Santa Barbara, California, USA, Aug. 21–25, 1994, Lecture Notes in Computer Science, vol. 839, Springer, pp. 95–107.

[171] Ling, S. and Xing, C., *Coding Theory: A First Course*, Cambridge University Press, 2004.

[172] Lipmaa, H., "AES/Rijndael: speed," Web page, http://research.cyber.ee/~lipmaa/research/aes/rijndael.html (Oct. 9, 2008).

[173] Liu, S., Bowen, F., King, B., and Wang, W., "Elliptic curves cryptosystem implementation based on a look-up table sharing scheme," in *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006*, Island of Kos, Greece, May 21–24, 2006, pp. 2921–2924.

[174] López, J. and Dahab, R., "Improved algorithms for elliptic curve arithmetic in $GF(2^n)$," in *Proceedings of the 5th Annual International Workshop on Selected Areas in Cryptography, SAC '98*, Kingston, Ontario, Canada, Aug. 17–18, 1998, Lecture Notes in Computer Science, vol. 1556, Springer, pp. 201–212.

[175] ———, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 1999*, Worcester, Massachusetts, USA, Aug. 12–13, 1999, Lecture Notes in Computer Science, vol. 1717, Springer, pp. 316–317.

[176] Lutz, A.K., Treichler, J., Gürkaynak, F.K., Kaeslin, H., Basler, G., Erni, A., Reichmuth, S., Rommens, P., Oetiker, S., and Fichtner, W., "2Gbit/s hardware realizations of RIJNDAEL and SERPENT: A comparative analysis," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 144–158.

[177] Lutz, J., *High Performance Elliptic Curve Cryptographic Co-processor*, Master's thesis, University of Waterloo, Canada, 2003.

[178] Lutz, J. and Hasan, A., "High performance FPGA based elliptic curve cryptographic co-processor," in *Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2004*, Las Vegas, Nevada, USA, Apr. 5–7, 2004, vol. 2, pp. 486–492.

[179] Mangard, S., Aigner, M., and Dominikus, S., "A highly regular and scalable AES hardware architecture," *IEEE Transactions on Computers*, vol. 52, no. 4, Apr. 2003, pp. 483–491.

[180] Mangard, S., Oswald, E., and Popp, T., *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Springer, 2007.

[181] Mastrovito, E.D., "VLSI designs for multiplication over finite fields $GF(2^m)$," in *Proceedings of the 6th International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, AAECC-6*, Rome, Italy, Jul. 4–8, 1988, Lecture Notes in Computer Science, vol. 357, Springer, pp. 297–309.

[182] ———, *VLSI architectures for computations in Galois fields*, Ph.D. thesis, Linköping University, 1991.

[183] Masuda, A.M., Moura, L., Panario, D., and Thomson, D., "Low complexity normal elements over finite fields of characteristic two," *IEEE Transactions on Computers*, vol. 57, no. 7, Jul. 2008, pp. 990–1001.

[184] McIvor, C.J., McLoone, M., and McCanny, J.V., "Hardware elliptic curve cryptographic processor over $GF(p)$," *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, vol. 53, no. 9, Sep. 2006, pp. 1946–1957.

[185] McLoone, M. and McCanny, J.V., "High performance single-chip FPGA Rijndael algorithm implementations," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 65–76.

[186] McLoone, M. and McCanny, J., "Rijndael FPGA implementation utilizing look-up tables," in *Proceedings of the 2001 IEEE Workshop on Signal Processing Systems, SIPS 2001*, Antwerp, Belgium, Sep. 26–28, 2001, pp. 349–360.

[187] Meier, W. and Staffelbach, O., "Efficient multiplication on certain nonsupersingular elliptic curves," in *Proceedings of the 12th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO '92*, Santa Barbara, California, USA, Aug. 16–20, 1992, Lecture Notes in Computer Science, vol. 740, Springer, pp. 333–344.

[188] Menezes, A.J., Okamoto, T., and Vanstone, S.A., "Reducing elliptic curve logarithms to logarithms in a finite field," *IEEE Transactions on Information Theory*, vol. 39, no. 5, Sep. 1993, pp. 1639–1646.

[189] Menezes, A.J., van Oorschot, P.C., and Vanstone, S.A., *Handbook of Applied Cryptography*, CRC Press, 1997.

[190] Mentens, N., Batina, L., Preneel, B., and Verbauwhede, I., "A systematic evaluation of compact hardware implementations for the Rijndael S-box," in *Proceedings of the Cryptographers' Track at the RSA Conference, Topics in Cryptology — CT-RSA 2005*, San Francisco, California, USA, Feb. 14–18, 2005, Lecture Notes in Computer Science, vol. 3376, Springer, pp. 323–333.

[191] Mentens, N., Örs, S.B., and Preneel, B., "An FPGA implementation of an elliptic curve processor over $GF(2^m)$," in *Proceedings of the 14th ACM Great Lakes symposium on VLSI, GLVLSI 2004*, Boston, Massachusetts, USA, Apr. 26-28, 2004, pp. 454–457.

[192] Meurice de Dormale, G., Bulens, P., and Quisquater, J.J., "Collision search for elliptic curve discrete logarithm over $GF(2^m)$ with FPGA," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2007*, Vienna, Austria, Sep. 10–13, 2007, Lecture Notes in Computer Science, vol. 4727, Springer, pp. 378–393.

[193] Meurice de Dormale, G. and Quisquater, J.J., "High-speed hardware implementations of elliptic curve cryptography: A survey," *Journal of Systems Architecture*, vol. 53, no. 2-3, Feb.–Mar. 2007, pp. 72–84.

[194] Miller, V., "Use of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85*, Santa Barbara, California, USA, Aug. 18–22, 1985, Lecture Notes in Computer Science, vol. 218, Springer, pp. 417–426.

[195] Mishra, P.K. and Sarkar, P., "Application of Montgomery's trick to scalar multiplication for elliptic and hyperelliptic curves using a fixed base point," in *Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography, PKC 2004*, Singapore, Mar. 1–4, 2004, Lecture Notes in Computer Science, vol. 2947, Springer, pp. 41–54.

[196] Montgomery, P.L., "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, Apr. 1985, pp. 519–521.

[197] ———, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, no. 177, Jan. 1987, pp. 243–264.

[198] Morain, F. and Olivos, J., "Speeding up the computations of an elliptic curve using addition-subtraction chains," *RAIRO Theoretical Informatics and Applications*, vol. 24, no. 6, 1990, pp. 531–543.

[199] Morales-Sandoval, M. and Feregrino-Uribe, C., "$GF(2^m)$ arithmetic modules for elliptic curve cryptography," in *Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGA's, ReConFig 2006*, San Luis Potosi, Mexico, Sep. 20–22, 2006.

[200] Morioka, S. and Satoh, A., "An optimized S-box circuit architecture for low power AES design," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 172–186.

[201] ———, "A 10-Gbps full-AES crypto design with a twisted BDD S-box architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 7, Jul. 2004, pp. 686–691.

[202] Mullin, R.C., Onyszchuk, I.M., Vanstone, S.A., and Wilson, R.M., "Optimal normal bases in $GF(p^n)$," *Discrete Applied Mathematics*, vol. 22, no. 2, 1989, pp. 149–161.

[203] National Institute of Standards and Technology (NIST), "Data encryption standard (DES)," *Federal Information Processing Standard, FIPS PUB 46*, Jan. 15, 1977.

[204] ———, "Data encryption standard (DES)," *Federal Information Processing Standard, FIPS PUB 46-3*, Oct. 25, 1999.

[205] ———, "Digital signature standard (DSS)," *Federal Information Processing Standard, FIPS PUB 186-2*, Jan. 27, 2000.

[206] ———, "Advanced encryption standard (AES)," *Federal Information Processing Standard, FIPS PUB 197*, Nov. 26, 2001.

[207] ———, "Secure hash standard (SHS)," *Federal Information Processing Standard, FIPS PUB 180-2*, Aug. 1, 2002.

[208] ———, "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA–3) family," *Federal Register*, vol. 72, no. 212, Nov. 2, 2007, pp. 62212–62220.

[209] ———, "Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC," *Special Publication 800-38D*, Nov. 2007.

[210] National Security Agency (NSA), "The case of elliptic curve cryptography," Web page, http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm (Oct. 9, 2008).

[211] Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., and Roback, E., "Report on the development of the Advanced Encryption Standard (AES)," *Techical report*, National Institute of Standards and Technology (NIST), Oct. 2, 2000.

[212] Nguyen, N., Gaj, K., Caliga, D., and El-Ghazawi, T., "Implementation of elliptic curve cryptosystems on a reconfigurable computer," in *Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology, FPT 2003*, Tokyo, Japan, Dec. 15–17, 2003, pp. 60–67.

[213] Ning, P. and Yin, Y.L., "Efficient software implementation for finite field multiplication in normal basis," in *Proceedings of the 3rd International Conference on Information and Communications Security, ICICS 2001*, Xian, China, Nov. 13—16, 2001, Lecture Notes in Computer Science, vol. 2229, Springer, pp. 177–188.

[214] Okada, S., Torii, N., Itoh, K., and Takenaka, M., "Implementation of elliptic curve cryptographic coprocessor over $GF(2^m)$ on an FPGA," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Worcester, Massachusetts, USA, Aug. 17–18, 2000, Lecture Notes in Computer Science, vol. 1965, Springer, pp. 25–40.

[215] Okeya, K. and Sakurai, K., "Fast multi-scalar multiplication methods on elliptic curves with precomputation strategy using Montgomery trick," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 564–578.

[216] Okeya, K., Takagi, T., and Vuillaume, C., "Short memory scalar multiplication on Koblitz curves," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2005*, Edinburgh, Scotland, UK, Aug. 29–Sep. 1, 2005, Lecture Notes in Computer Science, vol. 3659, Springer, pp. 91–105.

[217] Omura, J.K. and Massey, J.L., "Computational method and apparatus for finite field arithmetic," United States Patent 4,587,627, Oct. 6, 1986.

[218] OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS," Web page, http://www.openssl.org (Oct. 9, 2008).

[219] Orlando, G. and Paar, C., "A super-serial Galois fields multiplier for FPGAs and its application to public-key algorithms," in *Proceedings of the 7th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 1999*, Napa Valley, California, USA, Apr. 21–23, 1999, pp. 232–239.

[220] ———, "A high-performance reconfigurable elliptic curve processor for $GF(2^m)$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Worcester, Massachusetts, USA, Aug. 17–18, 2000, Lecture Notes in Computer Science, vol. 1965, Springer, pp. 41–56.

[221] ———, "A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 348–363.

[222] Örs, S.B., Batina, L., Preneel, B., and Vandewalle, J., "Hardware implementation of an elliptic curve processor over $GF(p)$," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors, ASAP 2003*, The Hague, The Netherlands, Jun. 24–26, 2003, pp. 433–443.

[223] Örs, S.B., Oswald, E., and Preneel, B., "Power-analysis attacks on an FPGA — first experimental results," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2003*, Cologne, Germany, Sep. 8–10, 2003, Lecture Notes in Computer Science, vol. 2779, Springer, pp. 35–50.

[224] Öztürk, E., Sunar, B., and Savaş, E., "Low-power elliptic curve cryptography using scaled modular arithmetic," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2004*, Cambridge, Massachusetts, USA, Aug. 10–13, 2004, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 92–106.

[225] Peter, S., Langendörfer, P., and Piotrowski, K., "Flexible hardware reduction for elliptic curve cryptography in $GF(2^m)$," in *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe, DATE 2007*, Nice, France, Apr. 16–20, 2007, pp. 1259–1264.

[226] Popp, T., Mangard, S., and Oswald, E., "Power analysis attacks and countermeasures," *IEEE Design and Test of Computers*, vol. 24, no. 6, Nov.–Dec. 2007, pp. 535–543.

[227] Potgieter, M.J. and van Dyk, B.J., "Two hardware implementations of the group operations necessary for implementing an elliptic curve cryptosystem over a characteristic two finite field," in *Proceedings of the 6th IEEE Africon Conference in Africa, Africon 2002*, George, South Africa, Oct. 2-4, 2002, vol. 1, pp. 187–192.

[228] Pramstaller, N. and Wolkerstorfer, J., "A universal and efficient AES coprocessor for field programmable logic arrays," in *Proceedings of the 14th International Conference on Field Programmable Logic and Application, FPL 2004*, Antwerp, Belgium, Aug. 30–Sep. 1, 2004, Lecture Notes in Computer Science, vol. 3203, Springer, pp. 565–574.

[229] Preneel, B., "A survey of recent developments in cryptographic algorithms for smart cards," *Computer Networks*, vol. 51, no. 9, Jun. 2007, pp. 2223–2233.

[230] Proos, J., "Joint sparse forms and generating zero columns when combing," *Techical Report CORR 2003-23*, University of Waterloo, Canada, Jul. 7, 2003.

[231] Ravi, S., Raghunathan, A., Kocher, P., and Hattangady, S., "Security in embedded systems: Design challenges," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, Aug. 2004, pp. 461–491.

[232] Reyhani-Masoleh, A., "Efficient algorithms and architectures for field multiplication using Gaussian normal bases," *IEEE Transactions on Computers*, vol. 55, no. 1, Jan. 2006, pp. 34–47.

[233] Reyhani-Masoleh, A. and Hasan, M.A., "On low complexity bit parallel polynomial basis multipliers," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2003*, Cologne, Germany, Sep. 8–10, 2003, Lecture Notes in Computer Science, vol. 2779, Springer, pp. 189–202.

[234] ———, "Efficient digit-serial normal basis multipliers over binary extension fields," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, Aug. 2004, pp. 575–592.

[235] ———, "Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 53, no. 8, Aug. 2004, pp. 945–959.

[236] ———, "Low complexity word-level sequential normal basis multipliers," *IEEE Transactions on Computers*, vol. 54, no. 2, Feb. 2005, pp. 98–110.

[237] Rijmen, V., "Efficient implementation of the Rijndael S-box," Online document, http://citeseer.ist.psu.edu/293912.html (Oct. 9, 2008).

[238] Rivest, R., "The MD5 message-digest algorithm," Request for Comments, RFC 1321, Apr. 1992, http://www.ietf.org/rfc/rfc1321.txt (Oct. 9, 2008).

[239] Rivest, R.L., Shamir, A., and Adleman, L., "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, Feb. 1978, pp. 120–126.

[240] Rodríguez-Henríquez, F., Saqib, N.A., and Díaz-Pérez, A., "4.2 Gbit/s single-chip FPGA implementation of AES algorithm," *Electronics Letters*, vol. 39, no. 15, Jul. 24, 2003, pp. 1115–1116.

[241] ———, "A fast parallel implementation of elliptic curve point multiplication over $GF(2^m)$," *Microprocessors and Microsystems*, vol. 28, no. 5–6, Aug. 2004, pp. 329–339.

[242] Rosing, M., *Implementing Elliptic Curve Cryptography*, Manning Publications Co., 1999.

[243] Rosner, M.C., *Elliptic Curve Cryptosystems on Reconfigurable Hardware*, Master's thesis, Worcester Polytechnic Institute, 1998.

[244] Rouvroy, G., Standaert, F.X., Quisquater, J.J., and Legat, J.D., "Efficient uses of FPGAs for implementations of DES and its experimental linear cryptanalysis," *IEEE Transactions on Computers*, vol. 52, no. 4, Apr. 2003, pp. 473–482.

[245] ———, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," in *Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2004*, Las Vegas, Nevada, USA, Apr. 5–7, 2004, vol. 2, pp. 583–587.

[246] RSA Security, Inc., "The RSA factoring challenge," Web page, http://www.rsa.com/rsalabs/node.asp?id=2092 (Oct. 9, 2008).

[247] Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J.R., and Rohatgi, P., "Efficient rijndael encryption implementation with composite field arithmetic," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14–16, 2001, Lecture Notes in Computer Science, vol. 2162, Springer, pp. 171–184.

[248] Saggese, G.P., Mazzeo, A., Mazzocca, N., and Strollo, A.G.M., "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm," in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications, FPL 2003*, Lisbon, Portugal, Sep. 1–3, 2003, Lecture Notes in Computer Science, vol. 2778, Springer, pp. 292–302.

[249] Sakiyama, K., Batina, L., Preneel, B., and Verbauwhede, I., "Superscalar coprocessor for high-speed curve-based cryptography," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, Yokohama, Japan, Oct. 10–13, 2006, Lecture Notes in Computer Science, vol. 4249, Springer, pp. 415–429.

[250] ———, "Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over $GF(2^n)$," *IEEE Transactions on Computers*, vol. 56, no. 9, Sep. 2007, pp. 1269–1282.

[251] Sakiyama, K., Mentens, N., Batina, L., Preneel, B., and Verbauwhede, I., "Reconfigurable modular arithmetic logic unit supporting high-performance RSA and ECC over $GF(p)$," *International Journal of Electronics*, vol. 94, no. 5, May 2007, pp. 501–514.

[252] Saqib, N.A., Rodríguez-Henríquez, F., and Díaz-Pérez, A., "A parallel architecture for fast computation of elliptic curve scalar multiplication over $GF(2^m)$," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium, IPDPS 2004*, Santa Fe, New Mexico, USA, Apr. 26–30, 2004.

[253] Satoh, A., "High-speed hardware architectures for authenticated encryption mode GCM," in *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006*, Island of Kos, Greece, May 21–24, 2006, pp. 4831–4834.

[254] Satoh, A., Morioka, S., Takano, K., and Munetoh, S., "A compact Rijndael hardware architecture with S-box optimization," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology — ASIACRYPT 2001*, Gold Coast, Australia, Dec. 9–13, 2001, Lecture Notes in Computer Science, vol. 2248, Springer, pp. 239–254.

[255] Satoh, A. and Takano, K., "A scalable dual-field elliptic curve cryptographic processor," *IEEE Transactions on Computers*, vol. 52, no. 4, Apr. 2003, pp. 449–460.

[256] Savaş, E. and Koç, Ç.K., "The Montgomery modular inverse—revisited," *IEEE Transactions on Computers*, vol. 49, no. 7, Jul. 2000, pp. 763–766.

[257] Savaş, E., Tenca, A.F., and Koç, Ç.K., "A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Worcester, Massachusetts, USA, Aug. 17–18, 2000, Lecture Notes in Computer Science, vol. 1965, Springer, pp. 277–292.

[258] Schneier, B., *Applied Cryptography*, John Wiley & Sons, Inc., 2nd ed., 1996.

[259] Schroeppel, R., Beaver, C., Gonzales, R., Miller, R., and Draelos, T., "A low-power design for an elliptic curve digital signature chip," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 366–381.

[260] Schroeppel, R., Orman, H., O'Malley, S., and Spatscheck, O., "Fast key exchange with elliptic curve systems," in *Proceedings of the 15th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO '95*, Santa Barbara, California, USA, Aug. 27–31, 1995, Lecture Notes in Computer Science, vol. 963, Springer, pp. 43–56.

[261] Shacham, H. and Boneh, D., "Improving SSL handshake performance via batching," in *Proceedings of the Cryptographers' Track at the RSA*

Conference 2001, Topics in Cryptology — CT-RSA 2001, San Francisco, California, USA, Apr. 8–12, 2001, Lecture Notes in Computer Science, vol. 2020, Springer, pp. 28–43.

[262] Shu, C., Gaj, K., and El-Ghazawi, T., "Low latency elliptic curve cryptography accelerators for NIST curves over binary fields," in *Proceedings of the 2005 IEEE International Conference on Field Programmable Technology, FPT 2005*, Singapore, Dec. 11–14, 2005, pp. 309–310.

[263] Smart, N.P., "How secure are elliptic curves over composite extension fields?" in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology — EUROCRYPT 2001*, Aarhus, Denmark, May 21–23, 2001, Lecture Notes in Computer Science, vol. 2045, Springer, pp. 30–39.

[264] Solinas, J.A., "An improved algorithm for arithmetic on a family of elliptic curves," in *Proceedings of the 17th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO '97*, Santa Barbara, California, USA, Aug. 17–21, 1997, Lecture Notes in Computer Science, vol. 1294, Springer, pp. 357–371.

[265] ———, "Generalized Mersenne numbers," *Techical Report CORR 99-39*, University of Waterloo, Canada, 1999.

[266] ———, "Improved algorithms for arithmetic on anomalous binary curves," *Techical Report CORR 99-46*, University of Waterloo, Canada, 1999, corrected and updated version of [264].

[267] ———, "Efficient arithmetic on Koblitz curves," *Designs, Codes and Cryptography*, vol. 19, no. 2–3, 2000, pp. 195–249.

[268] ———, "Low-weight binary representations for pairs of integers," *Techical Report CORR 2001-41*, University of Waterloo, Canada, 2001.

[269] Song, L. and Parhi, K.K., "Low-energy digit-serial/parallel finite field multipliers," *Journal of VLSI Signal Processing*, vol. 19, no. 2, Jul. 1998, pp. 149–166.

[270] Sozzani, F., Bertoni, G., Turcato, S., and Breveglieri, L., "A parallelized design for an elliptic curve cryptosystem coprocessor," in *Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2005*, Las Vegas, Nevada, USA, Apr. 4–6, 2005, vol. 1, pp. 626–630.

[271] Standaert, F.X., Peeters, E., Rouvroy, G., and Quisquater, J.J., "An overview of power analysis attacks against field programmable gate arrays," *Proceedings of the IEEE*, vol. 94, no. 2, Feb. 2006, pp. 383–394.

[272] Standaert, F.X., Rouvroy, G., Quisquater, J.J., and Legat, J.D., "Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2003*, Cologne, Germany, Sep. 8–10, 2003, Lecture Notes in Computer Science, vol. 2779, Springer, pp. 334–350.

[273] Stein, J., "Computational problems associated with Racah algebra," *Journal of Computational Physics*, vol. 1, no. 3, Feb. 1967, pp. 397–405.

[274] Stinson, D.R., *Cryptography Theory and Practice*, Chapman & Hall/CRC, 2nd ed., 2002.

[275] Su, C.P., Lin, T.F., Huang, C.T., and Wu, C.W., "A high-throughput low-cost AES processor," *IEEE Communications Magazine*, vol. 41, no. 12, Dec. 2003, pp. 86–91.

[276] Sun Microsystems, Inc., "UltraSPARC T2 processor datasheet," Data sheet, 2007, http://www.sun.com/processors/UltraSPARC-T2/datasheet.pdf (Oct. 9, 2008).

[277] Sunar, B. and Ç. K. Koç, "Mastrovito multiplier for all trinomials," *IEEE Transactions on Computers*, vol. 48, no. 5, May 1999, pp. 522–527.

[278] ———, "An efficient optimal normal basis type II multiplier," *IEEE Transactions on Computers*, vol. 50, no. 1, Jan. 2001, pp. 83–87.

[279] Tillich, S. and Großschädl, J., "Instruction set extensions for efficient AES implementation on 32-bit processors," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, Yokohama, Japan, Oct. 10–13, 2006, Lecture Notes in Computer Science, vol. 4249, Springer, pp. 270–284.

[280] Todman, T.J., Constantinides, G.A., Wilton, S.J.E., Mencer, O., Luk, W., and Cheung, P.Y.K., "Reconfigurable computing: architectures and design methods," *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 2, Mar. 2005, pp. 193–207.

[281] Tommiska, M., *Applications of Reprogrammability in Algorithm Acceleration*, Ph.D. thesis, Helsinki University of Technology, 2005.

[282] Verbauwhede, I., Schaumont, P., and Kuo, H., "Design and performance testing of a 2.29-GB/s Rijndael processor," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, Mar. 2003, pp. 569–572.

[283] von zur Gathen, J. and Shokrollahi, J., "Efficient FPGA-based Karatsuba multipliers for polynomials over $\mathbb{F}_2$," in *Revised Selected Papers of the 12th International Workshop on Selected Areas in Cryptography, SAC 2005*, Kingston, Ontario, Canada, Aug. 10–12, 2005, Lecture Notes in Computer Science, vol. 3897, Springer, pp. 359–369.

[284] Vuillaume, C., Okeya, K., and Takagi, T., "Short memory scalar multiplication for koblitz curves," *IEEE Transactions on Computers*, vol. 57, no. 4, 2008, pp. 481–489.

[285] Wang, C.C., Troung, T.K., Shao, H.M., Deutsch, L.J., Omura, J.K., and Reed, I.S., "VLSI architectures for computing multiplications and inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 34, no. 8, Aug. 1985, pp. 709–717.

[286] Wang, X., Yin, Y.L., and Yu, H., "Finding collisions in the full SHA-1," in *Proceedings of the 25th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO 2005*, Santa Barbara, California, USA, Aug. 14–18, 2005, Lecture Notes in Computes Science, vol. 3621, pp. 17–36.

[287] Wang, X. and Yu, H., "How to break MD5 and other hash functions," in *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Advances in Cryptology — EUROCRYPT 2005*, Aarhus, Denmark, May 22–26, 2005, Lecture Notes in Computer Science, vol. 3494, Springer, pp. 19–35.

[288] Weeks, B., Bean, M., Rozylowicz, T., and Ficke, C., "Hardware performance simulations of round 2 advanced encryption standard algorithms," in *Proceedings of the 3rd Advanced Encryption Standard Candidate Conference*, New York, New York, USA, Apr. 13–14, 2000, pp. 286–304.

[289] Wiles, A., "Modular elliptic curves and Fermat's last theorem," *The Annals of Mathematics*, vol. 142, no. 3, May 1995, pp. 443–551.

[290] Wolf, W., *FPGA-Based System Design*, Prectice Hall, 2004.

[291] Wolkerstorfer, J., "Dual-field arithmetic unit for $GF(p)$ and $GF(2^m)$," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002*, Redwood City, California, USA, Aug. 13–15, 2002, Lecture Notes in Computer Science, vol. 2523, Springer, pp. 500–514.

[292] Wolkerstorfer, J., Oswald, E., and Lamberger, M., "An ASIC implementation of the AES Sboxes," in *Proceedings of the Cryptographers' Track at the RSA Conference, Topics in Cryptology — CT-RSA 2002*, San Jose, California, USA, Feb. 18–22, 2002, Lecture Notes in Computer Science, vol. 2271, Springer, pp. 67–78.

[293] Wollinger, T., Guajardo, J., and Paar, C., "Security on FPGAs: State-of-the-art implementations and attacks," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, Aug. 2004, pp. 534–574.

[294] Wollinger, T. and Paar, C., "How secure are FPGAs in cryptographic applications?" in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications, FPL 2003*, Lisbon, Portugal, Sep. 1–3, 2003, Lecture Notes in Computer Science, vol. 2778, Springer, pp. 91–100.

[295] Wu, H., "Low complexity bit-parallel finite field arithmetic using polynomial basis," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 1999*, Worcester, Massachusetts, USA, Aug. 12–13, 1999, Lecture Notes in Computer Science, vol. 1717, Springer, pp. 280–291.

[296] ———, "On complexity of polynomial basis squaring in $\mathbb{F}_{2^m}$," in *Proceedings of the 7th Annual International Workshop on Selected Areas in Cryptography, SAC 2000*, Waterloo, Ontario, Canada, Aug. 14–15, 2000, Lecture Notes in Computer Science, vol. 2012, Springer, pp. 118–129.

[297] ———, "Bit-parallel finite field multiplier and squarer using polynomial basis," *IEEE Transactions on Computers*, vol. 51, no. 7, Jul. 2002, pp. 750–758.

[298] ———, "Montgomery multiplier and squarer for a class of finite fields," *IEEE Transactions on Computers*, vol. 51, no. 5, May 2002, pp. 521–529.

[299] Xilinx, Inc., "Virtex-E 1.8 V field programmable gate arrays," Data sheet, Jul. 17, 2002, http://www.xilinx.com/support/documentation/data_sheets/ds022.pdf (Oct. 9, 2008).

[300] ———, "MicroBlaze processor reference guide," Data sheet, Jun. 25, 2007, http://www.xilinx.com/support/documentation/sw_manuals/edk92i_mb_ref_guide.pdf (Oct. 9, 2008).

[301] ———, "Virtex-4 family overview," Data sheet, Sep. 28, 2007, http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf (Oct. 9, 2008).

[302] ———, "Virtex-II platform FPGAs: Complete data sheet," Data sheet, Nov. 5, 2007, http://www.xilinx.com/support/documentation/data_sheets/ds031.pdf (Oct. 9, 2008).

[303] ———, "Virtex-5 family overview," Data sheet, Sep. 23, 2008, http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf (Oct. 9, 2008).

[304] Zambreno, J., Nguyen, D., and Choudhary, A., "Exploring area/delay tradeoffs in an AES FPGA implementation," in *Proceedings of the 14th International Conference on Field Programmable Logic and Application, FPL 2004*, Antwerp, Belgium, Aug. 30–Sep. 1, 2004, Lecture Notes in Computer Science, vol. 3203, Springer, pp. 575–585.

[305] Zhang, T. and Parhi, K.K., "Systematic design of original and modified Mastrovito multipliers for general irreducible polynomials," *IEEE Transactions on Computers*, vol. 50, no. 7, Jul. 2001, pp. 734–749.

[306] Zhang, X. and Parhi, K.K., "Implementation approaches for the advanced encryption standard algorithm," *IEEE Circuits and Systems Magazine*, vol. 2, no. 4, 2002, pp. 24–46.

[307] ———, "High-speed VLSI architectures for the AES algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, Sep. 2004, pp. 957–967.

[308] ———, "On the optimum constructions of composite field for the AES algorithm," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, vol. 53, no. 10, Oct. 2006, pp. 1153–1157.

[309] Zhou, G., Michalik, H., and Hinsenkamp, L., "Efficient and high-throughput implementations of AES-GCM on FPGAs," in *Proceedings of the 2007 International Conference on Field Programmable Technology, FPT 2007*, Kitakyushu, Japan, Dec. 12–14, 2007, pp. 185–192.

[310] Zhou, J.Y., Liu, X.W., and Jiang, X.G., "Implementing elliptic curve cryptography on Nios II processor," in *Proceedings of the 7th International Conference on ASIC, ASICON 2007*, Guilin, China, Oct. 26–29, 2007, pp. 205–208.