

Doctoral Thesis University of Patras No. 309

Versatile Architectures for Cryptographic Systems

A dissertation submitted to the
Department of Electrical & Computer Engineering
University of Patras

for the degree of

DOCTOR OF PHILOSOPHY

presented by

DIMITRIOS SCHINIANAKIS

Dipl. Eng., University of Patras, Greece
born 21.01.1983 in Athens
Greek citizen

2013

Doctoral Thesis University of Patras No. 309

Versatile Architectures for Cryptographic Systems

A dissertation submitted to the
Department of Electrical & Computer Engineering
University of Patras

for the degree of

DOCTOR OF PHILOSOPHY

presented by

DIMITRIOS SCHINIANAKIS

Dipl. Eng., University of Patras, Greece
born 21.01.1983 in Athens
Greek citizen

2013



This research has been co-financed by the European Union (European Social Fund–ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η παρούσα διδακτορική διατριβή με τίτλο

Ευέλικτες αρχιτεκτονικές συστημάτων κρυπτογραφίας (Ελλ.)

Versatile architectures for cryptographic systems (Eng.)

του Σχοινιανάκη Δημητρίου του Μιχαήλ, διπλωματούχου Ηλεκτρολόγου Μηχανικού & Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών, παρουσιάστηκε δημοσίως στην αίθουσα συνεδριάσεων του τμήματος Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών στις 11/10/2013, εξετάστηκε και εγκρίθηκε από την ακόλουθη Εξεταστική Επιτροπή:

Στουραϊτης Αθανάσιος, Καθηγητής Πολυτεχνικής Σχολής Πανεπιστημίου Πατρών (Τμ. Ηλ. Μηχ. & Τεχν. Υπολ.)

Κουφοπαύλου Οδυσσέας, Καθηγητής Πολυτεχνικής Σχολής Πανεπιστημίου Πατρών (Τμ. Ηλ. Μηχ. & Τεχν. Υπολ.)

Ζαρολιάγκης Χρήστος, Καθηγητής Πολυτεχνικής Σχολής Πανεπιστημίου Πατρών (Τμ. Μηχ. Η/Υ & Πληροφ.)

Σερπάνος Δημήτριος, Καθηγητής Πολυτεχνικής Σχολής Πανεπιστημίου Πατρών (Τμ. Ηλ. Μηχ. & Τεχν. Υπολ.)

Παλιουράς Βασίλειος, Επίκουρος Καθηγητής Πολυτεχνικής Σχολής Πανεπιστημίου Πατρών (Τμ. Ηλ. Μηχ. & Τεχν. Υπολ.)

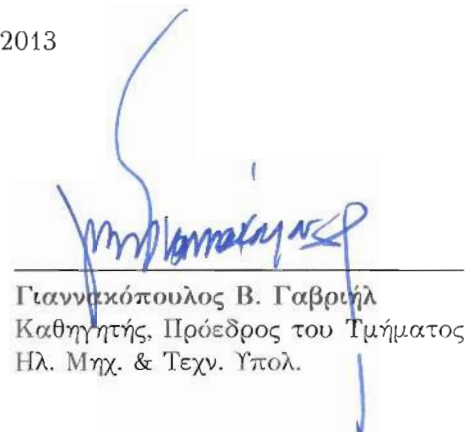
Θεοδωρίδης Γεώργιος, Επίκουρος Καθηγητής Πολυτεχνικής Σχολής Πανεπιστημίου Πατρών (Τμ. Ηλ. Μηχ. & Τεχν. Υπολ.)

Κωνσταντίνου Ελισάβετ, Επίκουρη Καθηγήτρια Πανεπιστημίου Αιγαίου (Τμ. Μηχ. Πληροφ. & Επικοινων. Συστ.)

Πάτρα, 11 Οκτωβρίου 2013



Στουραϊτης Αθανάσιος
Επιβλέπων, Καθηγητής Τμήματος
Ηλ. Μηχ. & Τεχν. Υπολ.



Γιαννακόπουλος Β. Γαβριήλ
Καθηγητής, Πρόεδρος του Τμήματος
Ηλ. Μηχ. & Τεχν. Υπολ.

“Aut viam inveniam aut faciam”
(Either I shall find a way or I shall make one)
Hannibal, 247–183 BCE



Abstract

We live in a world that demands more and more connectivity. Our everyday life is dominated by an overwhelming amount of information that needs to be controlled and maintained. Everyday transactions that few years ago required our physical existence, have been replaced by electronic applications. Users may now use friendly, fast and safe interfaces to perform easily numerous tasks, varying from money transfers using the web and remote health checks to e-learning and e-commerce.

Being exposed in an unprecedented number of threats and frauds, safe connectivity for all network-based systems has now become a predicate necessity. The science of cryptography provides the necessary tools and means towards this direction. Cryptographic hardware and software play now a dominant role in e-commerce, mobile phone communications, military applications, private emails, digital signatures for e-commerce, ATM cards, web banking, maintenance of health records and so on.

This doctoral thesis approaches the problem of designing versatile architectures for cryptographic hardware. By the term versatile we define hardware architectures capable of supporting a variety of arithmetic operations and algorithms useful in cryptography, with no need to reconfigure the internal interconnections of the integrated circuit.

A versatile architecture could offer considerable benefits to the end-user. By embedding a variety of crucial operations in a common architecture, the user is able to switch seamlessly the underlying cryptographic protocols. This not only gives an added value in the design from flexibility but also from practicality point of view. The total cost of a cryptographic application can also be benefited; assuming a versatile integrated circuit which requires no additional circuitry for other vital operations (for example input–output converters) it is easy to deduce that the total cost of development and fabrication of these extra components is eliminated, thus reducing the total production cost.

We follow a systematic approach for developing and presenting the proposed versatile architectures. First, an in-depth analysis of the algorithms of interest is carried out, in order to

identify new research areas and weaknesses of existing solutions. The proposed algorithms and architectures operate on Galois Fields GF of the form $GF(p)$ for integers and $GF(2^n)$ for polynomials. Alternative number representation systems such as Residue Number System (RNS) for integers and Polynomial Residue Number System (PRNS) for polynomials are employed. The mathematical validity of the proposed algorithms and the applicability of RNS and PRNS in the context of cryptographic algorithms is also presented. The derived algorithms are decomposed in a way that versatile structures can be formulated and the corresponding hardware is developed and evaluated. New cryptanalytic properties of the proposed algorithms against certain types of attacks are also highlighted.

Furthermore, we try to approach a fundamental problem in Very Large Scale Integration (VLSI) design, that is the problem of evaluating and comparing architectures using models independent from the underlying fabrication technology. Generic methods to evaluate the optimal operation parameters of the proposed architectures and methods to optimize the proposed architectures in terms of speed, area, and area \times speed product, based on the needs of the underlying application are provided. The proposed methodologies can be expanded to include applications other than cryptography.

Finally, novel algorithms based on new mathematical and design problems for the crucial operation of modular multiplication are presented. The new algorithms preserve the versatile characteristics discussed previously and it is proved that, along with existing algorithms in the literature, they may form a large family of algorithms applicable in cryptography, unified under the common frame of the proposed versatile architectures.



Περίληψη της διατριβής στα Ελληνικά

Η σύγχρονη ζωή βασίζεται εν πολλοίς στη διασύνδεση. Καθημερινά βαλλόμαστε από καταϊγισμό πληροφοριών οι οποίες πρέπει να μεταφέρονται, να αποθηκεύονται, και να αξιολογούνται ως προς την ακεραιότητά τους σε ελάχιστο χρόνο, προκειμένου η ροή τους να είναι συνεχής και απρόσκοπτη. Καθημερινές συναλλαγές που πριν από μερικά χρόνια θα ήταν αδύνατες χωρίς τη φυσική μας παρουσία, έχουν πλέον αντικατασταθεί από ηλεκτρονικές εφαρμογές. Οι χρήστες μπορούν πλέον μέσω φιλικών, γρήγορων και κυρίως ασφαλών εφαρμογών να ελέγχουν και να πραγματοποιούν πληθώρα εργασιών, από τραπεζικές συναλλαγές και αγορές αγαθών μέχρι εφαρμογές τηλεϊατρικής και απομακρυσμένης εκπαίδευσης.

Παράλληλα όμως με την ανάπτυξη κάθε είδους ηλεκτρονικής και διαδικτυακής διευκόλυνσης, αναπτύχθηκε συνακόλουθα μια ευρεία κατηγορία απειλών και επιθέσεων ενάντια σε τέτοια συστήματα, γεγονός που κατέστησε την ασφάλεια πρωταρχικό πεδίο έρευνας και ανάπτυξης και μείζονα λειτουργικό παράγοντα για τον σύγχρονο σχεδιαστή ηλεκτρονικών συστημάτων. Η επιστήμη της κρυπτογραφίας έρχεται να παίξει στο σημείο αυτό πρωταρχικό ρόλο στην παροχή της επιθυμητής ασφάλειας. Κρυπτογραφικές εφαρμογές και ειδικά κρυπτογραφικά κυκλώματα μεγάλης ολοκλήρωσης βρίσκονται πλέον εγκατεστημένα σε όλες τις κρίσιμες εφαρμογές που απαιτούν υψηλά επίπεδα ασφάλειας, όπως συστήματα ηλεκτρονικού εμπορίου, συστήματα κινητής τηλεφωνίας (μαζί με τις συνακόλουθες εφαρμογές), στρατιωτικές εφαρμογές, ιδιωτικό ηλεκτρονικό ταχυδρομείο, ψηφιακές υπογραφές, τραπεζικές κάρτες ATM, διατήρηση ευαίσθητων ηλεκτρονικών αρχείων υγείας ασθενών κ.α.

Η παρούσα διατριβή άπτεται του θέματος της ανάπτυξης ευέλικτων αρχιτεκτονικών κρυπτογραφίας σε ολοκληρωμένα κυκλώματα υψηλής ολοκλήρωσης (VLSI). Με τον όρο ευέλικτες ορίζονται οι αρχιτεκτονικές που δύνανται να υλοποιούν πλήθος βασικών αριθμητικών πράξεων για την εκτέλεση κρυπτογραφικών αλγορίθμων, χωρίς την ανάγκη επαναπροσδιορισμού των εσωτερικών διατάξεων στο ολοκληρωμένο κύκλωμα.

Η χρήση ευέλικτων αρχιτεκτονικών παρέχει πολλαπλά οφέλη στο χρήστη. Η ενσωμάτωση

κρίσιμων πράξεων απαραίτητων στη κρυπτογραφία σε μια κοινή αρχιτεκτονική δίνει τη δυνατότητα στο χρήστη να εναλλάσσει το υποστηριζόμενο κρυπτογραφικό πρωτόκολλο, εισάγοντας έτσι χαρακτηριστικά ευελιξίας και πρακτικότητας, χωρίς επιπρόσθετη επιβάρυνση του συστήματος σε υλικό. Αξίζει να σημειωθεί πως οι εναλλαγές αυτές δεν απαιτούν την παρέμβαση του χρήστη. Σημαντική είναι η συνεισφορά μιας ευέλικτης αρχιτεκτονικής και στο κόστος μιας εφαρμογής. Αναλογιζόμενοι ένα ολοκληρωμένο κύκλωμα που μπορεί να υλοποιεί αυτόνομα όλες τις απαραίτητες πράξεις ενός αλγόριθμου χωρίς την εξάρτηση από εξωτερικά υποσυστήματα (π.χ. μετατροπείς εισόδου-εξόδου), είναι εύκολο να αντιληφθούμε πως το τελικό κόστος της εκάστοτε εφαρμογής μειώνεται σημαντικά καθώς μειώνονται οι ανάγκες υλοποίησης και διασύνδεσης επιπρόσθετων υποσυστημάτων στο ολοκληρωμένο κύκλωμα.

Η ανάπτυξη των προτεινόμενων αρχιτεκτονικών ακολουθεί μια δομημένη προσέγγιση. Διενεργείται εκτενής μελέτη για τον προσδιορισμό γόνιμων ερευνητικών περιοχών και εντοπίζονται προβλήματα και δυνατότητες βελτιστοποίησης υπάρχουσών κρυπτογραφικών λύσεων. Οι νέοι αλγόριθμοι που αναπτύσσονται αφορούν τα Galois πεδία $GF(p)$ και $GF(2^n)$ και χρησιμοποιούν εναλλακτικές αριθμητικές αναπαράστασης δεδομένων, όπως το αριθμητικό σύστημα υπολοίπων (Residue Number System (RNS)) για ακέραιους αριθμούς και το πολυωνυμικό αριθμητικό σύστημα υπολοίπων (Polynomial Residue Number System (PRNS)) για πολυώνυμα. Αποδεικνύεται η μαθηματική τους ορθότητα και βελτιστοποιούνται κατά τέτοιο τρόπο ώστε να σχηματίζουν ευέλικτες δομές. Αναπτύσσεται το κατάλληλο υλικό (hardware) και διενεργείται μελέτη χρήσιμων ιδιοτήτων των νέων αλγορίθμων, όπως για παράδειγμα νέες κρυπταναλυτικές ιδιότητες.

Επιπρόσθετα, προσεγγίζεται στα πλαίσια της διατριβής ένα βασικό πρόβλημα της επιστήμης σχεδιασμού ολοκληρωμένων συστημάτων μεγάλης κλίμακας (VLSI). Συγκεκριμένα, προτείνονται μέθοδοι σύγκρισης αρχιτεκτονικών ανεξαρτήτως τεχνολογίας καθώς και τρόποι εύρεσης των βέλτιστων συνθηκών λειτουργίας των προτεινόμενων αρχιτεκτονικών. Οι μέθοδοι αυτές επιτρέπουν στο σχεδιαστή να παραμετροποιήσει τις προτεινόμενες αρχιτεκτονικές με βάση την ταχύτητα, επιφάνεια ή το γινόμενο ταχύτητα \times επιφάνεια. Οι προτεινόμενες μεθοδολογίες μπορούν εύκολα να επεκταθούν και σε άλλες εφαρμογές πέραν της κρυπτογραφίας.

Τέλος, προτείνονται νέοι αλγόριθμοι για τη σημαντικότερη για την κρυπτογραφία πράξη του πολλαπλασιασμού με υπόλοιπα. Οι νέοι αλγόριθμοι ενσωματώνουν από τη μία τις ιδέες των ευέλικτων δομών, από την άλλη όμως βασίζονται σε νέες ιδέες και μαθηματικά προβλήματα τα οποία προσπαθούμε να προσεγγίσουμε και να επιλύσουμε. Αποδεικνύεται πως είναι δυνατή η ενοποίηση μιας μεγάλης οικογένειας αλγορίθμων για χρήση στην κρυπτογραφία, υπό τη στέγη των προτεινόμενων μεθοδολογιών για ευέλικτο σχεδιασμό.



Acknowledgements

I owe my deepest gratitude to my supervisor, Professor Thanos Stouraitis, for his excellent guidance throughout the 8 years of my Ph.D. studies. I consider myself more than lucky to have collaborated with him in the development of this doctoral thesis. I would like to gratefully thank him not only as a supervisor, but also as a friend. Thanos stood for me as a real source of inspiration, an example of out-of-the-box thinking, a paradigm of devotion and faith to hard work. This thesis would not have been possible without his exemplary support and his continuous and close supervision.

During all those years, I had the opportunity to collaborate with several remarkable people, who influenced my work and provided me with invaluable help in my research. I am grateful to my friends Dr. Athanasios Kakarountas, and Dr. Charalampos Michail for the collaboration and cooperation during the first years of my Ph.D. studies. The roots of this thesis share a lot of common thoughts and efforts.

I would also like to thank Mrs. Fotopoulou Eleni and Mr. Ferentinos Aris for our collaboration and all the fun moments we had in the Digital Signal & Image Processing Lab. Furthermore, I owe my deepest gratitude to my co-supervisors Prof. Odysseas Koufopavlou and Prof. Christos Zaroliagis, for the valuable comments, guidance and support during the preparation of this dissertation.

Last, but not least, my family and friends deserve my deepest gratitude for their support and belief in me throughout those 8 years. I sincerely believe that this work would not have been as successful without them.



Contents

Abstract	i
Abstract in Greek	iii
Acknowledgements	v
Contents	vi
List of Figures	x
List of Tables	xii
Acronyms	xv
Notation	xix
1 Introduction	1
1.1 Overview	2
1.2 Design challenges and motivation	3
1.3 Thesis overview and contributions	4
2 Mathematical Background	9
2.1 Basics on finite-field theory	10
2.1.1 $GF(p)$ arithmetic	11
2.1.1.1 Modular addition/subtraction	11
2.1.1.2 Montgomery Modular Multiplication (MMM)	12
2.1.2 $GF(2^n)$ arithmetic	13
2.1.3 Modular Exponentiation/Inversion	15
2.2 Public-Key Cryptography (PKC) algorithms	15

2.2.1	Rivest-Shamir-Adleman (RSA) cryptosystem	15
2.2.1.1	RSA-CRT algorithm	16
2.2.2	Elliptic Curve Cryptography (ECC)	16
2.2.2.1	Elliptic Curves over $GF(p)$	17
2.2.2.2	Elliptic Curves over $GF(2^n)$	18
2.2.2.3	Point Multiplication	19
2.3	Data representation systems	20
2.3.1	Residue Number System (RNS)	20
2.3.2	Polynomial Residue Number System (PRNS)	22
2.4	Summary	23
3	RNS application in Elliptic Curve Cryptography	25
3.1	Introduction	26
3.2	Combining RNS and ECC	27
3.2.1	Extended RNS	27
3.2.2	Embedding RNS in Elliptic Curve Arithmetic	27
3.2.3	Graph-Oriented Optimization Of Point Addition / Doubling Algorithms	29
3.2.3.1	Comments on the graph-oriented optimization	30
3.3	Hardware Implementation	32
3.3.1	Modular addition/subtraction	32
3.3.2	Modular multiplication	32
3.3.3	The Elliptic Curve Point Multiplier	32
3.3.4	The RNS-to-binary converter	34
3.3.5	Projective-to-affine coordinates conversion	37
3.4	Performance Results and Comparisons	39
3.4.1	Impact of the number of moduli and their word-lengths on the performance	41
3.5	Pipelined RNS structures	43
3.5.1	Modular multiplication in RNS	43
3.5.2	Design	44
3.5.2.1	Modular adders and multipliers	44
3.5.2.2	Conversion from base \mathcal{B} to base \mathcal{A}	47
3.5.2.3	Conversion from base \mathcal{A} to base \mathcal{B}	47
3.5.3	Hardware Architecture for RNS Montgomery multiplication	49
3.6	Implementation details of ECPM and comparisons	49
3.7	Summary	50
4	New RNS architectures for $GF(p)$ and $GF(2^n)$	53
4.1	Overview of RNS Montgomery modular multiplication	54
4.1.1	Base Conversion (BC) by Kawamura et al.	55
4.1.2	Base Conversion (BC) by Bajard et al.	56
4.1.3	Base Conversion (BC) by Gandino et al.	57
4.1.3.1	Modular reduction by the RNS moduli	59

4.1.3.2	Conversions to/from RNS	59
4.1.4	Architectural comparisons	60
4.2	New MRC-based Montgomery modular multiplication in $GF(p)$	62
4.2.1	The Proposed RNSMMM Architecture	62
4.2.2	Performance and Comparisons	64
4.2.2.1	Memory requirements	64
4.2.2.2	Frequency	64
4.2.2.3	Area requirements	65
4.3	New CRT-based Montgomery modular multiplication in $GF(2^n)$	65
4.3.1	The proposed PRNS Montgomery modular multiplication	66
4.3.2	Base Conversion (BC) algorithm for PRNSMMM	67
4.3.2.1	Proof of PRNSMMM's algorithm validity	67
4.3.3	The proposed PRNSMMM architecture	68
4.3.3.1	Polynomial-to-PRNS conversion	69
4.3.3.2	PRNS-to-Polynomial conversion	69
4.3.4	Performance	70
4.3.4.1	Memory requirements	70
4.3.4.2	Frequency	70
4.3.4.3	Area requirements	71
4.4	Summary	71
5	Novel versatile architectures	73
5.1	Decomposition of operations	74
5.1.1	Optimizing RNSMMM	74
5.1.2	Embedding PRNS in $GF(2^n)$ Montgomery Multiplication	75
5.1.3	The Proposed Versatile Architectures	76
5.1.4	Input-Output (IO) Conversions	78
5.1.4.1	Binary-to-Residue conversion	78
5.1.4.2	Residue-to-Binary Conversion	79
5.2	Versatile architectures - hardware design	79
5.2.1	Dual-Field Addition/Subtraction	79
5.2.1.1	Dual-Field Modular/Normal Addition/Subtraction	80
5.2.2	Dual-Field Multiplication	81
5.2.3	Dual-Field Modular Reduction	81
5.2.4	MAC Unit	83
5.2.4.1	Binary-to-residue conversion	83
5.2.4.2	Montgomery multiplication	84
5.2.4.3	Residue-to-binary conversion	85
5.3	Performance results	86
5.3.1	Area and Delay Estimations	86
5.3.1.1	Number of clock cycles	87
5.3.1.2	Memory Requirements	88
5.3.2	Comparisons with RNS implementations	88

5.3.3	Complexity comparisons with non-RNS implementations	89
5.3.4	Area-time-power comparisons	93
5.4	Summary	94
6	Novel RNS algorithms for modular multiplication	95
6.1	New RNS modular multiplication algorithm based on Barrett's technique . .	96
6.1.1	Barrett Modular Multiplication	96
6.1.2	Proposed RNS Barrett Modular Multiplication (RNSBMM) algorithm	96
6.1.3	Scaling and rounding of an RNS number	100
6.1.3.1	Divisibility check of an RNS number by 2^n	101
6.1.4	Numerical examples	104
6.2	Complexity analysis - comparisons	105
6.2.1	Complexity Comparisons	105
6.2.2	Architectural Study	107
6.2.2.1	Modular reduction by the RNS moduli	107
6.2.2.2	Conversions to/from RNS	107
6.2.2.3	Architectural comparisons	108
6.3	Summary	110
7	Cryptanalysis	113
7.1	Overview of side-channel attacks countermeasures	114
7.2	Fault handling in RNS-based multipliers	115
7.2.1	Hardware-fault tolerance in MRC-based RNS Montgomery multipliers	115
7.2.2	Hardware-fault tolerance in CRT-based RNS Montgomery multipliers	117
7.2.3	Remarks on Performance	118
7.3	Summary	118
8	Conclusions and Outlook	121
	Bibliography	125
	Curriculum Vitae	137



List of Figures

Chapter 1

1.1	Cryptographic applications	5
1.2	Thesis organization	6

Chapter 2

2.1	Modular adder/subtractor circuit	12
2.2	Operations on elliptic curves	19
2.3	General architecture of an RNS processor	22

Chapter 3

3.1	The DFG for the point addition algorithm	30
3.2	The DFG for the point doubling algorithm	31
3.3	The modular multiplier	33
3.4	General architecture of the RNS computing structures	33
3.5	General architecture of the RNS ECPM	35
3.6	Large multiplication paradigm	36
3.7	Architecture of the large operand multiplier	36
3.8	Architecture of the RNS-to-binary converter	37
3.9	The projective-to-affine converter	38
3.10	Number and word-length of moduli vs. (a) speed and (b) area	42
3.11	Impact of the RNS-to-binary converter on the area of ECPM	42
3.12	(a) Modulo m adder/subtractor [SFM ⁺ 09], (b) Proposed modulo $2^k - 2^{t_i} - 1$ multiplier, (c) Reduction circuit	46
3.13	(a) Proposed reconfigurable modular (RM) adder, (b) Proposed RM Multiplier, ($F = k, k - 1, k + 1$)	46

3.14	Calculation of H in RNS to MRS conversion for the first base (a) area efficient design, (b) Fast design	47
3.15	RNSMMM architecture, (a) Fast design, (b) Area efficient design	48

Chapter 4

4.1	MAC cell [GLP ⁺ 12]	62
4.2	The proposed MRC-based RNSMMM architecture	63
4.3	The proposed PRNSMMM architecture	69

Chapter 5

5.1	Dual-field full-adder cell	80
5.2	Dual-field CLA	80
5.3	Dual-field modular/normal adder/subtractor (DMAS)	81
5.4	Dual-field multiplier (DM)	82
5.5	Dual-field modular reduction unit (DMR)	82
5.6	Task distribution in the proposed DRAMMM	83
5.7	The proposed MAC unit	85
5.8	The proposed DRAMMM architecture	85
5.9	Normalized time complexity function $f(r, L)$	86
5.10	Normalized area complexity function $g(r, L)$	87
5.11	Area \times time product function $\sigma(r, L)$	88

Chapter 6

6.1	Region plot for inequality (6.7)	99
6.2	Scaling by two scheme [MBS03]	101
6.3	The proposed Scaling-Rounding (SR) scheme for RNSBMM	102
6.4	The proposed offset evaluation block	103
6.5	Multiply-accumulate cell architecture [GLP ⁺ 12]	109



List of Tables

Chapter 1

1.1 Information security objectives	4
---	---

Chapter 3

3.1 Comparison of ECPM architectures	40
3.2 The RNS base modulus set for the 192-bit implementation	41
3.3 Proposed RNS Bases	45
3.4 Comparison of ECPM architectures	51

Chapter 4

4.1 Number of modular multiplications in state-of-the-art RNSMMM	59
4.2 Number of multiplication steps per RNS modular multiplication in state-of-the-art RNSMMM ([GLP ⁺ 12] without BC correction)	61
4.3 Area and delay comparisons with $L = 33, r = 32, \epsilon = 3, M = 1, h = 11$	61
4.4 Basic logic library in CMOS technology (model from [Gaj97])	61
4.5 ROM requirements of the proposed RNSMMM architecture	64
4.6 Number of operations in RNSMMM algorithms	64
4.7 ROM requirements of the proposed PRNSMMM architecture	70
4.8 Number of modular multiplications in PRNSMMM algorithm	70

Chapter 5

5.1 Normalized area and delay of the proposed DRAMMM architecture	84
5.2 Parameters of the proposed DRAMMM stored in ROM	88
5.3 Number of modular multiplications in the DRAMMM algorithm	89
5.4 Normalized time and area complexity comparisons in $GF(p)$	90

5.5	Normalized area-time complexity comparisons for a 1024-bit $GF(p)$ Montgomery multiplication (CPA delays included)	91
5.6	Area-time comparisons for 1024-bit modular exponentiation	92
5.7	Normalized area and delay of standard cells	93

Chapter 6

6.1	Number of modular multiplications in state-of-the-art RNSMMM	106
6.2	Number of modular multiplications in the proposed RNSBMM	106
6.3	Number of multiplication steps per RNS modular multiplication in state-of-the-art RNSMMM ([GLP ⁺ 12] without BC correction)	107
6.4	Number of multiplication steps in the proposed RNSBMM	108
6.5	Area and delay comparisons with $k = 33$, $r = 32$, $\epsilon = 3$, $M = 1$, $h = 11$	110
6.6	Basic logic library in CMOS technology (model from [Gaj97])	110



Acronyms

ASIC	Application Specific Integrated Circuit
BC	Base Conversion
BMM	Barrett Modular Multiplication
CLA	Carry Lookahead Adder
CLB	Configurable Logic Blocks
CLG	Carry Lookahead Generator
CMOS	Complementary Metal-Oxide Semi-conductor
CPA	Carry Propagation Adder
CRT	Chinese Remainder Theorem
CSA	Carry Save Adder
DBC	Dual-field Base Conversion
DES	Data Encryption Sandard
DFA	Dual-field Full Adder
DFG	Data Flow Graph
DLP	Discrete Logarithm Problem
DM	Dual-field modular Multiplier
DMAS	Dual-field Modular Adder-Subtractor

DMR	Dual-field Modular Reduction
DRAMMM	Dual-field Residue Arithmetic Montgomery Modular Multiplication
DSA	Digital Signature Algorithm
DSP	Digital Signal Processing
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECPM	Elliptic Curve Point Multiplier
FA	Full Adder
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GCD	Greatest Common Divisor
GF	Galois Field
IO	Input-Output
ISE	Integrated Software Environment
LSB	Least Significant Bit
LUT	Look-Up Table
MAC	Multiply Accumulate
MLE	Montgomery Ladder Exponentiation
MM	Modular Multiplication
MMM	Montgomery Modular Multiplication
MRC	Mixed Radix Conversion
MRS	Mixed Radix System
MSB	Most Significant Bit
NAF	Non-adjacent form
NIST	National Institute of Standards Technology

PE	Processing Element
PKC	Public-Key Cryptography
PRA	Polynomial Residue Arithmetic
PRNS	Polynomial Residue Number System
PRNSMMM	PRNS Montgomery Modular Multiplication
RAM	Random Access Memory
RNS	Residue Number System
RNSBMM	RNS Barrett Modular Multiplication
RNSMMM	RNS Montgomery Modular Multiplication
ROM	Read Only Memory
RSA	Rivest-Shamir-Adleman
RSA-CRT	RSA-Chinese Remainder Theorem
SD	Signed-Digit
SPA	Simple Power Analysis
SR	Scaling-Rounding
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration
VPN	Virtual Private Network



Notation

Generic notation

\mathbb{Z}	the set of integers
\mathbb{R}	the set of reals
$GF(2^n)$	Galois Fields of characteristic 2
$GF(p)$	Galois Fields of primes, p a prime
\mathcal{E}	an elliptic curve \mathcal{E}
\mathcal{O}	point at infinity of an elliptic curve \mathcal{E}
\mathcal{A}	An RNS/PRNS base \mathcal{A}
$a_{\mathcal{A}}$	A quantity a (integer or polynomial) expressed in an RNS/PRNS base \mathcal{A}
$\langle a \rangle_p$	$a \bmod p$ operation
$x y$	x divides y
$x \nmid y$	x does not divide y
$\varphi(\bullet)$	Euler's totient function
$\lfloor \bullet \rfloor$	floor function
$\lceil \bullet \rceil$	ceiling function
\ll	left shift
$SR(a) = \lfloor \frac{a}{2^n} \rfloor$	Scaling-Rounding of an n -bit integer a
\tilde{a}	a value a affected by an error

CHAPTER

1

Introduction

1.1 Overview

Cryptography is not a new field in electrical engineering, mathematics, and computer science academic and industrial R&D communities. Its history stretches from its initial and limited use by the Egyptians 4000 years ago, to the twentieth century when it defined to a great extent the outcome of World War II. The main practitioners of cryptography were, historically, those associated with the military, diplomatic services or the government in general, and it was the main practice to protect national secrets and political or war-field strategies.

The bloom of computers and communication systems in the 1960s brought new requirements and demands, especially from the private sector, in order to store and protect data in digital format and to provide secure communication services. A pioneer at that age was the work of Feistel at IBM in the early 1970s and the adoption of the U.S. Federal Information Processing Standard for encrypting unclassified information, known as Data Encryption Standard (DES).

Later on, in 1976, Diffie and Hellman published a paper entitled “*New Directions in Cryptography*”, which introduced the revolutionary concept of Public-Key Cryptography (PKC) as well as a new and ingenious method for key-exchange, the security of which is based on the difficulty of solving the underlying Discrete Logarithm Problem (DLP). Although no practical realization of a public-key encryption scheme was provided, the idea was so revolutionary and its implications and advantages were so profound that it attracted the interest of the research community.

In 1978, Rivest, Shamir, and Adleman proposed probably the most widely deployed PKC system nowadays, known as RSA (from the initials of their last names) [RSA78]. RSA is also based on a hard-to-solve underlying mathematical problem, i.e, the intractability of factoring large integers. In general, the concept of a hard mathematical problem as the basis of a cryptographic system encouraged researchers to strive for more efficient methods to factorize large integers. During the 1980s another class of powerful and practical public-key schemes was proposed by ElGamal, also based on the discrete logarithm problem [Elg85]. An important advance at the same period was also the introduction of elliptic curves in cryptography and the associated Elliptic Curve Cryptography (ECC) systems, proposed independently by V. Miller [Mil86] and N. Koblitz in 1985 [Kob87].

Without doubt, one of the most significant contributions of public-key cryptography is the concept of digital signatures. In 1991 the first international standard for digital signatures (ISO/IEC 9796) was adopted. It is based on the RSA public-key scheme. In 1994 the U.S. Government adopted the Digital Signature Standard, a mechanism based on the ElGamal public-key scheme [MVO96].

Development of new public-key schemes, like the exotic solutions based on quantum physics [BCJL93, DFSS05] or lattice theory [GGH97, Reg06], improvements to existing cryptographic solutions, and formal mathematical proofs of security and cryptanalytic strength of cryptographic algorithms continue at a rapid pace. Various standards and large-scale configurations are

constantly released by the industry to cover the wide range of applications employing cryptography. These activities, indicative of a living and exciting science field, have marked cryptography as the key-player in the endeavor to address the security needs of an information intensive society.

1.2 Design challenges and motivation

Equally important with the development of new cryptographic algorithms and their associated cryptanalytic properties are their hardware and software realizations for the corresponding platforms they run on. There are significant challenges in high-speed, small-space (circuit size or number of code lines) implementations of cryptographic algorithms. Some of these challenges, particularly speed and space issues, have been understood and partially met as soon as public-key cryptography algorithms were invented. However, new challenges appeared as systems equipped with cryptography were deployed for commercial use.

For example, the timing and power attack scenarios [Des09, HVM04] made us realize that a cryptographic algorithm implemented in software or hardware is something totally different than its mathematical description. While it may be almost impossible to break a cryptographic algorithm in an acceptable time, since it requires computing resources that are far beyond our current algorithmic and resource capabilities, it may be quite easy to obtain the very same key practically, by simply observing the timing or power data trace from a device performing a signature or decryption operation.

Another important challenge in crypto-hardware design is the increase of key word-lengths, a consequence of the higher security standards posed constantly by modern applications. As a result, the associated crypto-hardware complexity, both in terms of silicon area as well as from a power consumption point of view, is also expected to increase. This inevitably creates the necessity to reduce algorithmic and hardware complexity, usually by the aid of computer arithmetic techniques [Des09, EL04].

Moreover, the wide range of applications requiring secure implementations generated the need for a smooth integration frame of various cryptographic components operating in different platforms. A typical example of the cryptographic application span, depicted in Figure 1.1, highlights this need. Various underlying mathematical problems at the basis of the pyramid generate numerous cryptographic algorithms and protocols and, as a result, a handful of applications with different implementation requirements and operational platforms are employed in modern communication systems.

At the same time, the prerequisites that modern cryptographic systems need to comply with, shown in Table 1.1, pose an extra design factor that also requires design flexibility to achieve interoperability and communication closure. These necessities, driven by technological advances in all fields of communications and computer science, generated the challenge to implement as many functions as possible in one architecture. In that way, flexible systems that could adapt in different computing environments could be realized. This

Table 1.1: Information security objectives

confidentiality	keeping information secret from all but those who are authorized to see it
data integrity	ensuring information has not been altered by unauthorized or unknown means
entity authentication	corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.)
message authentication	corroborating the source of information; also known as data origin authentication
signature	a means to bind information to an entity
authorization	conveyance, to another entity, of official sanction to do or be something
validation	a means to provide timeliness of authorization to use or manipulate information or resources
access control	restricting access to resources to privileged entities
certification	endorsement of information by a trusted entity
timestamping	recording the time of creation or existence of information
witnessing	verifying the creation or existence of information by an entity other than the creator
receipt	acknowledgment that information has been received
confirmation	acknowledgment that services have been provided
ownership	a means to provide an entity with the legal right to use or transfer a resource to others
anonymity	concealing the identity of an entity involved in some process
non-repudiation	preventing the denial of previous commitments or actions
revocation	retraction of certification or authorization

would have a profound affect on the applicability and flexibility of modern cryptographic systems, allowing multiple algorithmic support. Such architectures would also reduce the hardware requirements for implementing these functions considering that, dedicated systems are now required to implement the various counterparts of a cryptographic system.

Under this perspective, this thesis attempts to approach the problem of hardware design in modern cryptography in a holistic manner, meeting, in the best possible extent, the requirements for area, speed and power performance, flexibility characteristics and built-in cryptanalytic properties and security features.

1.3 Thesis overview and contributions

The main contributions of this work can be divided in seven parts, following the organization of the thesis in chapters.

In Chapter 2 a necessary introduction to the mathematical concepts required throughout

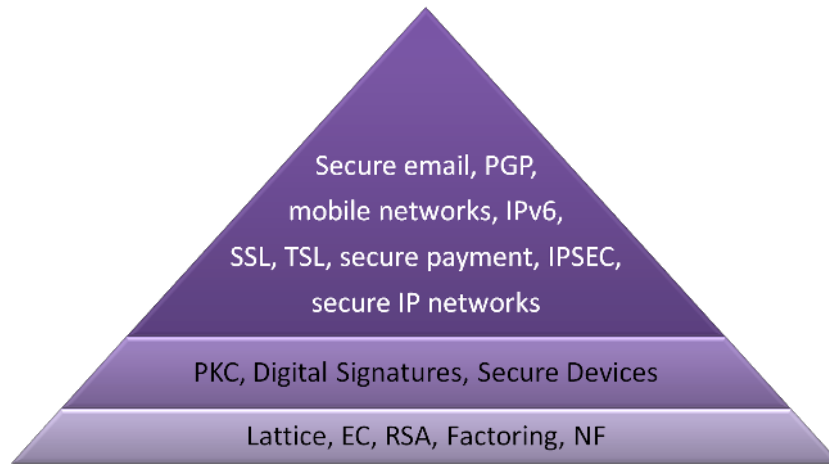


Figure 1.1: Cryptographic applications

this thesis is presented, including basics on finite field theory, public-key cryptography algorithms and data representation systems.

Chapter 3 presents, to the best of our knowledge, the first introduction of RNS arithmetic in ECC. Two RNS designs of an Elliptic Curve Point Multiplier (ECPM) are offered. Both designs are based on a novel Data Flow Graph (DFG) approach for the optimization of point addition and doubling operations. Through the proposed DFG approach not only the number of execution steps for a point addition is reduced, but also the same number of execution steps for both point operations is achieved, thus offering resistance against Simple Power Analysis (SPA) attacks for free.

For the first design an appropriate RNS range was selected to accommodate the full range of calculations without intermediate modular reductions, while the second design employed the RNS Montgomery Modular Multiplication (RNSMMM) algorithm. For the first architecture, extra care to the design for the output RNS-to-binary converter was given. A specially designed bit-serial multiplier was developed to handle large operands. The multiplier was then embedded in the architecture of the converter, forming a serial design suitable for large RNS ranges.

Area and timing results are offered proving the efficiency of the proposed implementation even towards dedicated Application Specific Integrated Circuit (ASIC) implementations. A study for various key lengths, number of RNS moduli and modulus word-lengths is also performed. It is proved that, in comparison to traditional arithmetic approaches, RNS has the tendency to perform better as the key length of an ECC system tends to increase.

The second design improved significantly our first effort by reducing the number of moduli channels required and by utilizing moduli of special form. This amounted to reduction of area and speed-ups in terms of total execution time for one point multiplication.

The rest chapters emphasize on the proposed versatile architectures and the accompanying

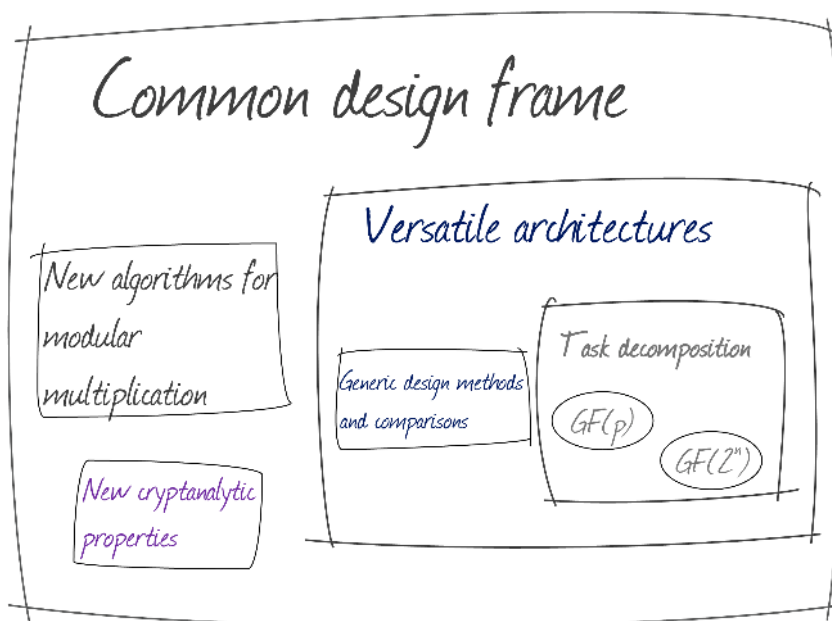


Figure 1.2: Thesis organization

methodologies and algorithms that were developed. A relation diagram of the various ideas developed in this thesis is shown in Figure 1.2.

In Chapter 4 an overview of state-of-the-art RNS Montgomery multiplication algorithms is presented, along with algorithmic and architectural comparisons. Following, new algorithms for modular multiplication that combine Montgomery multiplication and RNS-PRNS for $GF(p)$ and $GF(2^n)$ arithmetic are proposed, depicted as the inner boxes of Figure 1.2 under the title “*Task decomposition*”.

Especially for $GF(2^n)$, a methodology for incorporating Polynomial Residue Arithmetic (PRA) in the Montgomery multiplication algorithm for polynomials in $GF(2^n)$ is presented. The mathematical conditions that need to be satisfied, in order for this incorporation to be valid are also examined.

The ideas developed in Chapter 4 formed the basis of the proposed versatile architectures presented in Chapter 5, hence included in the box under the title “*Versatile Architectures*”. The mathematical framework and a flexible, dual-field, residue arithmetic architecture for Montgomery multiplication in $GF(p)$ and $GF(2^n)$ is developed and the necessary conditions for the system parameters (number of moduli channels, modulus word-length) are derived. The proposed architecture supports all operations of Montgomery multiplication in $GF(p)$ and $GF(2^n)$, residue-to-binary and binary-to-residue conversions, Mixed Radix Conversion (MRC) for integers and polynomials, dual-field modular exponentiation and inversion, in the same hardware.

An important derivative of this work is the development of a generic, technology-independent methodology to evaluate the optimal system parameters (number of moduli, mod-

ulus word-length). Generic complexity and real performance comparisons with state-of-the-art works prove the potential of residue arithmetic exploitation in Montgomery multiplication.

Chapter 6 presents, to the best of our knowledge, the first RNS modular multiplication algorithm based on Barrett's technique. The algorithm's validity is mathematically proved and the conditions to employ the proposed algorithm in the context of modular exponentiation are derived. Conditions for selecting the number and word-length of the RNS moduli are also provided. In the context of the proposed algorithm, methods to evaluate floor function and scaling by 2^n of an RNS number directly in RNS format are also proposed. Algorithmic and architectural comparisons with state-of-the-art algorithms based on Montgomery's technique prove the efficiency of the proposed algorithm in terms of total execution time. The idea of merging both types of algorithms (RNS Barrett Modular Multiplication (RNSBMM) and RNSMMM) into a common architecture is also considered. The chapter corresponds to the box entitled "*New algorithms for modular multiplication*".

In Chapter 7 an important property of RNS Montgomery multipliers in the context of the RSA-CRT crypto-algorithm is revealed. It is proved that the use of RNS multipliers offers resilience against hardware-fault attacks for free, with no need to modify in any way the original RSA-CRT algorithm, as opposed to the majority of current countermeasures. In this way, speedups offered by RSA-CRT in comparison to the original RSA algorithm are preserved.

Apparently, all chapters share common ideas, circuitry and methodologies, thus they are included in the larger box of Figure 1.2 under the general title "*Common Design Frame*". New ideas, derived from this thesis, for future research work on the field of cryptographic hardware design are offered in Chapter 8.

Mathematical Background

“We must be clear about the fact that the mathematical model cannot be used to prove anything about the real world, although a study of the model may help us discover important facts about the real world. A model is not true or false; rather, a model fits (i.e. corresponds properly to) or does not fit the real-life situation. A model is useful, or it is not.”

Paul E. Pfeiffer,
Concepts of Probability Theory,
McGraw-Hill, 1965

This chapter outlines the necessary mathematical concepts required in this doctoral thesis. The first sections present useful definitions from algebraic group theory and later, based on these definitions, we describe the main cryptographic algorithms implemented in this thesis. The last section is dedicated to alternative number representation systems, namely RNS and PRNS, which constitute the basis of the proposed versatile architectures.

2.1 Basics on finite-field theory

Let us refer to some general definitions useful for our discussion.

Definition 1. A group $\{G, \bullet\}$ is defined by a set of elements equipped with an operation \bullet whose result belongs also in the group G (closure property).

Assuming a, b elements of a group G , then the group should obey in the following laws:

- associative law: $(a \bullet b) \bullet c = a \bullet (b \bullet c)$
- has an identity element e such as: $e \bullet a = a \bullet e = a$
- its elements have inverses a^{-1} such as: $a \bullet a^{-1} = e$

Definition 2. If a group $\{G, \bullet\}$ is also commutative, i.e., $a \bullet b = b \bullet a$, then it forms an abelian group.

Definition 3. Let $\{G, \times\}$ be a group equipped with the operation of multiplication (multiplicative group). Assume $g \in G$ and $d \in \mathbb{Z}$. Then we define the operation of exponentiation as

$$g^d = \overbrace{g \times g \times \cdots \times g}^{d \text{ times}}. \quad (2.1)$$

If G is an abelian group then the group operation is addition, i.e., the group is $\{G, +\}$, and exponentiation is defined as

$$dg = \overbrace{g + g + \cdots + g}^{d \text{ times}}. \quad (2.2)$$

Definition 4. A group $\{G, \bullet\}$ is cyclic if every one of its elements, b_k , is some power of a certain group element g , i.e., $b_k = g^k$ and the identity element e is defined as $e = g^0$. The element g is called the generator of the group since it generates the group through repeated application of the operator on it.

Definition 5. A ring $\{R, +, \times\}$ is defined by a set of numbers, equipped with two operations of addition and multiplication for which it forms an abelian group for addition and multiplication has the properties of closure, associativity, and distributivity over addition, i.e., $a \times (b + c) = a \times b + a \times c$. If multiplication operation is also commutative, $\{R, +, \times\}$ forms a commutative ring. If multiplication operation has an identity element and no zero divisors, $\{R, +, \times\}$ forms an integral domain.

Definition 6. A field $\{F, +, \times\}$ is defined by a set of numbers, equipped with two operations of addition and multiplication for which it forms an abelian group for addition an abelian group for multiplication (ignoring 0), is a commutative ring and has multiplicative inverses for all non-zero elements, i.e., $a \times a^{-1} = e$.

The previous definitions imply that we can compute freely with $+$, $-$, \times , $/$ without leaving the set. In cryptography, finite fields (or Galois Field (GF)) are employed both for efficient implementations and for security reasons [Des09, BSS02, DBS06]. Details on the fields employed in this dissertation are given in the following section.

2.1.1 $GF(p)$ arithmetic

Field elements in $GF(p)$ are all integers in $[0, p - 1]$ and arithmetic is performed modulo p , where p a prime. We divide our discussion based on the most significant types of operations required in cryptography.

2.1.1.1 Modular addition/subtraction

Modular addition and subtraction are identical operations. The core idea is that the input operands are added or subtracted and the modulus p is subtracted or added to the previous results for a modular addition or subtraction respectively. Based on the output carries we select the appropriate result as the final result. The operations are summarized by Algorithms 2.1 and 2.2 below. In fact, with trivial modifications, a common modular adder/subtractor can be mechanized as shown in Figure 2.1.

Algorithm 2.1 Modular addition

Input: $p, 0 \leq x < p, 0 \leq y < p$

Output: $z = \langle x + y \rangle_p$

```

1  $z' = x + y$ 
2  $z'' = z' - p$ 
3 if  $z'' < 0$  then
4    $z = z'$ 
5 else
6    $z = z''$ 
7 end if
8 return  $z$ 
```

Algorithm 2.2 Modular subtraction

Input: $p, 0 \leq x < p, 0 \leq y < p$

Output: $z = \langle x - y \rangle_p$

```

1  $z' = x - y$ 
2  $z'' = z' + p$ 
3 if  $z' < 0$  then
4    $z = z''$ 
5 else
6    $z = z'$ 
7 end if
8 return  $z$ 
```

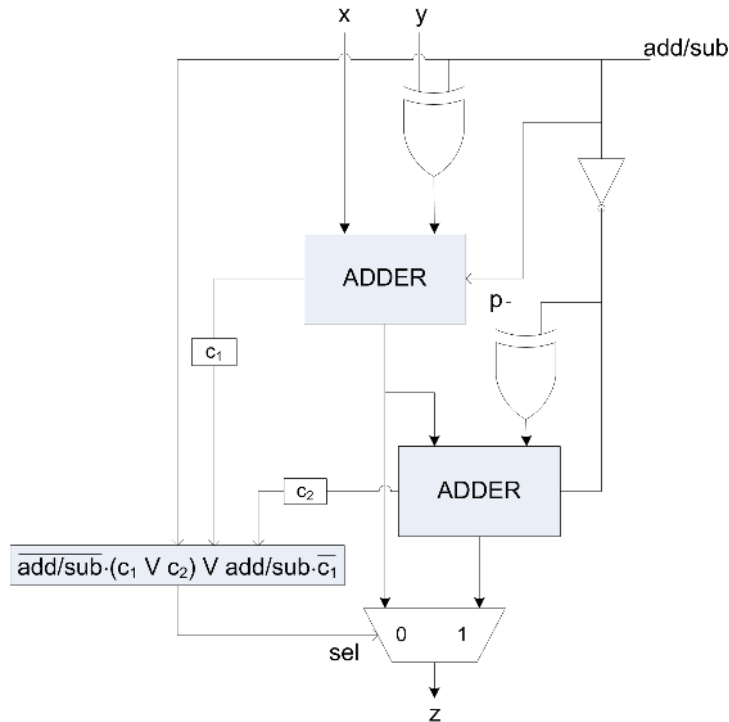


Figure 2.1: Modular adder/subtractor circuit

2.1.1.2 Montgomery Modular Multiplication (MMM)

Efficient field multiplication with large operands is crucial for achieving a satisfying system performance, since multiplication is the most time- and area-consuming operation. Cryptographic applications form a special case, since, for security reasons, they require large integer operands [Des09, LN86]. Various modular multiplication methods have been proposed in the literature including Montgomery, Barrett, Karatsuba-Offman algorithms etc [Mon85, Bar87, Des09, DBS06]. Details on Barrett’s method, along with a new algorithm for modular multiplication based on Barrett’s technique are provided in detail in Chapter 6.

Montgomery’s algorithm for modular multiplication without division [Mon85] is widely used today since it is well-suited to applications requiring consecutive multiplications, like in cryptography, computer algebra, digital signal processing, etc. On the other hand, the algorithm has undertaken huge analysis and numerous designs have been proposed, making the space for further improvements and development even narrower. We discuss the implications and new solutions offered by RNS and PRNS application to Montgomery multiplication in Chapters 3, 4, 5.

Montgomery’s algorithm is presented below, as Algorithm 2.3, in five steps, where R is the Montgomery radix, $\gcd(R, p) = 1$, $p < R$, and p is the reduction modulus. R must be chosen so that steps 2 and 5 are efficiently computed. It is usually chosen to be a power of 2, when radix-2 representation is employed. Condition $\gcd(R, p) = 1$ ensures the existence of p^{-1}

Algorithm 2.3 Montgomery Modular Multiplication MMM**Input:** $a, b, p, R, R^{-1} \{ a, b < p \}$ **Output:** $c \equiv abR^{-1} \pmod{p}, \{ c < 2p \}$

- 1 $s \leftarrow a \cdot b$
- 2 $t \leftarrow s \cdot (-p^{-1}) \pmod{R}$
- 3 $u \leftarrow t \cdot p$
- 4 $v \leftarrow s + u$
- 5 $c \leftarrow v/R$

\pmod{R} . Condition $p < R$ is sufficient for $c < 2p$ since

$$c = \frac{xy + tp}{R} < \frac{p^2 + pR}{R} = \left(\frac{p}{R} + 1\right)p < 2p. \quad (2.3)$$

Since $cR = ab + tp$, $cR \equiv ab \pmod{p}$ holds. By multiplying $R^{-1} \pmod{p}$ on both sides of (2.3), $c \equiv abR^{-1} \pmod{p}$ is obtained. Since Montgomery's method was originally devised to avoid divisions, it is expected to be well-suited to RNS implementations, considering that RNS division is inefficient to perform.

The algorithm requires first to transform the input operands to their corresponding Montgomery representations [Mon85]. Assuming an integer a , its Montgomery representation is defined as $\bar{a} = aR \pmod{p}$. This conversion may be realized by means of an extra Montgomery multiplication by $R^2 \pmod{p}$, i.e. $\bar{a} = a \times (R^2 \pmod{p}) \times R^{-1} \pmod{p} = aR \pmod{p}$. With these inputs the algorithm outputs the Montgomery residue of the result, i.e., $\bar{c} = cR \pmod{p} = abR \pmod{p}$.

An extra Montgomery multiplication needs to be executed to convert the Montgomery residue back to the integer domain representation. This iteration accepts as input the result $\bar{c} = cR \pmod{p}$ of the Montgomery multiplication and $1 \pmod{p}$ to produce $cR \times 1 \times R^{-1} \pmod{p} = c \pmod{p}$.

2.1.2 $GF(2^n)$ arithmetic

Field elements in $GF(2^n)$ are polynomials represented as binary vectors of dimension n , relative to a given polynomial basis $(1, \alpha, \alpha^2, \dots, \alpha^{n-1})$, where α is a root of an irreducible polynomial p of degree n over $GF(2)$. The field is then realized as $GF(2)[x]/(p)$ and the arithmetic is that of polynomials of degree at most $n - 1$, modulo p [BSS02].

The addition of two polynomials a and b in $GF(2^n)$ is performed by adding the polynomials, with their coefficients added in $GF(2)$, i.e., modulo 2. This is equivalent to a bit-wise XOR operation on the vectors a and b .

Important progress has been reported lately regarding $GF(2^n)$ multiplication. The Massey-Omura algorithm [OM86], the introduction of optimal normal bases [MOVW88] and their software and hardware implementations [MOVW88, AMV93], the Montgomery algorithm for multiplication in $GF(2^n)$ [KA98] are, among others, important advances. However, the

architectures proposed for the Massey-Omura algorithm, although compact and fast, are inflexible and expensive, while the Montgomery algorithm for $GF(2^n)$ multiplication proposed in [KA98] is targeted to software implementations.

The product of two elements a and b in $GF(2^n)$ is obtained by computing

$$c = a \cdot b \pmod{p}, \quad (2.4)$$

where c is a polynomial of degree at most $n - 1$ and $c \in GF(2^n)$.

A Montgomery multiplication algorithm suitable for polynomials in $GF(2^n)$ has been proposed [KA98]. Instead of computing the product $c = a \cdot b \pmod{p}$, the algorithm computes $c = a \cdot b \cdot R^{-1} \pmod{p}$, with $\deg\{c(x)\} < n$ and R is a special fixed element in $GF(2^n)$. The selection of $R(x) = x^n$ is the most appropriate, since modular reduction and division by x^n are simple shifts [Des09]. The algorithm is identical to Algorithm 2.3, except from the constant $-p^{-1}$ in step 2, which is p^{-1} in $GF(2^n)$. Moreover, in the integer case the output may exceed the modulus p , thus a final subtraction step is required. This is not necessary in polynomials, as it has been proven that the degree of the resulting polynomial c is less than n [KA98].

The Montgomery multiplication method in $GF(2^n)$ also requires that R and p are relatively prime, i.e., $\gcd\{R, p\} = 1$. This assumption always holds, since p is an irreducible polynomial in $GF(2)$, thus it is not divisible by x . Since R and p are relatively prime, there exist two polynomials R^{-1} and p^{-1} such that

$$R \cdot R^{-1} + p \cdot p^{-1} = 1, \quad (2.5)$$

where R^{-1} is the inverse of R modulo p . The polynomials R^{-1} and p^{-1} can be computed using the extended Euclidean algorithm [Des09, LN86, McE87]. The Montgomery multiplication of a and b is then defined as

$$c = a \cdot b \cdot R^{-1} \pmod{p}, \quad (2.6)$$

which can be computed according to Algorithm 2.4.

Algorithm 2.4 Montgomery Modular Multiplication in $GF(2^n)$

Input: a, b, R, p, p^{-1} $\{\deg\{a, b\} < n\}$

Output: $c = a \cdot b \cdot R^{-1} \pmod{p}$ $\{\deg\{c\} < n\}$

- 1 $s \leftarrow a \cdot b$
 - 2 $t \leftarrow s \cdot p^{-1} \pmod{R}$
 - 3 $u \leftarrow t \cdot p$
 - 4 $v \leftarrow s + u$
 - 5 $c \leftarrow v/R$
-

2.1.3 Modular Exponentiation/Inversion

Modular exponentiation, as will be shown in next sections, is a key-operation in PKC. It's mechanized through consecutive modular multiplications using any of the algorithms we described before for modular multiplication. A naive approach is through the binary expansion of the exponent. Assume the task of $b = z^e \pmod p$, such as $e = \sum_{i=0}^{l-1} e_i 2^i$. There are two possibilities for implementation; the first is a method starting from the Most Significant Bit (MSB) and working downwards (called left-to-right method) and the second starting from the Least Significant Bit (LSB) and working upwards (right-to-left). An example for left-to-right method using the MMM algorithm appears in Algorithm 2.5.

Algorithm 2.5 Left-to-right modular exponentiation

Input: $z, e = (e_{n-1} \dots e_1 e_0)_2$

Output: $b, b \equiv \langle z^e \rangle_p$

```

1  $b \leftarrow 1$ 
2 for  $i = n - 1, \dots, 0$  do
3    $b \leftarrow MMM(b, b)$ 
4   if  $e_i = 1$  then
5      $b \leftarrow MMM(b, z)$ 
6   end if
7 end for
8 return  $b$ 

```

According to Euler's theorem, if a is co-prime to p , i.e., $\gcd(a, p) = 1$ then

$$a^{\varphi(p)} \equiv 1 \pmod p, \quad (2.7)$$

where φ is the Euler's totient function. Therefore the modular inverse can be directly computed as

$$a^{\varphi(p)-1} \equiv a^{-1} \pmod p. \quad (2.8)$$

In the special case where p is a prime, then $\varphi(p) = p - 1$ and consequently the modular inverse can be computed using modular exponentiation according to

$$a^{-1} \equiv a^{p-2} \pmod p. \quad (2.9)$$

2.2 Public-Key Cryptography (PKC) algorithms

2.2.1 RSA cryptosystem

RSA is an algorithm for public-key cryptography that is based on the presumable difficult mathematical problem of factoring large integers. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks,

an English mathematician, had developed an equivalent system in 1973, but it was not classified until 1997.

In the RSA cryptosystem the public and private keys are generated by two distinct prime numbers p and q . We calculate the public modulus $N = pq$ and the quantity $\varphi(N) = (p - 1)(q - 1)$. We choose $e \in \mathbb{Z}$ co-prime to $\varphi(N)$ and we compute $d = e^{-1} \pmod{\varphi(N)}$. The public key is the pair (N, e) and the private key is d . The primes p, q are also kept secret. The public and private keys are referred as the public and secret exponent respectively. The encryption of a message M is defined by

$$C = M^e \pmod{N} \quad (2.10)$$

and decryption by

$$M = C^d \pmod{N}. \quad (2.11)$$

2.2.1.1 RSA-CRT algorithm

The security of RSA depends on the key size. With large keys varying from 1,024-bit, appropriate for protecting data through the year 2015, to 2,048-bit, appropriate through the year 2035 [Kal], it is apparent that efficient arithmetic operations on large operands are crucial for optimal RSA implementations.

A solution towards this direction was the introduction of the Chinese Remainder Theorem (CRT) to the RSA protocol, namely the RSA-CRT [Lab11b, Lab11a]. In RSA-CRT, the digital signature operation $S = M^d \pmod{N}$ is split in two operations $S_p = M^{d_p} \pmod{p}$ and $S_q = M^{d_q} \pmod{q}$, where $d_p = d \pmod{p - 1}$ and $d_q = d \pmod{q - 1}$. CRT ensures that the combination of these two values produces the signature S as

$$S = S_q + [(S_p - S_q) \cdot (q^{-1} \pmod{p}) \pmod{p}] \cdot q, \quad (2.12)$$

denoted from now on as $S = CRT(S_p, S_q)$ [Knu97]. In this way, an approximate 4-time speedup of operations is achieved [Lab11b, Lab11a].

2.2.2 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC), presented by N. Koblitz [Kob87] and V. Miller [Mil86] independently in 1985, has withstood a large number of attacks and has evolved significantly, so that it is considered nowadays a mature public-key cryptosystem. Extensive research work regarding the underlying mathematics, security, and its efficient implementations, is being carried out.

ECC offers the highest strength per bit and the smallest key size, when compared to other public-key cryptosystems, by exploiting the mathematical basis of ECC, i.e., the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECDLP states that given two points P, Q on an elliptic curve such that $Q = [k]P$, it is computationally infeasible to calculate $[k]$ [BSS02].

Although elliptic curves can be defined on a variety of different fields, only finite fields are employed in cryptography. Among them, prime fields $GF(p)$ and binary extension fields

$GF(2^n)$ are considered to be the ones that offer the most efficient and secure implementations [BSS02].

2.2.2.1 Elliptic Curves over $GF(p)$

An elliptic curve \mathcal{E} over $GF(p)$ is defined by an equation of the form

$$y^2 = x^3 + ax + b, \quad (2.13)$$

where $a, b \in GF(p)$ and $4a^3 + 27b^2 \neq 0 \pmod{p}$, together with a special point \mathcal{O} , called the *point at infinity*. The set $\mathcal{E}(GF(p))$ consists of all points (x, y) , $x, y \in GF(p)$, that satisfy (2.13), together with \mathcal{O} . Addition of two points on an elliptic curve can be defined by the group law. Together with this addition operation, the set of points $\mathcal{E}(GF(p))$ forms a group, with \mathcal{O} serving as its identity element. It is this group that is used in the construction of elliptic curve cryptosystems. The special case of adding a point to itself is called a point doubling.

Examples of point addition and point doubling are depicted in Figure 2.2. The double of a point P_0 is obtained by taking the tangent line on P_0 until a second intersection point on the curve is found (there is always a second point due to the form of (2.13)). The mirror point of this second intersection on the x -axis is $2P_0$. Similarly, to add two points P_0, P_1 , a third intersecting point is found by the line that connects P_0, P_1 . The mirror point on x -axis of the third intersection point is $P_2 = P_0 + P_1$.

For the case of $GF(p)$ let $P_0 = (x_0, y_0), P_1 = (x_1, y_1) \neq \mathcal{O}$ and $P_0 \neq -P_1$. The sum $P_2(x_2, y_2) = P_0 + P_1$ is given by

$$P_2 = P_0 + P_1 = \begin{cases} x_2 = \lambda^2 - x_0 - x_1 \\ y_2 = (x_0 - x_2)\lambda - y_0, \end{cases} \quad (2.14)$$

where $\lambda = \frac{y_1 - y_0}{x_1 - x_0}$. The double of a point is given by

$$P_2 = 2P_0 = \begin{cases} x_2 = \lambda^2 - 2x_0 \\ y_2 = (x_0 - x_2)\lambda - y_0, \end{cases} \quad (2.15)$$

where $\lambda = \frac{3x_0^2 + a}{2y_0}$.

From (2.14), (2.15) it is apparent that in order to perform an addition or a doubling of a point in *affine* representation one needs to compute the inverse of an element in $GF(p)$, which is a time consuming operation in $GF(p)$ [BSS02]. In order to avoid inversions, the use of *projective coordinates* of the EC points has been proposed [BSS02]. Given a point $P = (x, y)$ in affine coordinates, the projective coordinates $P = (X, Y, Z)$ are given by

$$X = x; Y = y; Z = 1. \quad (2.16)$$

There are various projective coordinate representations that lead to more efficient implementations than using the one in (2.16). Jacobian coordinates are an example of such a

representation, and will be employed in the implementations proposed in this thesis. Using Jacobian coordinates, the affine representation of an EC point is given by

$$x = \frac{X}{Z^2}; y = \frac{Y}{Z^3}. \quad (2.17)$$

while the point at infinity is given by $\mathcal{O} = (0, 0, 1)$.

Using the representation in (2.17), (2.13) rewrites to

$$\mathcal{E}(GF(p)): Y^2 = X^3 + aXZ^4 + bZ^6. \quad (2.18)$$

Let $P_0 = (X_0, Y_0, Z_0)$, $P_1 = (X_1, Y_1, Z_1) \in \mathcal{E}(GF(p))$. The sum $P_2 = (X_2, Y_2, Z_2) = P_0 + P_1 \in \mathcal{E}(GF(p))$ can be computed as follows.

If $P_0 = P_1$ then

$$P_2 = 2P_1 = \begin{cases} X_2 = M^2 - 2S \\ Y_2 = M(S - X_2) - T, \\ Z_2 = 2Y_1Z_1 \end{cases} \quad (2.19)$$

where $M = 3X_1^2 + aZ_1^4$, $S = 4X_1Y_1^2$ and $T = 8Y_1^4$. On the other hand, if $P_0 \neq P_1$, then

$$P_2 = P_0 + P_1 = \begin{cases} X_2 = R^2 - TW^2 \\ 2Y_2 = VR - MW^3, \\ Z_2 = Z_0Z_1W \end{cases} \quad (2.20)$$

where $R = Y_0Z_1^3 - Y_1Z_0^3$, $T = X_0Z_1^2 + X_1Z_0^2$, $W = X_0Z_1^2 - X_1Z_0^2$, $M = Y_0Z_1^3 + Y_1Z_0^3$, and $V = TW^2 - 2X_2$.

2.2.2.2 Elliptic Curves over $GF(2^n)$

Similar to the case of $GF(p)$, an elliptic curve \mathcal{E} over $GF(2^n)$ is defined by an equation of the form

$$y^2 + xy = x^3 + ax^2 + b \quad (2.21)$$

with $a, b \in GF(2^n)$ and $b \neq 0$. The corresponding equations for point doubling and point addition in affine coordinates are

$$P_2 = P_0 + P_1 = \begin{cases} x_2 = \lambda^2 + \lambda + x_0 + x_1 + a \\ y_2 = (x_0 - x_2)\lambda - y_0, \end{cases} \quad (2.22)$$

where $\lambda = \frac{y_0 + y_1}{x_0 + x_1}$ and

$$P_2 = 2P_0 = \begin{cases} x_2 = \lambda^2 - 2x_0 \\ y_2 = (x_0 - x_2)\lambda - y_0, \end{cases} \quad (2.23)$$

where $\lambda = \frac{3x_0^2 + a}{2y_0}$.

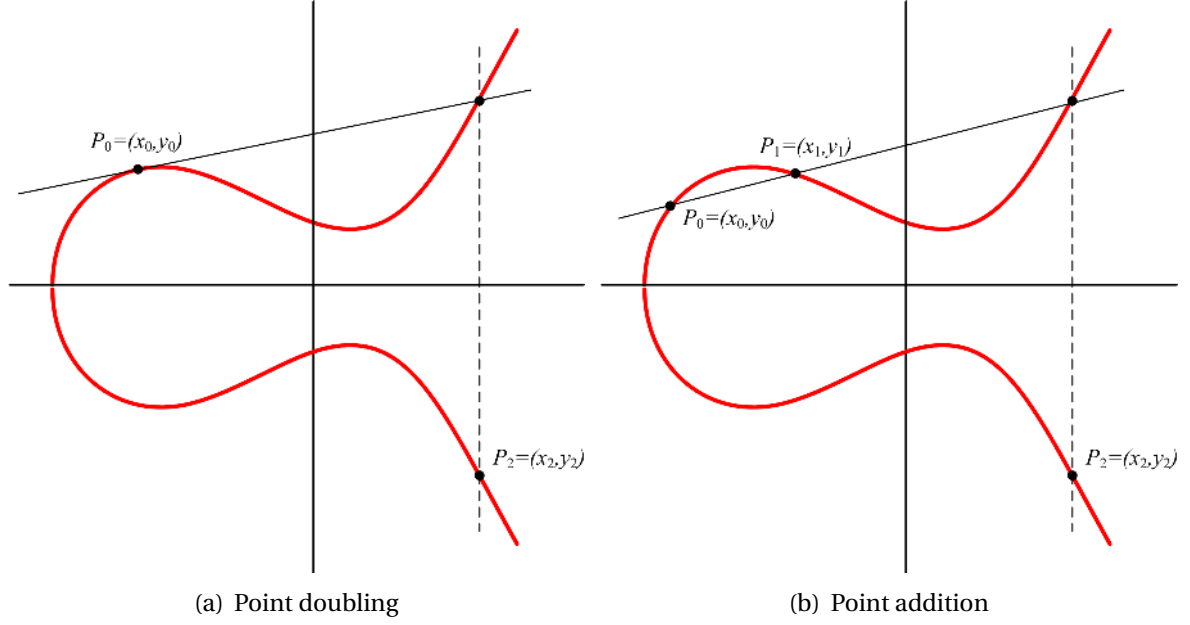


Figure 2.2: Operations on elliptic curves

As in the case of $GF(p)$, using the Jacobian representation for the coordinates, the equation for the curve rewrites to

$$\mathcal{E}(GF(2^n)): Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6 \quad (2.24)$$

and the equations for point doubling and point addition rewrite to

$$P_2 = 2P_1 = \begin{cases} X_2 = (X_1 - MZ_1^2)^4 \\ Y_2 = X_1^4 Z_3 + SX_3, \\ Z_2 = X_1 Z_1^2 \end{cases} \quad (2.25)$$

where $M = b^{2^{n-2}}$, $S = Z_3 + X_1^2 + Y_1 Z_1$ and

$$P_2 = P_0 + P_1 = \begin{cases} X_2 = aZ_3^2 + \lambda_6 \lambda_9 + \lambda_3^3 \\ Y_2 = \lambda_9 X_3 + \lambda_8 \lambda_7^2, \\ Z_2 = \lambda_7 Z_1 \end{cases} \quad (2.26)$$

where $\lambda_1 = X_0 Z_1^2$, $\lambda_2 = X_1 Z_0^2$, $\lambda_3 = \lambda_1 + \lambda_2$, $\lambda_4 = Y_0 Z_1^3$, $\lambda_5 = Y_1 Z_0^3$, $\lambda_6 = \lambda_4 + \lambda_5$, $\lambda_7 = Z_0 \lambda_3$, $\lambda_8 = \lambda_6 X_1 + \lambda_7 Y_1$, $\lambda_9 = \lambda_6 + Z_3$.

2.2.2.3 Point Multiplication

With the operations of point doubling and point addition available, the next step is to implement the scalar point multiplication, which is the most important operation in ECC. For the purposes of this thesis, the *binary method algorithm* was chosen, because it is easy to

Algorithm 2.6 Binary method for EC point multiplication

Input: A point P , an l -bit integer $k = \sum_0^{l-1} k_j 2^j$

Output: $Q = [k]P$

```

1  $Q \leftarrow \mathcal{O}$ 
2 for  $j = l - 1$  to 0 do
3    $Q \leftarrow [2]Q$ 
4   if  $k_j = 1$  then
5      $Q \leftarrow Q + P$ 
6   end if
7 end for
8 return  $Q$ 

```

implement and minimizes memory requirements. The binary method algorithm [BSS02] is based on the binary expansion of k , as follows.

The binary method requires $l - 1$ point doublings and $W - 1$ point additions, where l is the length and W the Hamming weight of the binary expansion of k . For any positive integer k , the notation $[k]$ is used to denote the *multiplication-by- k* map from the curve to itself. The notation $[k]$ is extended to $k \leq 0$ by defining $[0]P = \mathcal{O}$, and $[-k]P = -([k]P)$. Other methods based on various representations for the scalar $[k]$ include window-based algorithms, Signed-Digit (SD) representations, Non-adjacent form (NAF) representations etc, which are out of scope of this thesis to analyze further [BSS02].

2.3 Data representation systems

2.3.1 Residue Number System (RNS)

RNS is a number system that allows representing a number as a set of smaller numbers. RNS was originally described in terms of a game by Nicomachus of Gerasa (100 CE) in his book “*Introduction to Arithmetic*”. Later, the problem was re-described by Sun Tsu Suan-Ching (Master Sun’s Arithmetic Manual) in a 4th century CE book.

RNS consists of a set of L , pair-wise relatively prime integers $\mathcal{A} = (m_1, m_2, \dots, m_L)$ (called the *base*) and the range of the RNS is computed as $A = \prod_{i=1}^L m_i$. Any integer $z \in [0, A - 1]$ has a unique RNS representation $z_{\mathcal{A}}$ given by $z_{\mathcal{A}} = (z_1, z_2, \dots, z_L) = (\langle z \rangle_{m_1}, \langle z \rangle_{m_2}, \dots, \langle z \rangle_{m_L})$, where $\langle z \rangle_{m_i}$ denotes the operation $z \bmod m_i$. Assuming two integers a, b in RNS format, i.e., $a_{\mathcal{A}} = (a_1, a_2, \dots, a_L)$ and $b_{\mathcal{A}} = (b_1, b_2, \dots, b_L)$, then one can perform the operations $\otimes \in (+, -, *)$ in parallel by

$$a_{\mathcal{A}} \otimes b_{\mathcal{A}} = (\langle a_1 \otimes b_1 \rangle_{m_1}, \langle a_2 \otimes b_2 \rangle_{m_2}, \dots, \langle a_L \otimes b_L \rangle_{m_L}). \quad (2.27)$$

Equation (2.27) highlights the benefits of RNS; all operations are confined within each independent modulus channel and there is no need for carry propagation among channels of different moduli.

To reconstruct the integer from its residues, two methods may be employed [Tay88]. The first is through the CRT according to

$$z = \left\langle \sum_{i=1}^L \langle z_i \cdot A_i^{-1} \rangle_{m_i} \cdot A_i \right\rangle_A, \quad (2.28)$$

where $A_i = A/m_i$ and A_i^{-1} is the inverse of A_i modulo m_i . Note that (2.28) implies that in order to obtain the exact value of z we must compute

$$z = \sum_{i=1}^L \langle z_i \cdot A_i^{-1} \rangle_{m_i} \cdot A_i - \gamma A, \quad (2.29)$$

where γ is an integer correction factor. In practical implementations, (2.29) is preferred since it avoids the large $\text{mod } A$ reduction of (2.28) [KKSS00, BDK01, GLP⁺12, GLMB11].

The second method is through the MRC. The MRC of an integer z with an RNS representation $z_{\mathcal{A}} = (z_1, z_2, \dots, z_L)$ is given by

$$z = U_1 + W_2 U_2 + \dots + W_L U_L, \quad (2.30)$$

where $W_i = \prod_{j=2}^i m_{j-1}$, $\forall i \in [2, L]$ and the U_i s are computed according to

$$\begin{aligned} U_1 &= z_1 \\ U_2 &= \langle (z_2 - U_1) m_{1,2}^{-1} \rangle_{m_2} \\ U_3 &= \langle ((z_3 - U_1) m_{1,3}^{-1} - U_2) m_{2,3}^{-1} \rangle_{m_3} \\ &\vdots \\ U_L &= \langle (\dots (z_L - U_1) m_{1,L}^{-1} - \dots - U_{L-1}) m_{L-1,L}^{-1} \rangle_{m_L}, \end{aligned} \quad (2.31)$$

where $m_i m_{i,j}^{-1} \equiv 1 \pmod{m_j}$. The mixed-radix digits U_1, U_2, \dots, U_L are referred as the Mixed Radix System (MRS) representation of z . Equation (2.31) requires $L \frac{L-1}{2}$ modular multiplications. Another version of MRC that simplifies (2.31) and reduces the total number of modular multiplications to only $L - 2$ is based on

$$\begin{aligned} U_1 &= z_1 \\ U_2 &= \langle z_2 - z_1 \rangle_{m_2} \\ U_3 &= \langle z_3 - z_1 - W_2 U_2 \rangle_{m_3} \\ &\vdots \\ U_L &= \langle z_L - z_1 - W_2 U_2 - W_3 U_3 - \dots - W_{L-1} U_{L-1} \rangle_{m_L}, \end{aligned} \quad (2.32)$$

providing that the predetermined factors $V_1 \equiv 1$ and $V_i \equiv \left\langle \left(\prod_{j=1}^{i-1} m_j \right)^{-1} \right\rangle_{m_i}$, $\forall i \in [2, L]$ are all unity [YM91]. Of the three methods, the proposed architectures utilize the MRC of (2.32), as it avoids the problem of evaluating the correction factor γ of (2.29) and reduces the total complexity of the original MRC in terms of number of modular multiplications.

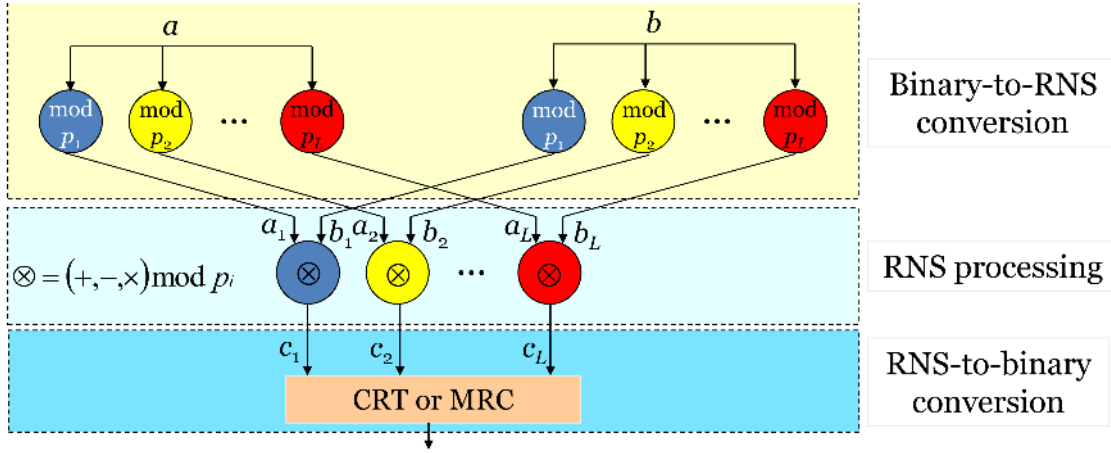


Figure 2.3: General architecture of an RNS processor

2.3.2 Polynomial Residue Number System (PRNS)

Similar to RNS, a PRNS is defined through a set of L , pair-wise relatively prime polynomials $\mathcal{A} = (m_1(x), m_2(x), \dots, m_L(x))$. We denote by $A(x) = \prod_{i=1}^L m_i(x)$ the dynamic range of the PRNS. In PRNS, every polynomial $z(x) \in GF(2^n)$, with $\deg\{z(x)\} < \deg\{A(x)\}$, has a unique PRNS representation:

$$z_{\mathcal{A}} = (z_1, z_2, \dots, z_L), \quad (2.33)$$

such as $z_i = z(x) \bmod m_i(x)$, $i \in [1, L]$, denoted as $\langle z \rangle_{m_i}$. In the rest of this thesis, the notation " (x) " to denote polynomials shall be omitted, for simplicity. The notation z will be used interchangeably to denote either an integer z or a polynomial $z(x)$, according to context.

Assuming the PRNS representation $a_{\mathcal{A}} = (a_1, a_2, \dots, a_L)$ and $b_{\mathcal{A}} = (b_1, b_2, \dots, b_L)$ of two polynomials $a, b \in GF(2^n)$, then all operations $\otimes \in (+, -, *)$ can be performed in parallel, as

$$a_{\mathcal{A}} \otimes b_{\mathcal{A}} = (\langle a_1 \otimes b_1 \rangle_{m_1}, \langle a_2 \otimes b_2 \rangle_{m_2}, \dots, \langle a_L \otimes b_L \rangle_{m_L}). \quad (2.34)$$

Conversion from PRNS to weighted polynomial representation is identical to the MRC for integers. The only difference is that, the subtractions in (2.32) are substituted by polynomial additions. In the case of CRT for polynomials, the conversion is based on

$$z(x) = \sum_{i=1}^L \langle z_i(x) \cdot A_i^{-1}(x) \rangle_{m_i(x)} \cdot A_i(x), \quad (2.35)$$

where $A_i(x) = A(x)/m_i(x)$ and $A_i^{-1}(x)$ is the inverse of $A_i(x)$ modulo $m_i(x)$. Unlike the integer case in (2.28), the final reduction by the product polynomial $A(x)$ is not necessary in the case of polynomials over $GF(2^n)$.

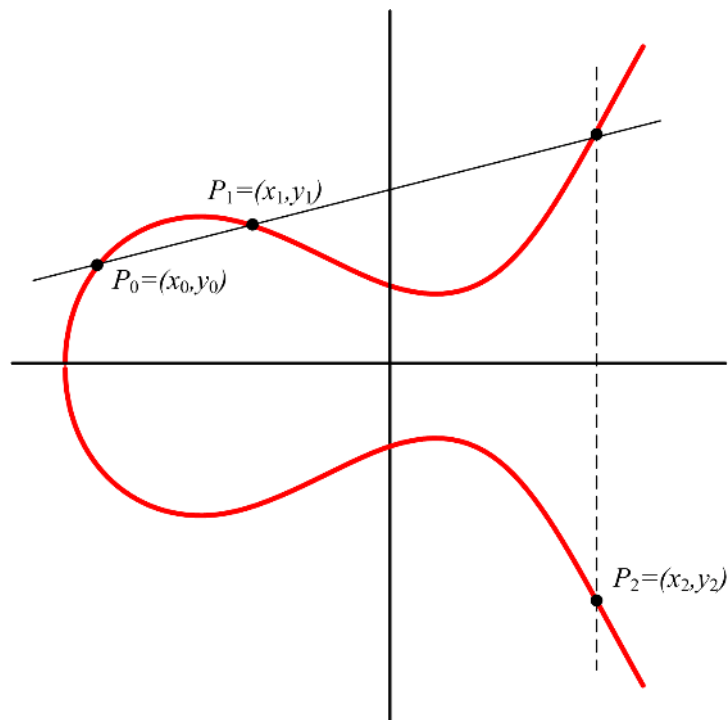
A general architecture of a system based on residue arithmetic appears in Figure 2.3. There, Input-Output (IO) converters for the binary-to-RNS and RNS-to-binary conversions are employed, which increase significantly the total computational and hardware complexity

of an RNS-based processor. Apparently, RNS is well-suited to applications requiring multiple executions of a core algorithm in the main RNS processing core, while IO operations are executed only once at the beginning and end of the the algorithm.

2.4 Summary

In this chapter, the mathematical tools and concepts necessary for this thesis were presented. The all-important concepts of group and field theory were elaborated and the most significant algorithms in PKC that will be implemented in this thesis were analyzed. Various other schemes like the Digital Signature Algorithm (DSA) scheme for digital signatures or the Diffie-Hellman key-exchange [DH76] also popular in PKC, require the same modular operations, however their analysis is out of scope of this thesis. The reader should be familiar by now with modular exponentiation which, as we have shown, involves consecutive modular multiplications and constitutes the core operation in PKC. Alternative representations using RNS and PRNS were also presented. The next chapters are dedicated to new applications and methods for modular multiplication using RNS and PRNS.

RNS application in Elliptic Curve Cryptography



This chapter presents, to the best of our knowledge, the first practical implementation of an elliptic curve processor using the RNS representation. We approach the problem by evaluating an appropriate range for the calculations, and new task execution graphs for point doubling and point addition are proposed. The tasks are optimized to be resistant against power and timing attacks. The idea is further enhanced and more efficient designs based on pipelined RNS structures and moduli of special form are also proposed.

3.1 Introduction

Elliptic Curve Cryptography (ECC), presented by N. Koblitz [Kob87] and V. Miller [Mil86] independently in 1985, has withstood a large number of attacks and has evolved significantly, so that it is considered a mature public-key cryptosystem. Extensive research work regarding the underlying mathematics, security, and its efficient implementations is being carried out.

ECC offers the highest strength per bit and the smallest key size, when compared to other public-key cryptosystems, by exploiting the mathematical basis of ECC, i.e., the discrete logarithm problem in the group of points over elliptic curves.

Although Elliptic Curves (EC) can be defined on a variety of different fields, only finite fields are employed for cryptography. Among them, prime fields $GF(p)$ and binary extension fields $GF(2^n)$ are considered to be the ones that offer the most efficient and secure implementations [BSS02].

The operands of ECC operations are large finite field elements. Implementing point multiplication algorithms in hardware leads to designs with high area complexity and high multiplication time delay. Therefore, there is a need for increasing the speed of ECC systems with the least possible area penalty. An obvious approach to achieve this would be through parallelization of their operations.

In recent years, RNS has enjoyed renewed scientific interest due to its ability to perform parallel and fast modular arithmetic. Apart from its traditional use in digital signal processing, RNS is also employed for the design of cryptographic systems [SFM⁺09, NMSK01, BI04, BDEM06, ESJ⁺13, SS14]. Using RNS, a given data range can be decomposed into parallel paths of smaller dynamic ranges, with no need for exchanging information between paths employing different moduli. As a result, the use of RNS can offer reduced complexity and power consumption of arithmetic units with large word lengths. On the other hand, RNS implementations bear the extra cost of an input converter to translate numbers from a standard binary format into residues and an output converter to implement the translation from RNS to a binary representation [Tay88].

The first practical deployment of RNS for the implementation of point multiplication over elliptic curves was proposed in [SFKS06, SKS06]. This implementation proved to be competitive towards existing designs in terms of speed, but the additional area overhead was significant.

The contribution of this work is an RNS architecture and detailed implementation of an ECPM, with the input and output converters included. The impact of various RNS bases, in terms of number of moduli and their bit lengths, on the area and speed of the proposed implementation is investigated, as this determines, to a large degree, the potential use of RNS in ECC. The results of this work consolidate the application of RNS in ECC, as the proposed implementation is highly effective. Furthermore, as key lengths are expected to grow in the coming years, RNS could be used to effectively counter balance the increasing computational complexity.

3.2 Combining RNS and ECC

In the following, the notation used in Chapter 2 for the RNS representation is followed. Let us rewrite some basic equations for easiness. RNS consists of a set of L , pair-wise relatively prime integers $\mathcal{A} = (m_1, m_2, \dots, m_L)$ (called the *base*) and the range of the RNS is computed as $A = \prod_{i=1}^L m_i$. Any integer $z \in [0, A - 1]$ has a unique RNS representation $z_{\mathcal{A}}$ given by $z_{\mathcal{A}} = (z_1, z_2, \dots, z_L) = (\langle z \rangle_{m_1}, \langle z \rangle_{m_2}, \dots, \langle z \rangle_{m_L})$, where $\langle z \rangle_{m_i}$ denotes the operation $z \bmod m_i$.

For the purposes of this chapter, converting an integer $z_{\mathcal{A}}$ in RNS format to its associated binary representation will be based on CRT according to

$$z = \sum_{i=1}^L \langle z_i \cdot A_i^{-1} \rangle_{m_i} \cdot A_i - \gamma A, \quad (3.1)$$

where γ is an integer correction factor [KKSS00, BDK01, GLP⁺12, GLMB11].

3.2.1 Extended RNS

An algorithm developed in [SK89] may be employed at this point to extend the use of RNS to include negative numbers as well. The algorithm requires a redundant modulus $m_r \geq L$ so that the RNS base \mathcal{A} is extended to $\mathcal{A} = (m_1, m_2, \dots, m_L \sim m_r)$. This redundant channel will be available from now on throughout the calculations of the ECPM. Let z be an integer with a RNS representation $z_{\mathcal{A}} = (z_1, z_2, \dots, z_L \sim z_r)$, where $z_r = \langle z \rangle_{m_r}$. By reducing both sides of (3.1) $\bmod m_r$ we obtain that

$$\begin{aligned} \langle z \rangle_{m_r} &= \left\langle \left\langle \sum_{i=1}^L \langle z_i \cdot A_i^{-1} \rangle_{m_i} \cdot A_i \right\rangle_{m_r} - \langle \gamma A \rangle_{m_r} \right\rangle_{m_r} \Rightarrow \\ \langle \gamma \rangle_{m_r} &= \left\langle \langle A^{-1} \rangle_{m_r} \left(\left\langle \sum_{i=1}^L \langle z_i \cdot A_i^{-1} \rangle_{m_i} \cdot A_i \right\rangle_{m_r} - \langle z \rangle_{m_r} \right) \right\rangle_{m_r} \\ &= \left\langle \langle A^{-1} \rangle_{m_r} (\delta - \langle z \rangle_{m_r}) \right\rangle_{m_r}, \end{aligned} \quad (3.2)$$

where $\delta = \left\langle \sum_{i=1}^L \langle z_i \cdot A_i^{-1} \rangle_{m_i} \cdot A_i \right\rangle_{m_r}$. Since $\gamma < L$ and $m_r \geq L$ it follows that $\gamma = \langle \gamma \rangle_{m_r}$ [SK89].

The representation $z_{\mathcal{A}} = (z_1, z_2, \dots, z_L \sim z_r)$ forms the *extended* RNS representation. It has been proven that any number $z \in [-A + 1, \dots, A - 1]$ represented by $(z_1, z_2, \dots, z_L \mid z_r)$ can be correctly calculated by formulas (3.1) and (3.2) [AH93]. The fact that negative numbers can be calculated correctly by equations (3.1) and (3.2) will later prove to be extremely useful in order to employ RNS for the ECC arithmetic.

3.2.2 Embedding RNS in Elliptic Curve Arithmetic

In the remainder of this chapter we will focus on elliptic curves defined over $GF(p)$, where p is a “large” prime number. The arithmetic is the usual modulo p arithmetic as defined in Section 2.1. Elliptic curve arithmetic is implemented according to Section 2.2.

Using Jacobian coordinates, the affine representation of an EC point is given by

$$x = \frac{X}{Z^2}; y = \frac{Y}{Z^3}, \quad (3.3)$$

while the point at infinity is given by $\mathcal{O} = (0, 0, 1)$ and the curve equation corresponds to

$$\mathcal{E}(GF(p)): Y^2 = X^3 + aXZ^4 + bZ^6. \quad (3.4)$$

In this case, EC point addition and doubling can be defined as follows. Let the points $P_0 = (X_0, Y_0, Z_0)$, $P_1 = (X_1, Y_1, Z_1) \in \mathcal{E}(GF(p))$. The sum $P_2 = (X_2, Y_2, Z_2) = P_0 + P_1 \in \mathcal{E}(GF(p))$ can be computed as follows.

If $P_0 = P_1$ then

$$P_2 = 2P_1 = \begin{cases} X_2 = M^2 - 2S \\ Y_2 = M(S - X_2) - T, \\ Z_2 = 2Y_1 Z_1 \end{cases} \quad (3.5)$$

where $M = 3X_1^2 + aZ_1^4$, $S = 4X_1 Y_1^2$ and $T = 8Y_1^4$. On the other hand, if $P_0 \neq P_1$, then

$$P_2 = P_0 + P_1 = \begin{cases} X_2 = R^2 - TW^2 \\ 2Y_2 = VR - MW^3, \\ Z_2 = Z_0 Z_1 W \end{cases} \quad (3.6)$$

where $R = Y_0 Z_1^3 - Y_1 Z_0^3$, $T = X_0 Z_1^2 + X_1 Z_0^2$, $W = X_0 Z_1^2 - X_1 Z_0^2$, $M = Y_0 Z_1^3 + Y_1 Z_0^3$, and $V = TW^2 - 2X_2$.

All operations in equations (3.5) and (3.6) are performed \pmod{p} , where p is the characteristic of the field. In a traditional implementation of a cryptographic scheme with an n -bit key, all operands and finite field circuitry are n -bit long. Instead of this, smaller circuits can be used operating in parallel, to generate the result. In the proposed approach, finite field circuits are replaced with RNS ones. The benefits of using smaller RNS operands are exploited to implement a fast and area efficient ECPM architecture.

For a valid replacement, an equivalent RNS dynamic range must be defined. Initially, it can be assumed that (3.5) and (3.6) are computed over the integers, i.e., without the \pmod{p} reduction. Let $|w|$ be the absolute maximum value generated by these computations. For a valid RNS implementation, the RNS dynamic range must be chosen so that $A \geq |w|$. Then, data are represented in RNS format and point multiplication can be performed using RNS circuits.

For example, if the field characteristic p is 192-bit long, then the equivalent RNS range can be calculated to be 840 bits, after simulating the algorithms of point doubling and consecutive point addition in Mathematica. In the presented implementation, an RNS basis set of 20 moduli, 42-bit each, was used for a 192-bit ECC. If a point multiplication result needs to be transformed back to a finite field element, we only need to implement (3.1) to get a valid result over the integers and then perform a final \pmod{p} reduction of the result to get the

finite field element. Due to the ability of the *extended* RNS to represent negative numbers, the results of the calculations in (3.5) and (3.6) over the integers can be correctly mapped to finite field representation.

3.2.3 Graph-Oriented Optimization Of Point Addition / Doubling Algorithms

Our previous efforts for designing an RNS ECPM resulted in a very fast design, but lacked in terms of area towards other existing implementations [SFKS06, SKS06]. This work formed the basis for further investigation of the sequence of operations, for the point addition / doubling algorithms, at an architectural abstraction level.

Many abstract models for representing the behavior of an algorithm at the architectural level have been proposed in the technical literature [Mic94]. All of these models consider the sequence of tasks in the algorithm and their dependencies in terms of availability of data, availability of hardware, and serialization constraints. The sequencing flow of the algorithm's behavior can be represented graphically using a Data Flow Graph (DFG) model.

A DFG is a directed graph $G_d(V, E)$, whose vertex set V is in one-to-one correspondence with the set of tasks, while the directed set E is in correspondence to the data transfer from one operation to the other. An assumption is made, that, at the end of a computation stage of each member of V , there is a temporal storage unit (e.g. a register) making its output available for consumption.

The ordering of the tasks represents the temporal dependencies in the sequencing graph, highlighting the dependencies between the tasks. Exploitation of the DFGs can be made with several techniques, i.e., balancing the graph (minimization of power dissipation) or shortening the longest path (performance increase). Further improvements can be suggested, including parallelization in operations, pre-computations, etc. In order to increase the throughput for the targeted application, the latter techniques can exploit efficiently the characteristics of the point addition/doubling algorithms.

For example, it is considered useful to explore every task for spatial and temporal dependencies between calculations. It is then easier to select either when to perform a calculation, or how many arithmetic circuits are required to implement the targeted algorithm. The DFGs in Figures 3.1 and 3.2 depict the result of the optimization process in the point addition/doubling algorithms. Each circle denotes an arithmetic operation in the RNS representation and each computation step is shown as a dashed line. The operands participating as inputs in this operation are at the top of each circle.

The registers used for storing the result of an operation are defined on the left side of each figure (3.1, 3.2) and they are denoted as A, B, C, D, E, F, G, H and I. Their initial contents are given at the top of each figure. The result is shown at the bottom of each task and inside the rectangles the outputs are obtained. Parallelization between multiplication and addition or subtraction was also achieved. We have considered a single adder/ subtractor circuit, to save in terms of area. One pre-computation step exists in each figure. During time step

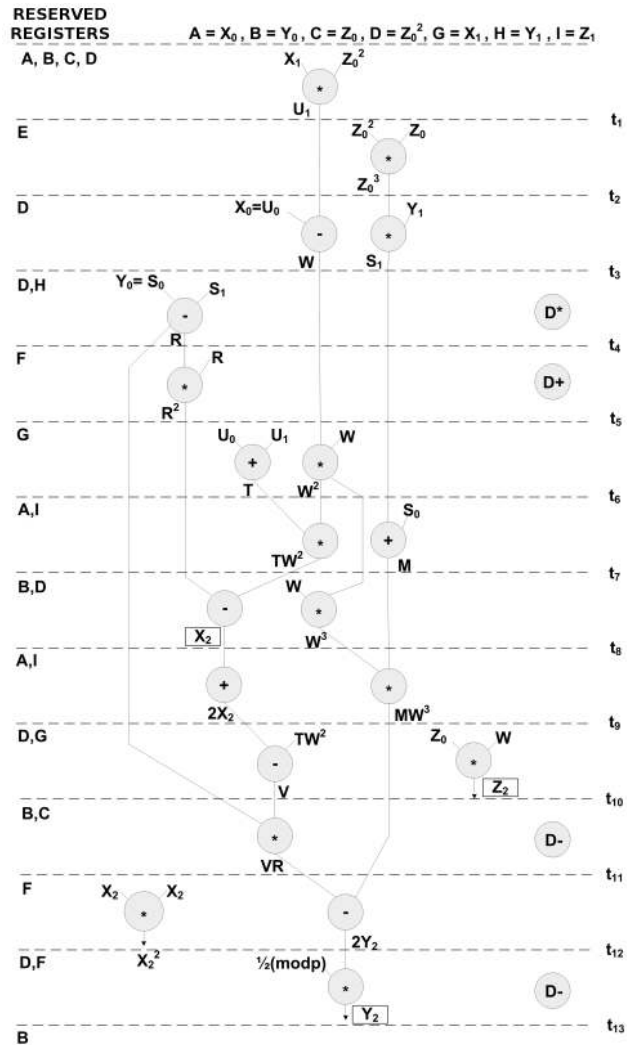


Figure 3.1: The DFG for the point addition algorithm

12 (Figure 3.1), an extra multiplication is performed that produces X_2^2 for use in the first step of the point doubling algorithm. Similarly, in time step 13 of the point doubling graph, an extra multiplication is performed to produce Z_2^2 for use in the following point addition. If the subsequent operation is not a point addition, instead of pre-computing Z_2^2 , X_2^2 is calculated for use in the first step of the following point doubling operation.

To increase the degree of parallelization of the proposed implementation, a special circuit could be used for squaring. Since the multiplication speed of the proposed design outperforms existing implementations, a squarer circuit is omitted for area reduction reasons.

3.2.3.1 Comments on the graph-oriented optimization

Since only nine registers compose the system's register file, large area savings are achieved, in comparison with previous works [SFKS06, SKS06], where, for each execution step, a sep-

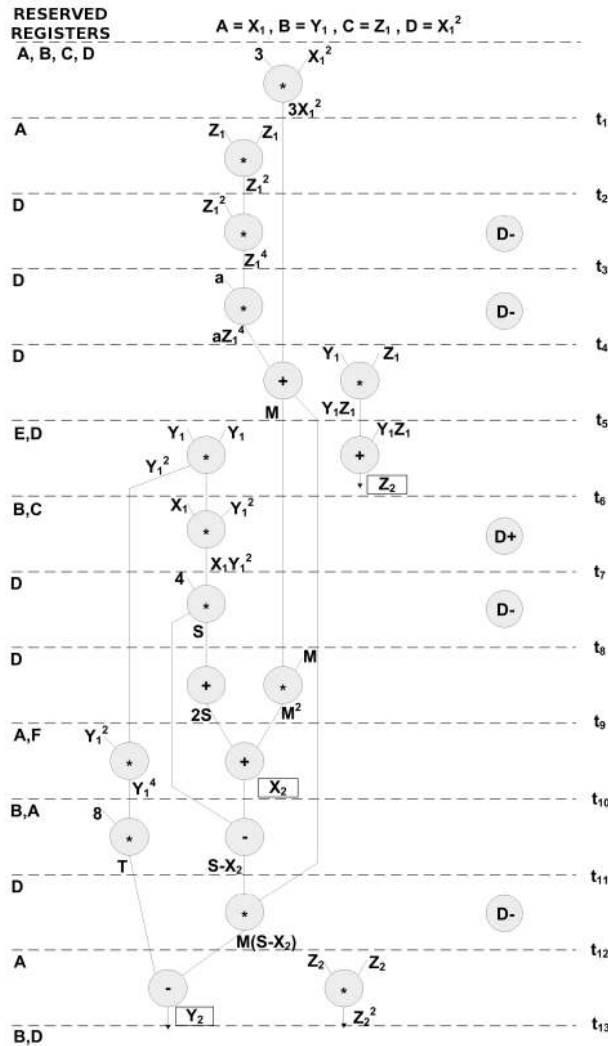


Figure 3.2: The DFG for the point doubling algorithm

arate register is used to store the result. Consequently, the circuit complexity for controlling the data flow of the system is also decreased. The above optimizations also result in a reduction of the multiplication operations in point addition/doubling algorithms. Only 13 execution steps are now needed both for a point addition and for a point doubling. As a result, savings in speed are obtained compared to [SFKS06, SKS06].

The total number of execution steps of Figures 3.1 and 3.2 for a doubling and a subsequent point addition is 26. This is also the case for a typical realization of (3.5) and (3.6) [BSS02]. In [LH08b], only 9 execution steps are required to perform a point doubling. This is expected since a separate circuit for squaring is utilized and multiplications by small constants are replaced with consecutive additions. In the proposed implementation, both squaring and multiplication by small constants are performed with the use of RNS multipliers. In [KF07], four parallelization levels are provided, i.e., with the use of one, two, three, or four modular multipliers working in parallel. In order for the comparison to be fair, the case of using one

multiplier is examined. In this case, the total number of execution steps is also 26. However, the proposed implementation outperforms these works, which also exploit a DFG approach to increase the parallelization degree of ECC operations (see Table 3.1).

An additional advantage of the proposed architecture is the equalization of execution steps and power consumption for both EC operations. This was achieved by inserting dummy operations in the DFGs' datapath. Dummy operations are denoted in Figures 3.1 and 3.2 as "D \otimes ", where $\otimes = +, -, *$ for a dummy addition, subtraction, or multiplication respectively.

Simple Power Analysis (SPA) attacks rely on the power traces of point addition and point doubling. Due to the dummy operations inserted, power consumption in each time step for both DFGs is equalized. Therefore, the proposed ECPM is protected against such threats without any extra cost in operations such as required by other designs [HMV04].

3.3 Hardware Implementation

3.3.1 Modular addition/subtraction

Modular addition and subtraction have already been described in Section 2.1 and the derived circuit is shown in Figure 2.1.

3.3.2 Modular multiplication

Modular multiplication was based on Horner's rule shown in (3.7), where r is the number of digits of X, Y and m an arbitrary r -bit modulus

$$\langle XY \rangle_m = \langle (\dots ((x_{r-1}Y)2 + x_{r-2}Y)2 + \dots)2 + x_0Y \rangle_m. \quad (3.7)$$

Algorithm 3.1 describes the realization of modular multiplication. It implies a bit-serial architecture for the modular multiplier, since our main optimization goal is the area reduction. A corresponding circuit is shown in Figure 3.3.

The RNS adder/subtractor and RNS multiplier are a parallel combination of the circuits in Figures 2.1 and 3.3, respectively. Their architecture is depicted in Figure 3.4. Each block represents a modular adder/subtractor or a modular multiplier. Each block executes the assigned modular operation (addition, subtraction, or multiplication) $\text{mod } m_i, 1 \leq i \leq L$, where the m_i 's define the modulus set of the RNS base.

3.3.3 The Elliptic Curve Point Multiplier

The system architecture for the proposed ECPM is presented in Figure 3.5. It consists of the two RNS operation units (RNS adder/subtractor, RNS multiplier), the system's register file, the input and output converters and a Finite State Machine (FSM) operating as a control unit. A selection module for driving the outputs of the registers to the appropriate RNS operation unit is also included.

Algorithm 3.1 Modular multiplication based on Horner's rule

Input: $0 \leq X, Y \in N$ and $r, m \in N^*$

Output: $Z[0] = \langle XY \rangle_m$

- 1 $Z[r] \leftarrow 0$
 - 2 **for** $i = 0$ to r **do**
 - 3 $Z[r - i] \leftarrow \langle 2Z[r - i + 1] + x_{r-i}Y \rangle_m$
 - 4 **end for**
 - 5 **return** Z
-

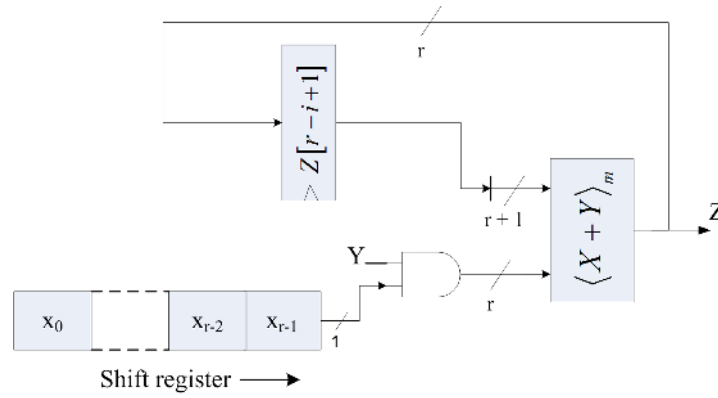


Figure 3.3: The modular multiplier

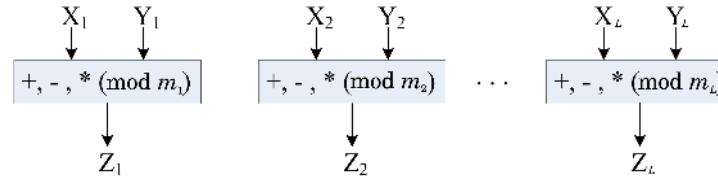


Figure 3.4: General architecture of the RNS computing structures

Initially, the projective coordinates (X, Y, Z) of point P enter serially into the binary-to-RNS converter. The converter consists of a modular multiplier which computes $\langle (X, Y, Z) * 1 \rangle_{m_i}$, $i = 1, \dots, L$, where the m_i 's are the RNS base moduli (see Figure 3.5). Following this conversion, a selection is made between the point at infinity \mathcal{O} and point P , used to perform the point multiplication, $[k]P$. Depending on whether the circuit is in initialization phase or not, the point \mathcal{O} or point P , accordingly, is stored in the G, H, and I registers.

All the registers of the register file are two input registers with *load* and *write enable* control signals. However, registers A to F accept inputs from both buses, i.e., the output of both RNS modules, but the G, H, I registers are connected directly to the input and the multiplication bus. The main reason for making this selection was to reuse the G, H and I registers for storing both the input data and intermediate results.

The FSM produces the appropriate signals to control the *load* and *write enable* signals of the register file, in order to store the result of an RNS operation. Afterward, the selection

module accepts these results and drives them to the input of the appropriate RNS module, in order to perform the next operation, according to the DFGs presented in Figures 3.1 and 3.2. When all the bits of k are processed, the result is driven to the RNS-to-binary converter to get the binary representation of the resulting point $[k]P$.

3.3.4 The RNS-to-binary converter

In RNS applications, the RNS-to-binary converter plays a crucial role in the performance of the overall system. In fact, in most of the cases, a well-designed and efficient converter is the criterion of whether an RNS application can be competitive or not towards non-RNS implementations. Carefully selected modulus sets or simply small RNS ranges can reduce the complexity of the converter, but in cases like ECC where the dynamic range of computations is very large, typical implementations can not be considered.

The following operations for the RNS-to-binary conversion have to be performed according to (3.1) and (3.2) (assume r to be the word length of each modulus and L to be the number of the moduli):

1. r -bit modular multiplications for the products $\langle A_i^{-1} z_i \rangle_{m_i}$.
2. Multiplication of the $r(L-1)$ -bit A_i constants with the r -bit products of the previous step to form the inner products $A_i \langle A_i^{-1} z_i \rangle_{m_i}$.
3. Addition of the L , rL -bit inner products of step 2, to form the term $\sum_{i=1}^L A_i \langle A_i^{-1} z_i \rangle_{m_i}$ of 3.1 and (3.2).
4. Assuming that L is q -bit long, then for (3.1) a $rL \times q$ bit multiplication has to be performed to form the term γA . This term is then subtracted from the outcome of step 3 to get the final result of (3.1).
5. In (3.2), a multiplication similar to step 2 is performed to form $A^{-1} \sum_{i=1}^L A_i \langle A_i^{-1} z_i \rangle_{m_i}$. Then, a modular subtraction between the previous term and the residue z_r is performed to form γ .

Due to the large operands of the multiplication in steps 2 and 5, a specially designed multiplier needs to be considered. Recall that the main operations shown in Figure 3.5, i.e., the binary-to-RNS conversion, point multiplication, and the RNS-to-binary conversion are pipelined. As a result, since point multiplication is slower than all other operations, a serial implementation of the converter and the large multiplier of steps 2 and 5 was realized. Such a design approach offers the best speed to area trade-off.

Assume for simplicity that X and Y are two variables rL -bit and r -bit long, respectively, and that we want to calculate their product $P = XY$. It holds that

$$P = XY = \sum_{i=0}^{rL-1} x_i 2^i Y. \quad (3.8)$$

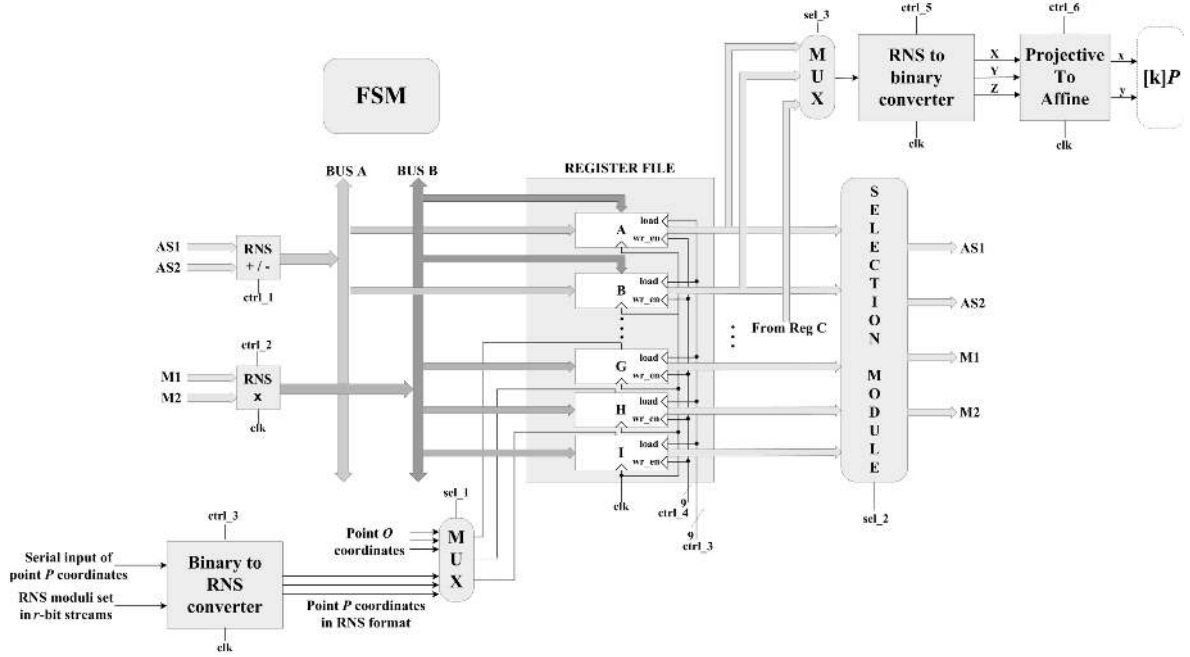


Figure 3.5: General architecture of the RNS ECPM

Since rL is divisible by r , (3.8) is decomposed as

$$\begin{aligned}
 P = XY &= \sum_{i=0}^{rL-1} x_i 2^i Y = \\
 &= \sum_{i=0}^{r-1} x_i 2^i Y + \sum_r^{2r-1} x_i 2^i Y + \dots + \sum_{(L-1)r}^{rL-1} x_i 2^i Y = \\
 &= \sum_{j=1}^L \left(\sum_{i=(j-1)r}^{jr-1} x_i 2^i Y \right). \tag{3.9}
 \end{aligned}$$

The “large” multiplication of (3.9) is decomposed into L , r -bit multiplications and r -bit additions. The validity of the above shall be illustrated with an example. Consider that we want to perform the multiplication of $101010110_2 \times 110_2$. In that case $r = 3$, $L = 3$ and $rL = 9$. We split the multiplication according to (3.9) into three 3-bit multiplications. The process is illustrated in Figure 3.6.

Beginning with the r LSBs of the multiplicand, r -bit streams of this operator are multiplied with the r -bit multiplier. The r LSBs of the first multiplication are driven directly to the output. We store the remaining r MSBs and we perform the next multiplication between the r -bit multiplier and the next r -bit stream of the large multiplicand. The MSBs of the previously stored result are added to the r -bit LSBs of the current result and the outcome is driven to the output.

The procedure is repeated until all L , r -bit streams of the multiplicand are processed (see Figure 3.7). Having designed the multiplication module, the realization of the RNS-to-binary converter is analyzed. The designing goal being the minimization of area, a serial architecture was adopted. The architecture of the converter is illustrated in Figure 3.8. The RNS digits z_i of the integer z are driven along with the constants A_i^{-1} into a modular multiplier to produce the inner products $\langle A_i^{-1} z_i \rangle_{m_i}$.

Afterward, each product is multiplied with the constants A_i and then they are added recursively to produce the term $\sum_{i=1}^L A_i \langle A_i^{-1} z_i \rangle_{m_i}$ of (3.1) and (3.2). For the case of γ , (3.2) is rewritten as:

$$\gamma = \left\langle -x_r K + K \sum_{i=1}^L \langle A_i \rangle_{m_r} \langle A_i^{-1} z_i \rangle_{m_i} \right\rangle_{m_r}, \quad (3.10)$$

where $K = \langle A^{-1} \rangle_{m_r}$. Thus, in parallel with the calculation of $\sum_{i=1}^L A_i \langle A_i^{-1} x_i \rangle_{m_i}$ shown before, we calculate γ by first calculating the product $\langle K \sum_{i=1}^L \langle A_i \rangle_{m_r} \langle A_i^{-1} z_i \rangle_{m_i} \rangle_{m_r}$ and then by adding those products to form $\langle K \sum_{i=1}^L \langle A_i \rangle_{m_r} \langle A_i^{-1} z_i \rangle_{m_i} \rangle_{m_r}$. A final modular subtraction by $x_r K$ produces γ . As soon as γ is calculated, another “large” multiplication is performed, i.e., the multiplication $-\gamma A$. The final stage is an addition of the summation term $\sum_{i=1}^L A_i \langle A_i^{-1} z_i \rangle_{m_i}$ with $-\gamma A$ to produce the binary representation of the integer z .

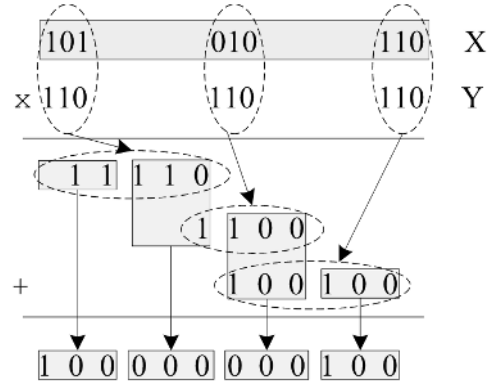


Figure 3.6: Large multiplication paradigm

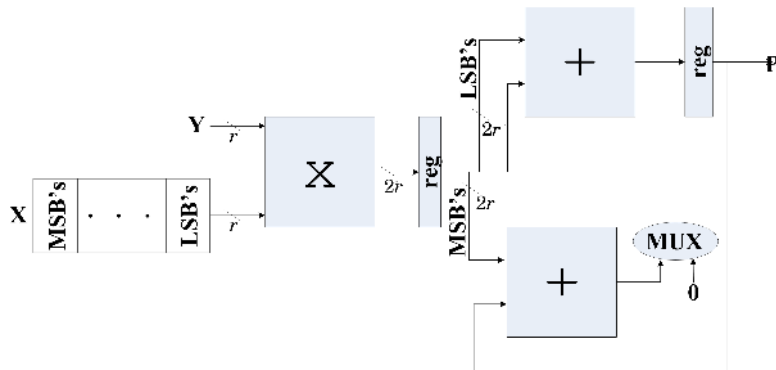


Figure 3.7: Architecture of the large operand multiplier

In case the result of the RNS-to-binary conversion is larger than p , an extra modular reduction needs to be conducted, in order to obtain the equivalent field element. Recall that the result of the RNS-to-binary conversion is a rL -bit integer, which has to be reduced \pmod{p} . Many dedicated modular reduction architectures exist in the literature, including look-up table methods or iterative algorithms [HGG07, KPH04, Par97, KH98, FP99]. Nevertheless, they are not well suited in the proposed implementation, considering that the input integer is very large and, consequently, the added area or memory penalty would be significant.

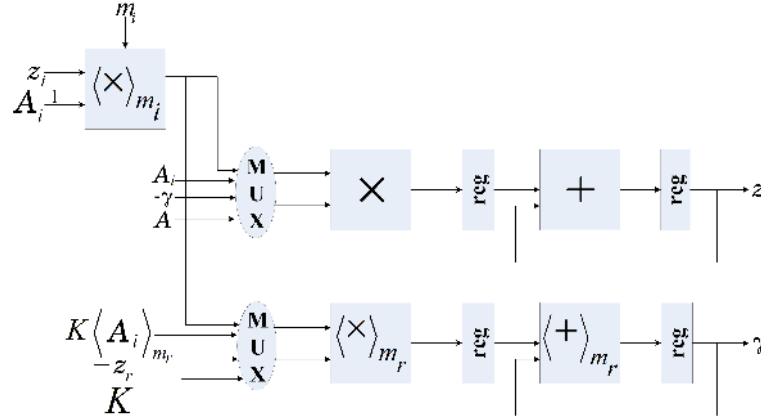


Figure 3.8: Architecture of the RNS-to-binary converter

Instead, modular reduction is conducted with the use of the modular bit-serial multiplier in Figure 3.3. Assuming c to be the result of the RNS-to-binary conversion the multiplier performs the operation $\langle 1 * c \rangle_p$, where c is fed to the shift register of the modular multiplier (see Figure 3.3). This slow but area efficient solution does not add to the execution time of the point multiplication operation. This is feasible due to the pipelined nature of the proposed architecture, considering that modular reduction is much slower than point multiplication.

The proposed converter holds some characteristics that make it suitable for large scale RNS implementations. The proposed serial converter is preferable for RNS bases with a large number of moduli, suitable for large dynamic ranges. The main reason is its low area cost compared to other conversion solutions, i.e., realization using carry-save adder trees [SBC98, Pie95].

Moreover, the designed multiplier can be reconfigured to support operands of any length, according to the needs of the implementation. The length of the operand is determined by the RNS range, which can be selected to have a length that is a multiple of the length required by the application.

Also, in cases of time-consuming and data-intensive algorithms, like in ECC, the designer has the flexibility to sacrifice some speed in favor of area, provided that the slow serial converter can be pipelined with the fast RNS core which executes the main algorithm. In that way, the internal speed gains due to the RNS are preserved and the area penalty introduced by the converter is not significant.

3.3.5 Projective-to-affine coordinates conversion

Point $[k]P$ is obtained from the RNS-to-binary converter in projective coordinates. However, in practical cryptographic applications, affine coordinates are exploited. Therefore, in order to increase the functionality of the proposed implementation, a separate module for the projective-to-affine coordinates conversion is employed.

The projective-to-affine conversion is the realization of (3.3). Let us rewrite it here for convenience:

$$x = \frac{X}{Z^2}; y = \frac{Y}{Z^3}. \quad (3.11)$$

It is apparent that one modular inversion ($T_1 = \frac{1}{Z}$) and 4 modular multiplications, namely ($T_2 = T_1^2$, $x = X * T_2$, $T_3 = T_1 * T_2$, $y = Y * T_3$) are required for the conversion.

Field characteristics recommended by the National Institute of Standards Technology (NIST) are prime numbers of special form (generalized Mersenne numbers), which can be exploited in order to implement fast modular arithmetic. Efficient modular reduction algorithms have been described [US 00], where modular reduction is replaced with simple additions. Therefore, modular multiplication described in Algorithm 3.1 was not considered for this conversion. Thus, the modular multiplier depicted in Figure 3.9 was realized by a $\lceil \log_2 p \rceil$ -bit multiplier, followed by a reduction process [US 00]. The prime field characteristics utilized in the proposed implementation are offered in Section VII.

From the many works regarding modular inversion in the literature [SK00, KAK96, Mon85, Wal99, Kal95, dDBQ04, GTK02, GT03, ZWBC02, TT04, BL06, DMP03], the implementation proposed in [BL06] is adopted, as it efficiently encompasses the proposed implementation characteristics, namely the modular inverter can support all the bit lengths used in the proposed implementation (160, 192, 224 and 256-bit). Moreover, as it is implemented in a Xilinx FPGA (Virtex 2 3000), the added area was accurately calculated according to Xilinx's FPGA data sheets [Xil05].

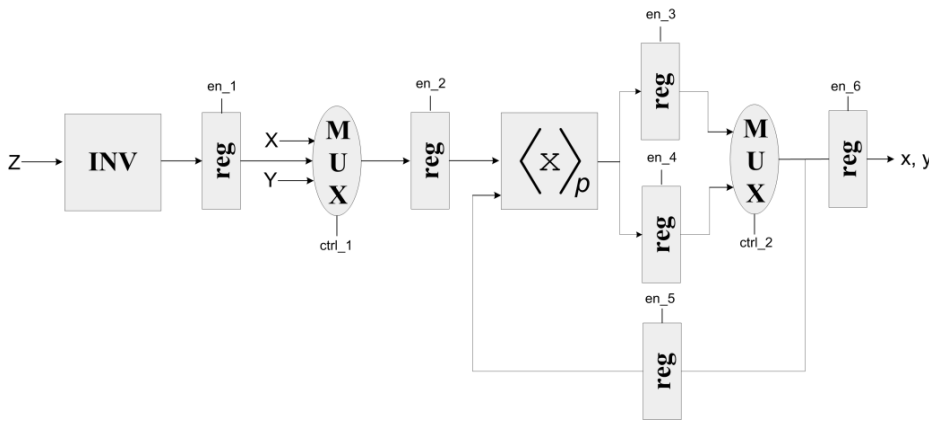


Figure 3.9: The projective-to-affine converter

The projective-to-affine converter is depicted in Figure 3.9. It consists of the modular multiplier described earlier and depicted by the symbol “ \times ”, the modular inverter of [BL06] and some control logic. The inverter first produces $T_1 = \frac{1}{Z}$, which is driven to a multiplexer, together with the X , Y coordinates. The leftmost multiplexer selects T_1 to be the first operand of the multiplier and the rightmost multiplexer selects “1” as the other operand. Then, the modular multiplier first produces $1 * T_1$, which is then driven back to the multiplier input. Following, $T_2 = T_1^2$ and $T_3 = T_1 * T_2$ are computed, which are stored in the two registers following the multiplier. The leftmost multiplexer then selects the X coordinate as

the first operand of the multiplier, and the rightmost multiplexer selects T_2 to be the second operand. The result $x = X * T_2$ is stored in the register that previously contained T_2 . Similarly, $y = Y * T_3$ is computed and the x, y coordinates are driven serially to the output register.

3.4 Performance Results and Comparisons

The proposed ECPM was synthesized in a Xilinx Virtex E-xcv1000E, FG680 FPGA device. In Table 3.1, for a full pipeline exploitation case, the timings for an elliptic curve point multiplication are given and comparisons are made with other state-of-the-art implementations. If only one point multiplication is performed, i.e., pipeline is not fully utilized by constant streams of input points, then the point multiplication timings are 4.84 ms, 4.08 ms, 3.54 ms and 2.35 ms for a 256, 224, 192 and 160-bit implementation, respectively. The field characteristic p is $p = 2^{160} + 7$ for the 160-bit implementation and NIST P-192, P-224 and P-256 ($2^{192} - 2^{64} - 1$, $2^{224} - 2^{96} + 1$, $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$) for 192-bit, 224-bit and 256-bit ECC respectively [US 00]. The corresponding RNS set for the 192-bit implementation is depicted in Table 3.2 and consists of 20 moduli, 42-bit long each. In cases where other reported implementations supported smaller or larger fields, a relevant decrease or increase, accordingly, in the RNS range was made to support those fields as well. In that way, fair and accurate results were extracted.

In the frequency and elliptic curve point multiplication time columns, four values are contained, referring to a 256, 224, 192, and 160-bit implementation. The proposed architecture achieves competitive timings compared to existing implementations. When compared to [ST03], [CBC07], and [ESG⁺05], the proposed design has smaller frequency and higher multiplication delay. However, as these works refer to ASIC implementations, the FPGA implementation presented here is still competitive. In [DMKP04], while the timings for point addition and point doubling are given, the final execution time for the point multiplication is not provided. As a result, the corresponding value, denoted with an asterisk in Table 3.1, was estimated by the authors, according to the point doubling/addition timings given in [DMKP04]. In [Wol03] the execution time of point multiplication is given in cycles and it is claimed that operations like addition, subtraction or modular addition/subtraction are executed in one cycle. However, no further information is given regarding the actual execution time of a cycle, thus no fair comparison can be made. Note that [KF07] and [Wol03] were omitted from Table 3.1, since no data are offered.

Comparisons in terms of area are difficult to be made, due to the different implementation platforms adopted in the literature. The presented implementation occupies 25,012 4-input LUTs for a 192-bit architecture. For other bit configurations, results are offered in Table 3.1. The reported area for the existing implementations was 50,000 LUTs in [SFKS06] and [SKS06], 11,416 LUTs in [OP01] and 11,227 LUTs in [OBPV03]. As far as the ASIC versions are concerned, in [XB01] no measurements are given, while in [ST03], for the fast configuration case, the circuit occupies 118K gates. In [MMM06], the presented architec-

Table 3.1: Comparison of ECPM architectures

	Field (bits)	Platform	Max Freq. (MHz)	EC mult. time (ms)	LUT	Gates (thousands)
This work	$GF(p)$ 256	Xilinx xcv1000E-8	39.7	3.95	32,716	103
	$GF(p)$ 224		46.8	3.31	29,610	93.362
	$GF(p)$ 192		52.9	2.97	25,012	78.865
	$GF(p)$ 160		58	1.77	21,140	66.656
[SFKS06] [SKS06]	$GF(p)$ 160	Xilinx Virtex2 Pro	75	2.41	50,000	157
[LH08b]	$GF(p)$ 160	Xilinx Virtex2 Pro	100	6.282	3,015	-
[OP01]	$GF(p)$ 192	Xilinx xcv1000E-8	40	3	11,416	35.983*
[OBPV03]	$GF(p)$ 160	Xilinx xcv1000E-8	91.3	14.41	11,227	35.390*
[MMM06]	$GF(p)$ 256	Xilinx Virtex2 Pro	39.46	3.86	35,450*	-
[DMKP04]	$GF(p)$ 192	Xilinx VirtexE-2000	19	9.3*	-	83.236
[SMB ⁺ 07]	$GF(p)$ 256	Xilinx Spartan3	40	17.7	-	4,100*
[ST03]	Dual-field 192	0.13m CMOS ASIC	137.7	1.44	-	118
[CBC07]	$GF(p)$ 256	0.13m CMOS ASIC	556	1.01	-	122
[XB01]	$GF(p)$ 192	ASIC	50	30	N/A	N/A
[ESG ⁺ 05]	$GF(p)$ 224	Custom processor	1,500	0.256	N/A	N/A

* values calculated by the authors

ture utilizes 15,755 CLB slices and 256 18×18 -bit embedded multipliers in a Xilinx Virtex2 Pro FPGA device. The architecture in [SMB⁺07], utilizes 27,597 slices in the Xilinx Spartan3 FPGA. In [Wol03] the authors state that only rough estimations regarding the area are given. The total number of gates and the equivalent area in a 0.35 CMOS process, in terms of mm^2 , are provided, therefore no fair comparison can be conducted. In [DMKP04] the reported area is 83,236 gates and in [LH08b] 3,015 LUTs. All area data are summarized in Table 3.1. The implementations in [XB01, Wol03, KF07, ESG⁺05] were omitted, since no comparable data are offered. Values in Table 3.1 denoted with an asterisk were calculated by the authors, according to the official Xilinx FPGA data sheets[Xil05].

In terms of versatility, the presented architecture supports prime fields, while [ST03] and [Wol03] support both prime and binary extension fields. The proposed implementation

Table 3.2: The RNS base modulus set for the 192-bit implementation

2446268224217	2446268224261	2446268224273
2446268224289	2446268224321	2446268224381
2446268224409	2446268224427	2446268224441
2446268224447	2446268224451	2446268224453
2446268224457	2446268224481	2446268224493
2446268224513	2446268224579	2446268224601
2446268224639	2446268224657	

supports various field sizes and prime characteristics, by reducing or expanding the RNS base accordingly. Furthermore, the proposed architecture embeds all the predicate converters (binary-to-RNS, RNS-to-binary, projective-to-affine). Therefore, it offers a front-end solution, which can be embedded to existing systems not exploiting RNS arithmetic or projective coordinates for the elliptic curve point representation.

Finally, the moduli that comprise the RNS base of the proposed implementation, are not of special form, i.e., Mersenne or Generalized Mersenne Numbers. Only in the case of the projective-to-affine conversion, the special form of the field characteristic p was exploited, to achieve fast modular reduction, as described in the previous sections. Comparing with other works that fully exploit field characteristics of special form, as in [OP01], the proposed architecture performs better in terms of both speed and area.

3.4.1 Impact of the number of moduli and their word-lengths on the performance

The impact of the number of moduli and their word lengths on the performance of the proposed implementation is investigated. Thus, a clear picture of whether RNS can be an efficient platform for improvement in ECC can be formed, as key lengths are expected to grow in the future.

A number of simulations were performed for various ECC implementations requiring 160, 192, 224, and 256 bits, respectively, in an effort to determine the performance of each implementation in terms of frequency and area. For each of these implementations, different combinations of number of moduli and their word lengths were employed, so that each such combination would correspond to the dynamic range required for the implementation. These combinations are shown as pairs of numbers separated by commas, under the X-axis in each of the Figures 3.10 and 3.11, which present the results of these simulations.

Figure 3.10(a) depicts the impact of the number and word length of the moduli on the ECPM's speed. For various ECC's key word-lengths, the frequency of the implementation remains the same, regardless of the number of moduli and their word lengths. In contrast, an important decrease in area is achieved as the number of moduli increases and simulta-

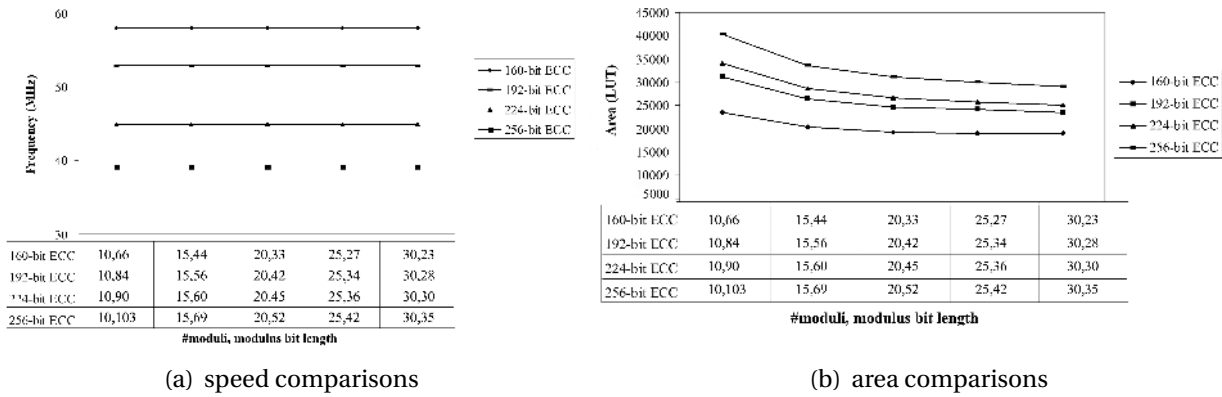
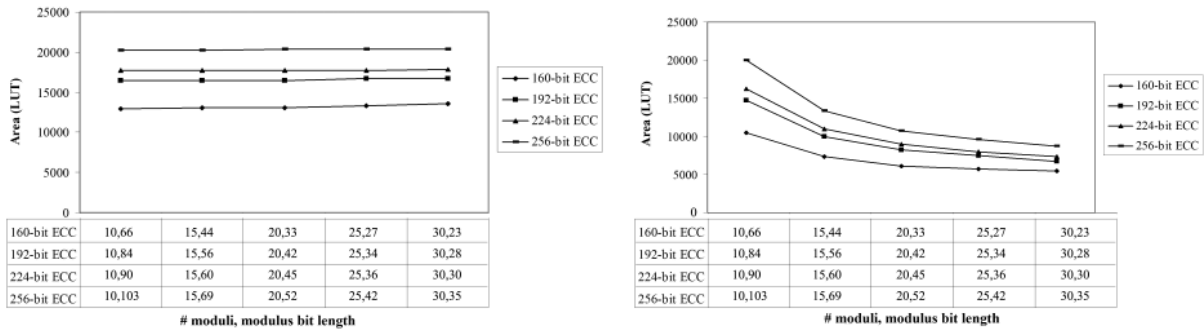


Figure 3.10: Number and word-length of moduli vs. (a) speed and (b) area



(a) Area of the ECPM without the RNS-to-binary converter (b) Area of the RNS-to-binary converter

Figure 3.11: Impact of the RNS-to-binary converter on the area of ECPM

neously their word lengths decrease. This outcome is depicted in 3.10(b). In order to further investigate the performance, two separate synthesis simulations were performed. The first was for the synthesis of the ECPM without the output converter and the second was for the output converter as a stand-alone module. The results are shown in Figures 3.11(a) and 3.11(b).

From Figure 3.11(a), it can be remarked that there is a slight increase in the area, as we transit from one configuration to another with more moduli. Therefore, it is the output converter that determines the performance depicted in Figure 3.10(b). From Figure 3.11(b), it is clear that the converter is very sensitive to the different RNS base configurations, as far as the area is concerned, and that the behavior of the overall system follows the converter's behavior. Thus, as it was stated before, the number and the word-lengths of the moduli are crucial for the converter's performance. From the previous figures, it can be derived that large modulus sets with small word lengths can achieve a beneficial area to speed ratio.

The presented simulations introduce a significant result. Assuming the same number of

moduli, for a transition from a 160-bit ECC to a 192-bit one, the required increase in the modulus word length is less than 32 bits, i.e. less than the increase of the word length of the finite field operands, in the case of typical ECC implementations. As a result, the ratio

$$\lambda = \frac{\text{ECC key length}}{\text{RNS modulus word-length}} \quad (3.12)$$

is expected to grow, as key lengths are expected to increase in the future. This behavior indicates that, as we transit from smaller ECC key word-lengths to larger, the growth of area and the consequent reduction of frequency become smaller in the RNS implementation case. Thus, RNS will remain a viable solution for the implementation of point multiplication over elliptic curves.

3.5 Pipelined RNS structures

Our previous approach towards a fully parallel RNS implementation of an ECPM was based on a “naive” approach of calculating the maximum range for calculations over the integers, that is the intermediate results in RNS format were not reduced modulo p , where p the field characteristic of $GF(p)$. This conversion was executed only once at the end of a point multiplication. The architecture achieved competitive performance compared to other RNS and non-RNS implementations, but the large dynamic range required a considerable amount of moduli channels, hence an analogous area overhead.

In the following section, an approach for RNS elliptic curve point multiplication based on the RNSMMM algorithm is considered [ESJ⁺13]. RNSMMM allows for RNS calculations, but the final result of the multiplication is already reduced modulo p , thus a significant reduction in the RNS range is achieved. In the following, a brief overview of the algorithm is provided for the needs of this section. Detailed analysis and further enhancements are presented in Chapter 4.

3.5.1 Modular multiplication in RNS

The RNSMMM is actually a transformation of the original MMM in Algorithm 2.3 to support RNS arithmetic [PP95, KKSS00]. Two RNS bases are introduced, namely $\mathcal{A} = (p_1, p_2, \dots, p_L)$ and $\mathcal{B} = (q_1, q_2, \dots, q_L)$, such that $\gcd(p_i, q_j) = 1, \forall i, j \in [1, L]$. The 5 steps of the Montgomery algorithm are translated to RNS computations in both bases, denoted from now on as $\mathcal{T} = \mathcal{A} \cup \mathcal{B}$.

Initially, the inputs a, b are expressed in RNS representation in both bases as $a_{\mathcal{T}}$ and $b_{\mathcal{T}}$. Steps 1, 3, and 4 of Algorithm 2.3, involve addition and multiplication operations, thus their transformation to RNS is straightforward. For steps 2 and 5, the Montgomery radix R is replaced by $B = \prod_{i=1}^L q_i$, which is the range of \mathcal{B} . We also denote as $A = \prod_{i=1}^L p_i$ the range of base \mathcal{A} . Then, in the second step, t in RNS format is computed in base \mathcal{B} by $t_{\mathcal{B}} = s_{\mathcal{B}} \cdot p_{\mathcal{B}}^{-1}$. Nevertheless, the computations in base \mathcal{B} can't be continued for steps 3, 4, and 5 of Algorithm 2.3, since in step 5 we would need to compute a quantity of the form

$B^{-1} \bmod q_i$, which does not exist since q_i s are factors of B . Thus, a base conversion Base Conversion (BC) step, from base \mathcal{B} to base \mathcal{A} , is inserted, to compute $t_{\mathcal{A}}$. $t_{\mathcal{A}}$ is then used to execute the old steps 3, 4, and 5 in base \mathcal{A} . The result at the end of this algorithm is a quantity $c_{\mathcal{T}}$ in RNS format that equals $c \equiv abB^{-1} \bmod p$, since BC is error-free.

Algorithm 3.2 RNS Montgomery Modular Multiplication (RNSMMM)

Input: $a_{\mathcal{T}}, b_{\mathcal{T}} \{ a, b < 2p \}$

Output: $c_{\mathcal{T}}, \{ c < 2p \text{ and } c \equiv abB^{-1} \bmod p \}$

Precompute: $(-p^{-1})_{\mathcal{B}}, B_{\mathcal{A}}^{-1}, p_{\mathcal{A}}$

- 1 $s_{\mathcal{T}} \leftarrow a_{\mathcal{T}} \cdot b_{\mathcal{T}}$
 - 2 $t_{\mathcal{B}} \leftarrow s_{\mathcal{B}} \cdot (-p^{-1})_{\mathcal{B}}$
 - 3 $t_{\mathcal{A}} \leftarrow t_{\mathcal{B}} \{ \text{base conversion step} \}$
 - 4 $u_{\mathcal{A}} \leftarrow t_{\mathcal{A}} \cdot p_{\mathcal{A}}$
 - 5 $v_{\mathcal{A}} \leftarrow s_{\mathcal{A}} + u_{\mathcal{A}}$
 - 6 $c_{\mathcal{A}} \leftarrow v_{\mathcal{A}} \cdot B_{\mathcal{A}}^{-1}$
 - 7 $c_{\mathcal{B}} \leftarrow c_{\mathcal{A}} \{ \text{base conversion step} \}$
-

Clearly, the total complexity of the algorithm is determined by the BC steps. A BC transforms an integer expressed in an RNS base \mathcal{A} to an another base \mathcal{B} . A BC is essentially a residue-to-binary conversion while the final result is computed modulo each modulus of the new RNS base. In this context, the methods employed for BC are either CRT-based methods according to (2.29) or MRC-based methods according to (2.30), (2.31). In the proposed implementation an MRC-based method is utilized. Equation (2.31) is first employed to obtain the MRC digits of the result, while (2.30) is computed modulo each modulus of the new base.

Sets of three and four moduli are proposed in order to implement the RNSMMM of Algorithm 3.2. The form of the moduli determines the efficiency of the arithmetic operations and the structure of the residue-to-binary and binary-to-residue converters [NME11]. The RNS bases employed are shown in Table 3.3. In the first base, RNS moduli of the form $2^k - 2^{t_i} - 1$, where $t_i < k/2$ are employed, which offer simple modulo reduction operations [BKP09].

The second base is realized by sets of three and four moduli of the special forms $\{2^k, 2^{k+1} - 1, 2^k - 1\}$ [Moh07] and $\{2^k, 2^k - 1, 2^{k+1} - 1, 2^{k-1} - 1\}$, which also provide efficient arithmetic operations and residue-to-binary and binary-to-residue conversions [BKP09]. In order to use the result of RNSMMM in subsequent modular multiplications, it is required that $4p < P < Q$ [BKP09]. It is easy to check that the employed bases are sufficient to cover this requirement.

3.5.2 Design

3.5.2.1 Modular adders and multipliers

For the first base, where moduli of the form $2^k - 2^{t_i} - 1$ are used, the modular adder and multiplier depicted in Figure 3.12 are employed. Regarding modular multiplication, two

Table 3.3: Proposed RNS Bases

	Field (bit)	First Base \mathcal{A}	Second Base \mathcal{B}
Three-modulus RNS bases (D1)	160	$\{2^{56} - 2^{11} - 1, 2^{56} - 2^{16} - 1, 2^{56} - 2^{20} - 1\}$	$\{2^{56}, 2^{56} - 1, 2^{57} - 1\}$
Three-modulus RNS bases (D2)	192	$\{2^{66} - 2^{17} - 1, 2^{66} - 2^{18} - 1, 2^{66} - 2^{24} - 1\}$	$\{2^{66}, 2^{66} - 1, 2^{67} - 1\}$
Four-modulus RNS bases (D3)	192	$\{2^{50} - 2^{20} - 1, 2^{50} - 2^{22} - 1, 2^{50} - 2^{18} - 1, 2^{50} - 2^{10} - 1\}$	$\{2^{50}, 2^{50} - 1, 2^{51} - 1, 2^{49} - 1\}$
Four-modulus RNS bases (D4)	224	$\{2^{58} - 2^{22} - 1, 2^{58} - 2^{13} - 1, 2^{58} - 2^{10} - 1, 2^{58} - 2^{16} - 1\}$	$\{2^{58}, 2^{58} - 1, 2^{59} - 1, 2^{57} - 1\}$
Four-modulus RNS bases (D5)	256	$\{2^{66} - 2^{22} - 1, 2^{66} - 2^{24} - 1, 2^{66} - 2^{18} - 1, 2^{66} - 2^{17} - 1\}$	$\{2^{66}, 2^{66} - 1, 2^{67} - 1, 2^{65} - 1\}$

k -bit operands are multiplied and a $2k$ -bit value is obtained. Modular reduction of a $2k$ -bit value w with moduli of the form $2^k - 2^{t_i} - 1$ can be written using its higher k bits, denoted as w_h , and its k lower bits, denoted as w_l , as

$$w = \langle w_h 2^k + w_l \rangle_{2^k - 2^{t_i} - 1}. \quad (3.13)$$

Since $2^k \pmod{(2^k - 2^{t_i} - 1)} = 2^{t_i} + 1$, it holds that

$$w = \left\langle \underbrace{w'_h}_{t_i \text{ bit}} 2^k + \underbrace{w'_{hl}}_{k \text{ bit}} + w_h + w_l \right\rangle_{2^k - 2^{t_i} - 1} \quad (3.14)$$

$$w' = \left\langle \underbrace{w'_{hh}}_{t_i \text{ bit}} \overbrace{0 \dots 0}^{t_i \text{ bit}} + \underbrace{w'_{hh} + w'_{hl}}_{t_i \text{ bit}} \right\rangle_{2^k - 2^{t_i} - 1}. \quad (3.15)$$

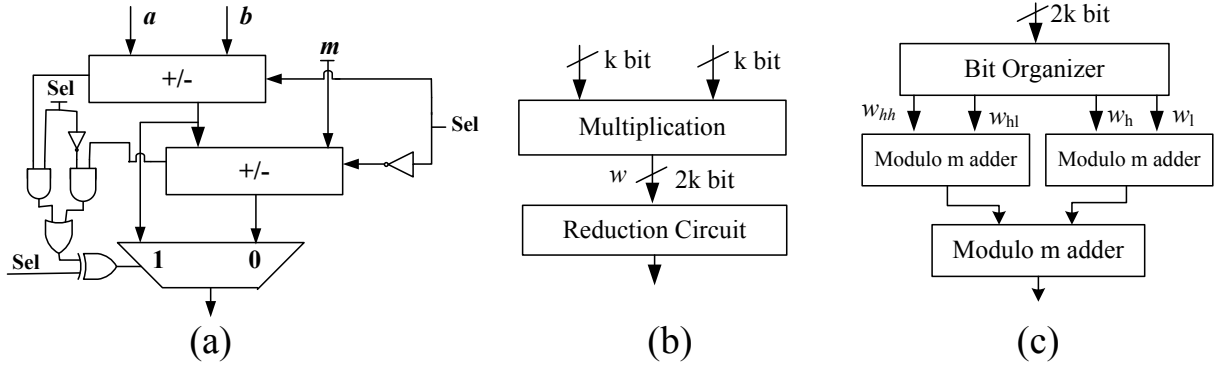


Figure 3.12: (a) Modulo m adder/subtractor [SFM⁺09], (b) Proposed modulo $2^k - 2^{t_i} - 1$ multiplier, (c) Reduction circuit

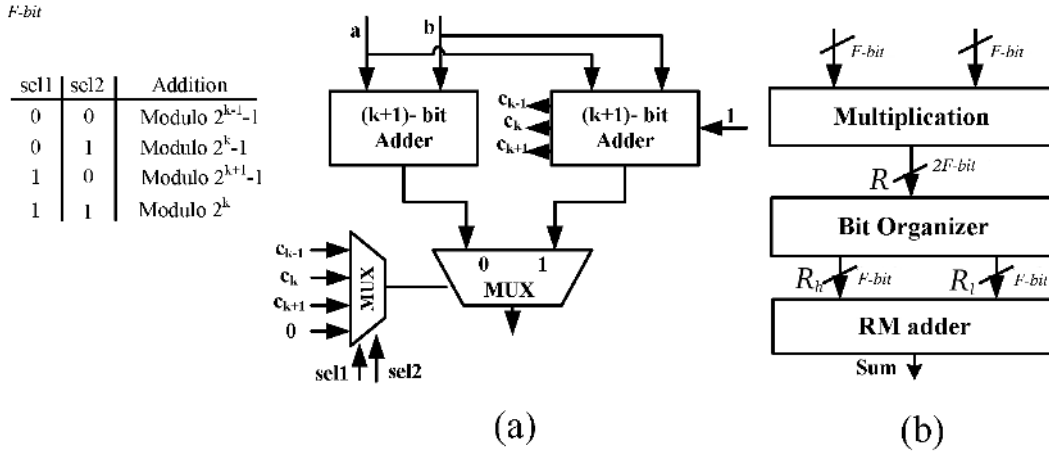


Figure 3.13: (a) Proposed reconfigurable modular (RM) adder, (b) Proposed RM Multiplier, ($F = k, k - 1, k + 1$)

Since w'_{hh} has t_i bits, it can be concatenated at the end of $w'_{hh} \underbrace{0 \dots 0}_{t_i \text{ bit}}$. Therefore w can be calculated by

$$w = \left\langle \underbrace{w'_{hh} w'_{hh}}_{w_{hh}} + w'_{hl} + w_h + w_l \right\rangle_{2^k - 2^{t_i} - 1} \quad (3.16)$$

For the second base, a reconfigurable modular (RM) adder is employed shown in Figure 3.13. Based on the proposed adder, addition and multiplication modulo 2^k , $2^{k-1}-1$, 2^k-1 , and $2^{k+1}-1$ can be done without hardware redundancy. Note that the RM adder shown in Figure 3.13 has $(k - 1)$ -bit delay of full adder less than the modulo m adder (Figure 3.12) in the worst case, thus the second base supports more efficient arithmetic operations. The RM multiplier is shown in Figure 3.13. After multiplication, the $2F$ -bit result R is split into two F -bit LSB and MSB parts (R_l and R_h respectively) ($F = k, k - 1, k + 1$) and reduction modulo $2^F - 1$ can be achieved by a modular addition of R_l and R_h [BKP09].

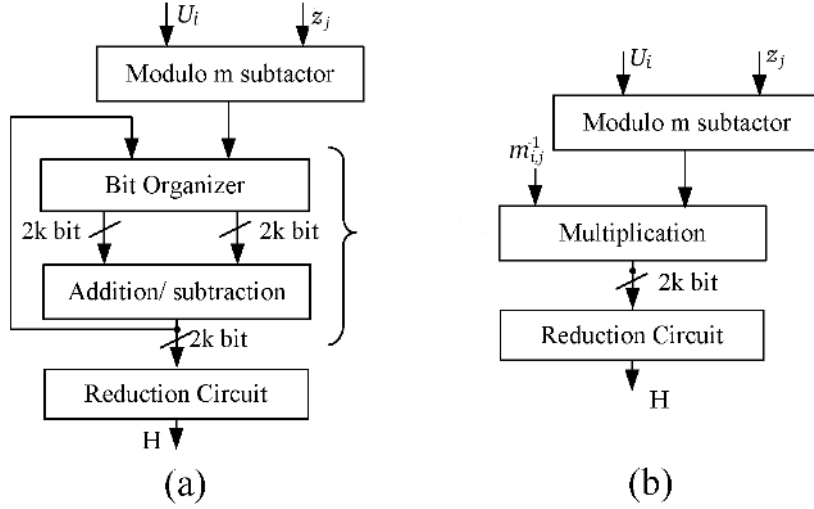


Figure 3.14: Calculation of H in RNS to MRS conversion for the first base (a) area efficient design, (b) Fast design

3.5.2.2 Conversion from base \mathcal{B} to base \mathcal{A}

In step 3 of RNSMMM a base conversion from base \mathcal{B} to base \mathcal{A} is required, which consists of a residue-to-MRS conversion in base \mathcal{B} and then a MRS-to-residue conversion in base \mathcal{A} . Efficient RNS to MRS conversion for base \mathcal{A} is reported in [BKP09]. The core operation in calculation of $U_i, \forall i = 2, 3, 4$ in (2.31) is

$$H = \left\langle (z_j - U_i) m_{i,j}^{-1} \right\rangle_{m_j}. \quad (3.17)$$

Hardware implementations of (3.17) for area and time efficient designs are shown in Figure 3.14. Considering four-modulus RNS bases, for each U_i ($i = 2, 3, 4$), an implementation shown in Figure 3.14 is employed. The bit organizer provides the required shifts according to pre-calculated multiplicative inverses.

Residues in \mathcal{A} must be calculated after the calculation of mixed radix digits in base \mathcal{B} . In the calculation of MRS to RNS from \mathcal{B} to base \mathcal{A} for four-modulus RNS bases, it holds that

$$z_j = \langle U_1 + m_1 (U_2 + m_2 (U_3 + m_3 U_4)) \rangle_{m_j}, \quad (3.18)$$

where m_j are the moduli $2^k, 2^k - 1, 2^{k+1} - 1$, and $2^{k-1} - 1$. $m_i = m_1, m_2$, and m_3 are the moduli of the form $2^k - 2^{t_i} - 1$. Based on the form of the considered bases with simple multiplicative inverses, for fast and area efficient design, adder-based structure can be simply realized by using one RM adder for each modulus.

3.5.2.3 Conversion from base \mathcal{A} to base \mathcal{B}

In order to mechanize RNS-to-MRS conversion in base $\mathcal{A} = \{2^k, 2^k - 1, 2^{k+1} - 1\}$, based on (2.31) and considering $m_1 = 2^k, m_2 = 2^k - 1, m_3 = 2^{k+1} - 1$, we get

$$U_1 = z_1 \quad (3.19)$$

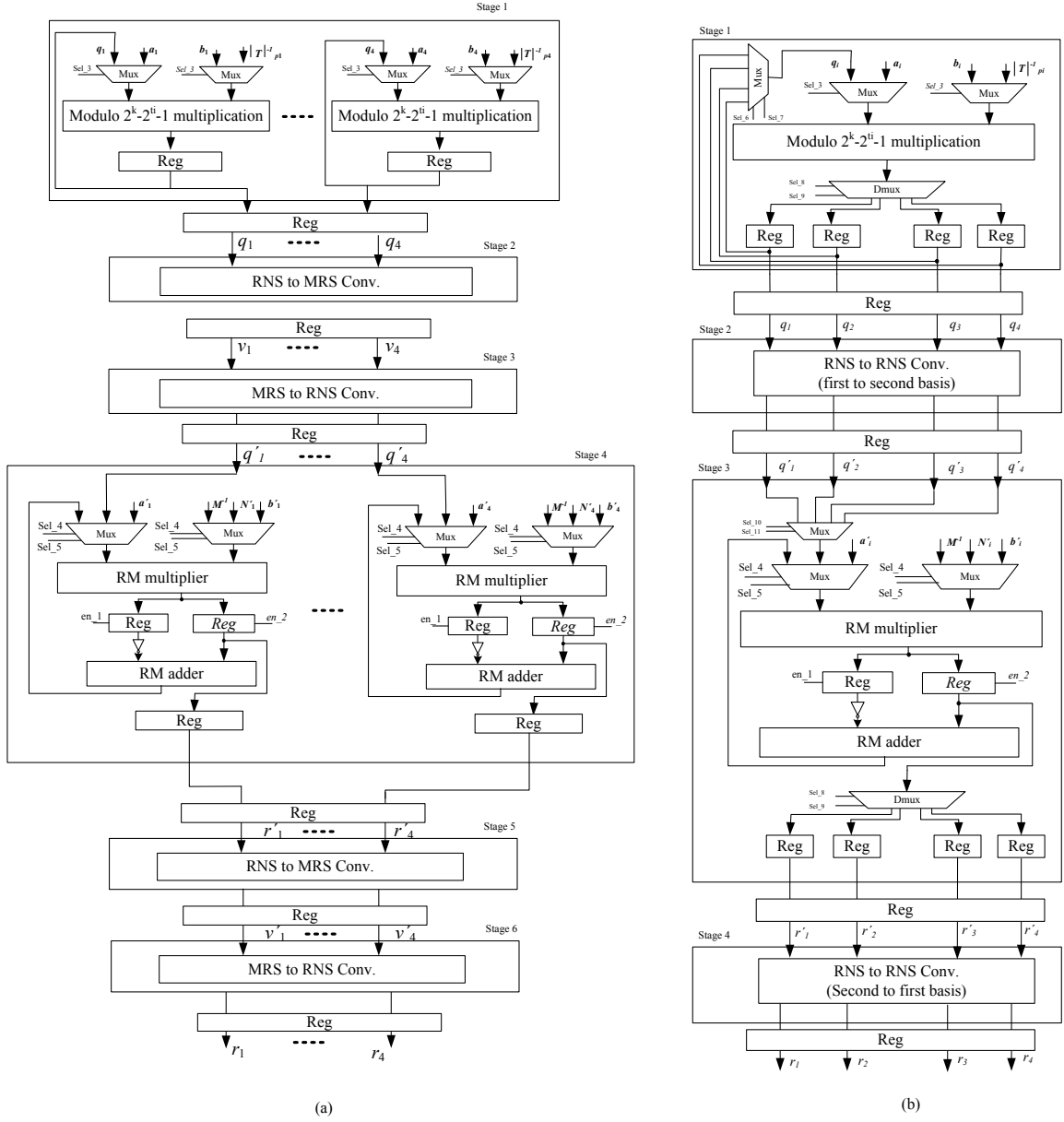


Figure 3.15: RNSMMM architecture, (a) Fast design, (b) Area efficient design

$$U_2 = \langle (z_2 - U_1) m_{1,2}^{-1} \rangle_{m_2} \quad (3.20)$$

$$U_3 = \langle ((z_3 - U_1) m_{1,3}^{-1} - U_2) m_{2,3}^{-1} \rangle_{m_3}. \quad (3.21)$$

The required multiplicative inverses in (3.20) and (3.21) are $\langle m_1^{-1} \rangle_{m_2} = 1$, $\langle m_1^{-1} \rangle_{m_3} = 2$ and $\langle m_2^{-1} \rangle_{m_3} = -2$ [Moh07]. Due to the simple form of multiplicative inverses, the proposed adder-based structure was employed both for the fast and the area efficient design.

Regarding the MRS-to-RNS conversion to base \mathcal{B} , it holds that

$$z_j = \langle U_1 + m_1 (U_2 + m_2 (U_3 + m_3 U_4)) \rangle_{m_j}. \quad (3.22)$$

It is apparent that all calculations in (3.22) consist of simple shifts and addition operations.

3.5.3 Hardware Architecture for RNS Montgomery multiplication

The proposed architectures for the RNSMMM are shown in Figure 3.15. The area efficient architecture consists of one modulo $(2^k - 2^{t_i} - 1)$ multiplier, one RM multiplier, one RM adder and two base conversion units with adder-based structure, connected in a four-stage pipelined fashion (Figure 3.15b).

The alternate design optimized for high-speed is implemented in a six-stage pipelined architecture, shown in Figure 3.15a. In each modulus channel in stages one and four of the pipelined implementation, the modular multipliers and adders in Figures 3.12 and 3.13 are employed. For the base conversion operations, the modulo adders and multipliers described in previous subsections are utilized.

3.6 Implementation details of ECPM and comparisons

Tables 3.4 summarizes the delay and area comparisons of the proposed ECPM with recent state-of-the-art works. The field characteristic is $p = 2^{160} + 7$ for a 160-bit implementation and NIST recommendations for p -192, p -224 and p -256 corresponding to $2^{192} - 2^{64} - 1$, $2^{224} - 2^{96} + 1$, and $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, respectively [SFM⁺09].

Regarding point multiplication, the binary method is employed. For the point addition and doubling operations, the optimized DFGs presented in the previous sections as well as in [SFM⁺09] are employed. Compared to [SFM⁺09] fewer moduli are required in the proposed architecture. Moduli in Table 3.2 are of no special form, while well-formed moduli employed here lead to a very simple reverse converter structure. The required conversions consist of a simple adder-based structure, while the computations are pipelined with the main ECPM core.

The architecture for the proposed ECPM consists of a binary-to-residue converter, a register file, one RNSMMM, a control unit, and a residue-to-binary converter. In the proposed architecture a binary-to-residue converter is employed to compute the RNS representation of the projective coordinates (X, Y, Z) . Due to the use of well-formed moduli, adder-based structure can be utilized for the binary-to-residue conversion as discussed in [BKP09]. The control unit provides the required input operands and control signals for the arithmetic unit, while the distributed RAMs on FPGA are used in designing the register file. Residue-to-binary conversion can be also realized by an adder-based structure according to (2.30) and (2.31).

Due to the pipelined implementation of the RNSMMM, reduction in the execution time of a point multiplication is also achieved. Unlike [SFM⁺09], [Gui10], fewer number of moduli are required and the use of well-formed RNS bases results in efficient realization of modular addition and multiplication required in the RNSMMM. Another advantage of the proposed architecture is the simple, adder-based implementation of the residue-to-binary converter,

compared to [SFM⁺09], which requires multiplications by large operands. The proposed ECPM architecture is implemented on Xilinx VirtexE, Virtex 2 Pro and Altera Startix II to achieve a fair, 1-to-1 comparison with the state-of-the-art implementations in Table 3.4. In [Gui10], although moduli of smaller word-length are employed, a large number of moduli is utilized and thus the RNS base conversions are complex resulting to worse performance, as shown in Table 3.4.

3.7 Summary

In this chapter two RNS implementations of an ECPM were presented. A DFG approach for the optimization of point addition and doubling was utilized for both designs to achieve the same number of execution steps for both operations. For the first design an appropriate RNS range was selected to accommodate the full range of calculations without intermediate modular reductions, while the second design employed the RNSMMM algorithm.

For the first architecture extra care to the design for the output RNS-to-binary converter was given. A specially designed bit-serial multiplier was developed to handle large operands. The multiplier was then embedded in the architecture of the converter, forming a serial design suitable for large RNS ranges.

Area and timing results were offered, proving the efficiency of the proposed implementation even toward dedicated ASIC implementations. A study for various key lengths, number of RNS moduli and modulus bit lengths was also performed. It was proved that, in comparison to traditional arithmetic approaches, RNS has the tendency to perform increasingly better as the key word-lengths of an ECC will increase in the near future.

The second design improved significantly our first effort, by reducing the number of moduli channels required and by utilizing moduli of special form. This amounted to reduction of area and speed-ups in terms of total execution time for one point multiplication.

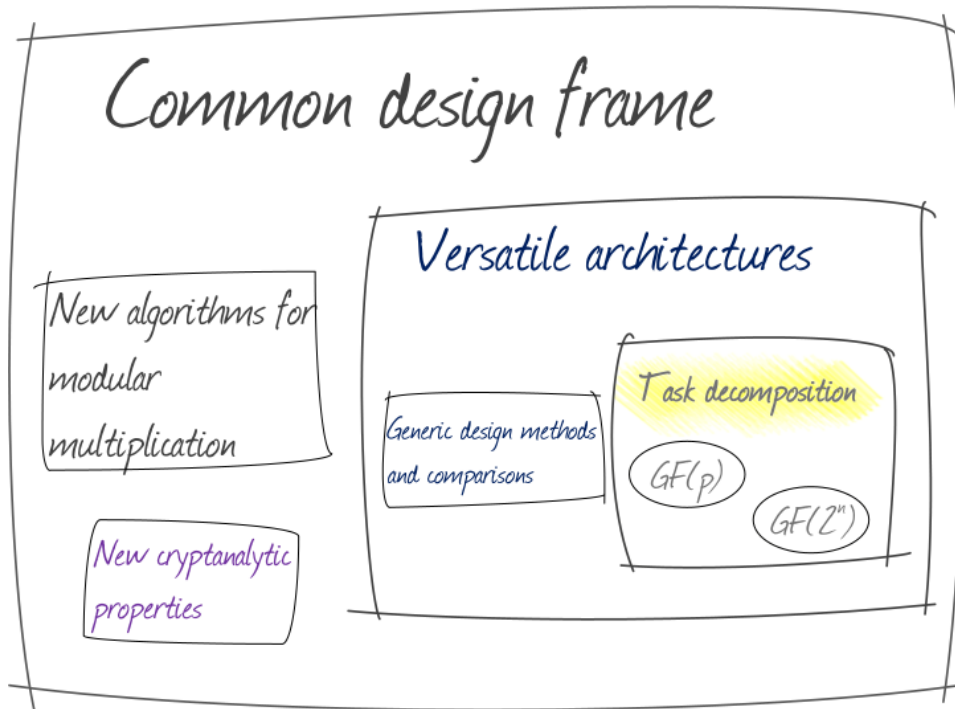
Table 3.4: Comparison of ECPM architectures

	Field (bits)	Max Freq. (MHz)	EC mult. time (ms)	Area
3-moduli (area efficient)	192	34.7	2.56	20,014 LUT
	160	38.2	1.83	15,448 LUT
4-moduli (area efficient)	256	34.7	3.41	28,318 LUT
	224	37.0	2.92	25,912 LUT
	192	38.4	2.67	21,380 LUT
[SFM ⁺ 09]	256	39.7	3.95	32,716 LUT
	224	46.8	3.31	29,610 LUT
	192	52.9	2.97	25,012 LUT
[OP01]	192	40	3	11,416 LUT
[OBPV03]	160	91.3	14.41	11,227 LUT
3-moduli (area efficient)	192	50.2	1.82	9,310 LUT
	160	52.5	1.36	8,742 LUT
4-moduli (area efficient)	256	50.2	2.62	18,942 LUT
	192	53.6	2.07	14,782 LUT
3-moduli (fast design)	192	50.2	0.35	19,224 LUT
	160	52.5	0.31	18,114 LUT
4-moduli (fast design)	256	50.2	0.59	28,746 LUT
	192	53.6	0.52	25,304 LUT
[MMM06]	256	39.46	3.86	35,450 LUT
[SFKS06]	160	75	2.41	50,000 LUT
[LH08a]	256	94.7	2.66	41,595 Slices
	192	94.7	1.25	40,219 Slices
	160	94.7	0.78	39,531 Slices
3-moduli (fast design)	192	54.1	0.33	5,248 ALUT
	160	56.8	0.29	4,892 ALUT
4-moduli (fast design)	256	54.1	0.54	12,324 ALUT
	192	59.9	0.42	7,932 ALUT
	192	61.2	0.38	5,148 ALUT
[Gui10]	256	157.2	0.68	9,177 ALUT
	192	160.5	0.44	6,203 ALUT
	160	165.5	0.32	5,896 ALUT

¹ Implemented on Xilinx VirtexE ² Implemented on Xilinx Virtex 2 Pro

³ Implemented on Altera Stratix II

New RNS architectures for $GF(p)$ and $GF(2^n)$



This chapter presents an important class of algorithms that formed the basis of the proposed versatile architectures, namely the RNS Montgomery Modular Multiplication (RNSMMM) and PRNS Montgomery Modular Multiplication (PRNSMMM) algorithms. The most important features and characteristics of these algorithms are analyzed. New, improved versions for both algorithms are proposed, while an algorithmic and architectural analysis proves the superiority of the proposed solutions compared to existing ones.

4.1 Overview of RNS Montgomery modular multiplication

The previous chapter presented one of the first applications of RNS in ECC. The main characteristic of this methodology was the calculation of the maximum dynamic range required for a point-addition and point-doubling operation. Subsequently, an RNS base that could accommodate this range was selected. This method is actually a translation of integer arithmetic to a RNS system, and the final modulo reduction by the field modulus p of $GF(p)$ is executed once at the end of each point addition or doubling operations.

This solution, however, did not take into account the possibility of embedding modular arithmetic within an RNS system, that is to perform calculations using RNS but at the same time the result is reduced modulo p , where p the field characteristic of $GF(p)$. An important advance towards this direction was the introduction of RNS to MMM. To the best of our knowledge, the authors in [PP95] were the first to propose such a type of algorithm. Later, researchers produced more robust algorithms and implementations, mainly for use in the context of modular exponentiation for RSA. The complete algorithms are described by Kawamura et al. in [KKSS00], Bajard et al. in [BI04] and Gandino et al. in [GLMB11]. Let us rewrite here the original MMM for convenience.

Algorithm 4.1 Montgomery Modular Multiplication MMM

Input: $a, b, N, R, R^{-1} \{ a, b < N \}$

Output: $c \equiv abR^{-1} \pmod{N}, \{ c < 2N \}$

- 1 $s \leftarrow a \cdot b$
 - 2 $t \leftarrow s \cdot (-N^{-1}) \pmod{R}$
 - 3 $u \leftarrow t \cdot N$
 - 4 $v \leftarrow s + u$
 - 5 $c \leftarrow v/R$
-

The challenges to transform this algorithm to RNS format are steps 2 and 5. Step 2 is a modulo R operation, where R is the Montgomery radix. In non-RNS implementations R is usually chosen to be a power of 2, thus modulo R operations amount to simple shifts. A method that provides the modulo operation of step 2 within RNS representation needs to be devised for the RNS version of the algorithm. Similar problems arise for step 5, where division by the Montgomery radix needs to be performed in RNS.

The trick to overcome these issues is to choose a new Montgomery radix. Assume that a base $\mathcal{B} = \{q_1, q_2, \dots, q_L\}$ is employed with a corresponding range B . By assigning the Montgomery radix to be the range B itself, step 2 is transformed to a step computed modulo B , since RNS is a closed modulo B system. Unfortunately, computations in the same base cannot be continued, since in step 5 a quantity of the form $B^{-1} \pmod{B}$ needs to be computed, which mathematically does not exist. For this reason, a new base $\mathcal{A} = \{p_1, p_2, \dots, p_L\}$ is employed and a BC from base \mathcal{B} to base \mathcal{A} is performed at step 2 of the algorithm. A BC is the transformation of an RNS representation in a base \mathcal{B} to another base \mathcal{A} .

Computations for steps 3, 4 and 5 are computed in the new base \mathcal{A} , since now it is feasible to compute $B^{-1} \pmod{A}$ in step 5. The complete RNSMMM algorithm is shown below.

Algorithm 4.2 RNS Montgomery Modular Multiplication (RNSMMM)

Input: $a_{\mathcal{T}}, b_{\mathcal{T}} \{ a, b < 2N \}$

Output: $c_{\mathcal{T}}, \{ c < 2N \text{ and } c \equiv abB^{-1} \pmod{N} \}$

Precompute: $(-N^{-1})_{\mathcal{B}}, B_{\mathcal{A}}^{-1}, N_{\mathcal{A}}$

- 1 $s_{\mathcal{T}} \leftarrow a_{\mathcal{T}} \cdot b_{\mathcal{T}}$
 - 2 $t_{\mathcal{B}} \leftarrow s_{\mathcal{B}} \cdot (-N^{-1})_{\mathcal{B}}$
 - 3 $t_{\mathcal{A}} \leftarrow t_{\mathcal{B}} \{ \text{base conversion step} \}$
 - 4 $u_{\mathcal{A}} \leftarrow t_{\mathcal{A}} \cdot N_{\mathcal{A}}$
 - 5 $v_{\mathcal{A}} \leftarrow s_{\mathcal{A}} + u_{\mathcal{A}}$
 - 6 $c_{\mathcal{A}} \leftarrow v_{\mathcal{A}} \cdot B_{\mathcal{A}}^{-1}$
 - 7 $c_{\mathcal{B}} \leftarrow c_{\mathcal{A}} \{ \text{base conversion step} \}$
-

An extra BC is issued at the end of the algorithm so that inputs and outputs are compatible with each other, thus the algorithm may be used repeatedly in the context of any modular exponentiation algorithm. Steps performed in both bases are denoted as $\mathcal{T} = \mathcal{A} \cup \mathcal{B}$. Clearly, the complexity of the algorithm depends on the BC steps.

The algorithms for RNSMMM proposed by Kawamura et al. in [KKSS00] and Bajard et al. in [BI04] differ only in the way the BC is performed. Gandino et al. offered optimizations for both algorithms by reducing the total number of steps required using pre-computations [GLMB11].

4.1.1 Base Conversion (BC) by Kawamura et al.

In 2000, Kawamura et al. presented the first practical method for base conversion [KKSS00]. Their approximation method evaluates the correction factor γ of the CRT in (2.29). Starting from (2.29), and substituting by

$$\xi_i = \langle z_i \cdot A_i^{-1} \rangle_{m_i} \quad (4.1)$$

we obtain

$$z = \sum_{i=1}^L \xi_i \cdot A_i - \gamma A. \quad (4.2)$$

Dividing both sides by A , we obtain

$$\sum_{i=1}^L \frac{\xi_i}{m_i} = \frac{z}{A} + \gamma. \quad (4.3)$$

Since $0 \leq z/A < 1$, $\gamma \leq \sum_{i=1}^L \frac{\xi_i}{m_i} < \gamma + 1$ holds. Therefore,

$$\gamma = \left\lfloor \sum_{i=1}^L \frac{\xi_i}{m_i} \right\rfloor \quad (4.4)$$

with $0 \leq \gamma < L$, since $0 \leq \xi_i/m_i < 1$.

Two approximations were employed in [KKSS00]. The denominator m_i is replaced by 2^r , where $2^{r-1} < m_i \leq 2^r$ while the numerator ξ_i is approximated by its most significant q bits, where $q < r$. Thus, instead of γ , an approximated value γ^* can be calculated by

$$\gamma^* = \left\lfloor \sum_{i=1}^L \frac{\text{trunc}(\xi_i)}{2^r} + \alpha \right\rfloor, \quad (4.5)$$

where $\text{trunc}(\xi_i) = \xi_i \wedge \overbrace{(1 \dots 1)}^q \overbrace{(0 \dots 0)}^{(r-q)}$ and \wedge denotes an AND operation. An offset value $0 \leq \alpha < 1$ is introduced to compensate the error produced by the approximations. Division by powers of 2 are simple shifts thus (4.5) can be realized by additions alone. A theorem that provides the offset value α , so that the error issued by the approximations is zero, was also provided in [KKSS00]. The complete BC algorithm is shown below as Algorithm 4.3.

Algorithm 4.3 Base Conversion (BC) algorithm by Kawamura et al. [KKSS00]

Input: $\zeta_B = (\zeta_1, \zeta_2, \dots, \zeta_L)$, A, B, α

Output: $\zeta_A = (\zeta'_1, \zeta'_2, \dots, \zeta'_L)$

Precompute: $(B_i^{-1})_{q_i}, (B_i)_A (\forall i = 1 \dots L), (-B)_A$

```

1   $\sigma_0 = \alpha$ 
2  for all  $i = 1 \dots L$  do
3       $\xi_i = \langle \zeta_i \cdot B_i^{-1} \rangle_{q_i}$ 
4       $\delta_{i,0} = 0$ 
5  end for
6  for all  $i = 1 \dots L$  do
7      for  $j = 1 \dots L$  do
8           $\sigma_j = \sigma_{(j-1)} + \text{trunc}(\xi_j)/2^r$ 
9           $\gamma_j^* = \lfloor \sigma_j \rfloor, \{\gamma_j^* = \{0, 1\}\}$ 
10          $\sigma_j = \sigma_j - \gamma_j^*$ 
11          $\delta_{i,j} = \delta_{i,(j-1)} + \xi_j \cdot \langle B_j \rangle_{p_i} + \gamma_j^* \cdot \langle -B \rangle_{p_i}$ 
12     end for
13 end for
14 for all  $i = 1 \dots L$  do
15      $\zeta'_i = \langle \delta_{i,L} \rangle_{p_i}$ 
16 end for

```

Kawamura et al. offered theorems which provide optimum values for the quantity α , so that the error introduced by their approximation method is equal to 0 [KKSS00] ($\alpha = 0$ in the first BC and $\alpha = 0.5$ in the second one). There is however a limitation to this method, namely the input operand that is to be extended should not be too close to the range B [KKSS00].

4.1.2 Base Conversion (BC) by Bajard et al.

Bajard et al. employed two different methods for the first and second BC of the RNSMMM [BI04]. The first, shown below as Algorithm 4.4, issues an approximation error but performs faster than Kawamura's algorithm, while the second depicted as Algorithm 4.5, was origi-

nally proposed by Shenoy and Kumaresan [SK89] and corrects the previous result. The two algorithms convert the corresponding quantities in Algorithm 4.2, i.e., the values t and c respectively. Note also that, the first algorithm is nothing more than the CRT expression in (2.28), while the second computes the correction factor γ of the CRT expression in (2.29).

The key idea is that we can relax our restriction to obtain a correct result during the first base conversion, since, as it was shown in [BI04], if larger bases are selected, such as $A, B > N(L+2)^2$, then this relaxation does not affect the final result.

Algorithm 4.4 First BC algorithm by Bajard et al. [BI04]

Input: $t_{\mathcal{B}} = (t_1, t_2, \dots, t_L)$

Output: $t_{\mathcal{A} \cup p_r} = (t'_1, t'_2, \dots, t'_L, t'_r)$

Precompute: $\langle B_i^{-1} \rangle_{q_i} (\forall i = 1 \dots L, (B_i)_{\mathcal{A} \cup p_r})$

1 $q_i = t_i \cdot \langle B_i^{-1} \rangle_{q_i}, \forall i = 1 \dots L$

2 **for all** $j = 1 \dots L$ and $j = r$ **do**

3 $t'_j = \langle \sum_{i=1}^L q_i \cdot B_i \rangle_{p_j}$

4 **end for**

Algorithm 4.5 Second BC algorithm by Bajard et al. [BI04]

Input: $c_{\mathcal{A} \cup p_r} = (c_1, c_2, \dots, c_L, c_r)$

Output: $c_{\mathcal{B}} = (c'_1, c'_2, \dots, c'_L)$

Precompute: $\langle A_j^{-1} \rangle_{p_j} (\forall j = 1 \dots L, r), (-A)_{\mathcal{B}}, (A_j)_{\mathcal{B}}, \langle A_j \rangle_{p_r} (\forall j = 1 \dots L)$

1 $\tilde{c}_j = \langle c_j \cdot A_j^{-1} \rangle_{p_j}, \forall j = 1 \dots L$

2 $c''_r = \langle \sum_{j=1}^L \tilde{c}_j \cdot A_j \rangle_{p_r}$

3 $\gamma = \langle (c''_r - c_r) A_r^{-1} \rangle_{p_r}$

4 $c'_i = \langle \sum_{j=1}^L \tilde{c}_j \cdot A_j \rangle_{q_i}, \forall i = 1 \dots L$

5 $c'_i = \langle c'_i - \gamma A \rangle_{q_i}, \forall i = 1 \dots L$

4.1.3 Base Conversion (BC) by Gandino et al.

Gandino et al. [GLMB11] improved both of the solutions presented previously by issuing pre-computation steps. In the following, values with a \widehat{hat} symbol denote values multiplied by A_j^{-1} in base \mathcal{A} , where $A_j = A/p_j, \forall j = 1 \dots L$. The re-organized versions of Kawamura's et al. BC are shown below as Algorithms 4.6 and 4.7, while the re-organized versions of Bajard et al. BC correspond to Algorithms 4.8 and 4.9 respectively.

Interestingly, while the BC algorithms proposed by Kawamura et al. and Bajard et. al convert only one value at steps 3 and 7 of Algorithm 4.3, the corresponding reorganized versions proposed by Gandino et al. include in the BC algorithm all other steps of Algorithm 4.3 as well. These savings are depicted in terms of reduced number of modular multiplications required for one RNSMMM, as shown in Table 4.1 [GLP⁺12].

Algorithm 4.6 Reorganized first BC algorithm for Kawamura et al. [GLMB11]

Input: $s_B, \alpha = 0, \hat{s}_A$

Output: \hat{c}_A

Precompute: $(B_A^{-1}A_j)_{\mathcal{A}}, (-NA_j)_{\mathcal{A}}, (B_iNB_A^{-1}A_j^{-1})_{\mathcal{A}} (\forall i = 1 \dots L), \langle -N^{-1}B_i^{-1} \rangle_{q_i} (\forall i = 1 \dots L)$

- 1 $\xi_i = \langle s_i \cdot (-N^{-1}B_i^{-1}) \rangle_{q_i}, \forall i = 1 \dots L$
 - 2 $\sigma = \alpha$
 - 3 $\hat{c}_j = \langle \hat{s}_j \cdot (B_A^{-1}A_j) \rangle_{p_j}, \forall j = 1 \dots L$
 - 4 **for** $i = 1 \dots L$ **do**
 - 5 $\sigma = \sigma + \text{trunc}(\xi_i)/2^r$
 - 6 $\gamma^* = \lfloor \sigma \rfloor$
 - 7 $\sigma = \sigma - \gamma^*$
 - 8 $\hat{c}_j = \langle \hat{c}_j + \xi_i \cdot (B_iNB_A^{-1}A_j^{-1}) + \gamma^* \cdot (-NA_j^{-1}) \rangle_{p_j}, \forall j = 1 \dots L$
 - 9 **end for**
-

Algorithm 4.7 Reorganized second BC algorithm for Kawamura et al. [GLMB11]

Input: $\hat{c}_A, \alpha = 0.5$

Output: c_B

Precompute: $\langle A_j \rangle_{\mathcal{B}}, (-A)_{\mathcal{B}}$

- 1 $\sigma = \alpha$
 - 2 $c_i = 0, \forall i = 1 \dots L$ in \mathcal{B}
 - 3 **for** $j = 1 \dots L$ **do**
 - 4 $\sigma = \sigma + \text{trunc}(\hat{c}_j)/2^r$
 - 5 $\gamma^* = \lfloor \sigma \rfloor$
 - 6 $\sigma = \sigma - \gamma^*$
 - 7 $c_i = \langle c_i + \hat{c}_j \cdot A_j + \gamma^* \cdot (-A) \rangle_{q_i}, \forall i = 1 \dots L$
 - 8 **end for**
-

Algorithm 4.8 Reorganized first BC algorithm for Bajard et al. [BI04]

Input: $s_{B \cup p_r} = (s_1, s_2, \dots, s_L, s_r), \hat{s}_A$

Output: \hat{c}_A, c in p_r

Precompute: $(B_A^{-1}A_j)_{\mathcal{A}}, \langle B_A^{-1} \rangle_{p_r}, \langle B_iNB_A^{-1} \rangle_{p_r} (\forall i = 1 \dots L), (B_iNB_A^{-1}A_j^{-1})_{\mathcal{A}} (\forall i = 1 \dots L), \langle -N^{-1}B_i^{-1} \rangle_{q_i} (\forall i = 1 \dots L)$

- 1 $q_i = \langle s_i \cdot (-N^{-1}B_i^{-1}) \rangle_{q_i}, \forall i = 1 \dots L$
 - 2 $\hat{c}_j = \langle \hat{s}_j \cdot B_A^{-1}A_j + \sum_{i=1}^L q_i \cdot (B_iNB_A^{-1}A_j^{-1}) \rangle_{p_j}, \forall j = 1 \dots L$
 - 3 $c_r = \langle s_r \cdot B_A^{-1} + \sum_{i=1}^L q_i \cdot (B_iNB_A^{-1}) \rangle_{p_r}$
-

Algorithm 4.9 Reorganized second BC algorithm for Bajard et al. [BI04]

Input: $\hat{c}_A = (c_1, c_2, \dots, c_L), \langle c \rangle_{p_r}$

Output: $c_B = (c'_1, c'_2, \dots, c'_L)$

Precompute: $\langle A_r^{-1} \rangle_{p_r}, (A_j)_B (\forall j = 1 \dots L), \langle A_j \rangle_{p_r} (\forall j = 1 \dots L), (-A)_B$

- 1 $c''_r = \langle \sum_{j=1}^L \hat{c}_j \cdot A_j \rangle_{p_r}$
 - 2 $\gamma = \langle (c''_r - c_r) A_r^{-1} \rangle_{p_r}$
 - 3 $c'_i = \langle \sum_{j=1}^L \hat{c}_j \cdot A_j \rangle_{q_i}, \forall i = 1 \dots L$
 - 4 $c'_i = \langle c'_i - \gamma A \rangle_{q_i}, \forall i = 1 \dots L$
-

Steps in RNSMMM	[KKSS00]	[BI04]	[GLMB11] applied in [KKSS00]	[GLMB11] applied [BI04]
1, 3, 4	$5L$	$5L$	$2L$	$2L$
First BC	$L^2 + 2L$	$L^2 + L$	$L^2 + 3L$	$L^2 + 2L$
Second BC	$L^2 + 2L$	$L^2 + 2L$	$L^2 + L$	$L^2 + L$
Total	$2L^2 + 9L$	$2L^2 + 8L$	$2L^2 + 6L$	$2L^2 + 5L$

Table 4.1: Number of modular multiplications in state-of-the-art RNSMMM

4.1.3.1 Modular reduction by the RNS moduli

The modular reduction technique by each RNS modulus is identical for all the works in [KKSS00, BI04, GLMB11], since not only it offers simple implementations but also allows for fair comparisons. Assuming moduli of the form $p_i = 2^r - c_i$, where $c_i < 2^h$ and $h < \frac{r-1}{2}$, the reduction of an integer $x < 2^{2r}$ requires two multiplications and three additions according to

$$y = x \bmod 2^r + ((x \ll r) \bmod 2^r) \cdot c_i + (x \ll 2r) \cdot c_i^2, \quad (4.6)$$

where \ll denotes a left-shift operation, $x < 2^r$, $z > 2r$, and $c_i < 2^h$ [GLMB11].

4.1.3.2 Conversions to/from RNS

To allow handling of large integers in each modulus channel, it is useful to employ high-radix representations so that each high-radix digit can be assigned to an RNS channel. A radix- 2^r representation of an integer x as a L -tuple $(x^{(L-1)}, \dots, x^{(0)})$ satisfies

$$x = \sum_{i=0}^{L-1} x^{(i)} 2^{ri} = (2^{r(L-1)}, \dots, 2^r, 1) \begin{bmatrix} x^{(L-1)} \\ \vdots \\ x^{(1)} \\ x^{(0)} \end{bmatrix}, \quad (4.7)$$

where $0 \leq x^{(i)} \leq 2^r - 1$. By applying the modulo p_j operation in (4.7) we can convert the integer x to its associated RNS representation by

$$\langle x \rangle_{p_j} = \left\langle \sum_{i=0}^{L-1} x^{(i)} \langle 2^{ri} \rangle_{p_j} \right\rangle_{p_j}, \forall j \in [1, L]. \quad (4.8)$$

If constants $\langle 2^{ri} \rangle_{p_j}$ are precomputed, this computation is a typical multiply-accumulate operation and can be computed in L steps, when executed by L units in parallel.

As (2.29) is the basis of the presented RNSMMM algorithms, it would be useful to employ it also for the RNS-to-decimal conversion. Let us rewrite (2.29) as

$$\begin{aligned} x &= \sum_{i=1}^L \langle x_i \cdot A_i^{-1} \rangle_{p_i} \cdot A_i - \gamma A = \\ &= (2^{r(L-1)}, \dots, 2^r, 1) \sum_{i=1}^L \left\{ \xi_i \cdot \begin{bmatrix} A_{i(L-1)} \\ \vdots \\ A_{i(1)} \\ A_{i(0)} \end{bmatrix} - \gamma \begin{bmatrix} A_{(L-1)} \\ \vdots \\ A_{(1)} \\ A_{(0)} \end{bmatrix} \right\}, \end{aligned} \quad (4.9)$$

where $\xi_i = \langle x_i \cdot A_i^{-1} \rangle_{p_i}$. As soon as γ has been evaluated using the methods of section 3, each row of (4.9) can be computed in parallel in each cell by means of multiply-accumulate operations. In this case, carry should be propagated from cell 1 until cell L [KKSS00].

4.1.4 Architectural comparisons

All works in [KKSS00, BI04, GLMB11] utilize cell-based architectures for implementing the algorithms in [KKSS00] and [BI04, GLMB11] respectively. Each cell corresponds to a single RNS modulus and utilizes a multiply-accumulate unit followed by a modular reduction unit which performs reduction by the corresponding RNS modulus using (4.6). Actually, with slight modifications, the architecture in [GLMB11] supports both algorithms in [KKSS00, BI04].

The cell structure is shown in Fig.4.1 [GLMB11]; a common bus that connects the cells and lines connecting one cell to a subsequent one are omitted, for simplicity reasons. The multiply-accumulate unit is depicted at the top of the cell and the modular reduction units at the bottom are a straightforward implementation of (4.6). Again, the prospective reader is instructed to refer to [GLP⁺12, GLMB11] for a detailed architectural analysis of the state-of-the-art RNSMMM algorithms.

Table 4.2 summarizes the number of clock cycles required for the considered algorithms. The metrics are based on the cell-based architecture in Figure 4.1 and depend on the pipeline stages ϵ of each cell, the number of cells L , and the number of parallel multipliers M in each cell.

Operation	Base	# multiplications	# cycles
$s = xy$	\mathcal{B}	L	ϵ
$\hat{\hat{s}} = \hat{x}\hat{y}$	\mathcal{A}	L	$\lfloor \frac{1}{\epsilon+M-1} \rfloor$
$q = s(-N^{-1}B_i^{-1})$	\mathcal{B}	L	ϵ
$\hat{w} = \hat{s}B_A^{-1}A_j$	\mathcal{A}	L	$\lfloor \frac{1}{\epsilon+M-1} \rfloor$
$\hat{w}_i = B_iNB_A^{-1}A_j^{-1}$	\mathcal{A}	L^2	$\lfloor \frac{L}{M} \rfloor - 1 + \epsilon$
$w = \hat{w}_jA_j$	\mathcal{B}	L^2	$\lfloor \frac{L}{M} \rfloor - 1 + \epsilon$

Table 4.2: Number of multiplication steps per RNS modular multiplication in state-of-the-art RNSMMM ([GLP⁺12] without BC correction)

Algorithm	BC	# steps	Step delay	MM delay	Expon. delay ($\times 2,050$)	Area ($\times 33$)
[KKSS00],[BI04]	[KKSS00]	88	93	8,184	8,184	99,873
[GLP ⁺ 12]	[KKSS00]	76	93	7,068	0.0034 + 7,068	99,873
[KKSS00, BI04]	[BI04]	89	86.6	7,707	7,707	99,840
[GLP ⁺ 12]	[BI04]	77	86.6	6,669	0.0034 + 6,669	99,840

Table 4.3: Area and delay comparisons with $L = 33$, $r = 32$, $\epsilon = 3$, $M = 1$, $h = 11$

Gate	Area (transistors)	Delay (inverter)
Inverter	2	1
NAND	4	1.4
XOR	4	1.4
XNOR	12	3.2
NAND3	8	1.8
NAND4	10	2.2
REGISTER	15	4.8

Table 4.4: Basic logic library in CMOS technology (model from [Gaj97])

Table 4.3 summarizes complexity comparisons based on the model in Table 4.4 [Gaj97]. Clearly, the approach in [GLP⁺12, GLMB11] further optimizes the algorithmic and hardware complexity of the considered RNSMMM algorithm. In the following sections, optimized versions of the considered RNSMMM algorithm based on a MRC approach for the base conversion operation are proposed.

4.2 New MRC-based Montgomery modular multiplication in $GF(p)$

In the following, the original MRC [ST67] is employed to construct new BC algorithms in the context of the RNSMMM in Algorithm 4.2 [SS11]. In the original case, the MRC of an integer x with an RNS representation $x_{\mathcal{A}} = (x_1, x_2, \dots, x_L)$ is given by

$$x = U_1 + W_2 U_2 + \dots + W_L U_L, \quad (4.10)$$

where $W_i = \prod_{j=2}^i m_{j-1}, \forall i \in [2, L]$ and the U_i s are computed according to

$$\begin{aligned} U_1 &= x_1 \\ U_2 &= \langle (x_2 - U_1) m_{1,2}^{-1} \rangle_{m_2} \\ U_3 &= \langle ((x_3 - U_1) m_{1,3}^{-1} - U_2) m_{2,3}^{-1} \rangle_{m_3} \\ &\vdots \\ U_L &= \langle (\dots (x_L - U_1) m_{1,L}^{-1} - \dots - U_{L-1}) m_{L-1,L}^{-1} \rangle_{m_L}, \end{aligned} \quad (4.11)$$

where $m_i m_{i,j}^{-1} \equiv 1 \pmod{m_j}$.

The proposed BC algorithm implements (4.11) in steps 2 to 7 to obtain the mixed-radix digits U_i of x . From step 8 to 15 (4.11) is realized by applying a Horner's rule scheme, while the whole summation is computed modulo each modulus p_i of the new base \mathcal{A} .

The proposed BC is error-free, as opposed to [KKSS00, BI04], where CRT is employed and an approximation of the correction factor γ of (2.29) is calculated. Conditions $\gcd(B, N) = 1$ and $\gcd(A, B) = 1$ are sufficient for the existence of $(N^{-1})_B$ and $(B^{-1})_A$, respectively. $4N \leq B$ is also sufficient for $c < 2N$ to hold when $a, b < 2N$. It holds that

$$c = \frac{v}{B} = \frac{ab + tN}{B} < \frac{(2N)^2 + BN}{B} = \left(\frac{4N}{B} + 1 \right) N \leq 2N, \quad (4.12)$$

which yields $4N \leq B$. Finally, (4.12) shows that $2N \leq A$ is sufficient for $c < A$ and $v < AB$. Since v is the maximum intermediate value, all values are less than AB .

4.2.1 The Proposed RNSMMM Architecture

Figure 4.2 depicts a suitable architecture that implements the proposed RNSMMM algorithm. Due to the algorithm's internal structure, all calculations are decomposed to simple

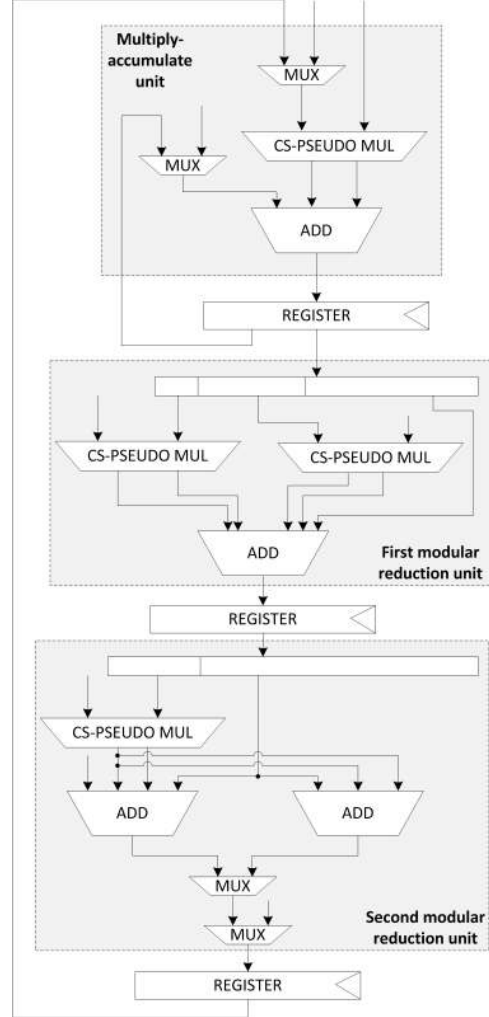


Figure 4.1: MAC cell [GLP⁺12]

Algorithm 4.10 Proposed MRC-based base conversion

Input: $x_B = (x_1, x_2, \dots, x_L)$

Output: $x_A = (x'_1, x'_2, \dots, x'_L)$

```

1   $U_1 \leftarrow x_1$ 
2  for all  $i = 2, \dots, L$  do
3     $U_i \leftarrow x_i$ 
4    for  $j = 1$  to  $i - 1$  do
5       $U_i \leftarrow \langle (U_i - U_j) q_{j,i}^{-1} \rangle_{q_i}$ 
6    end for
7  end for
8  for all  $i = 1, \dots, L$  do
9     $x'_i \leftarrow \langle U_L \rangle_{p_i}$ 
10   for  $j = L - 1$  to 1 do
11      $x'_i \leftarrow \langle x'_i q_j + U_j \rangle_{p_i}$ 
12   end for
13 end for

```

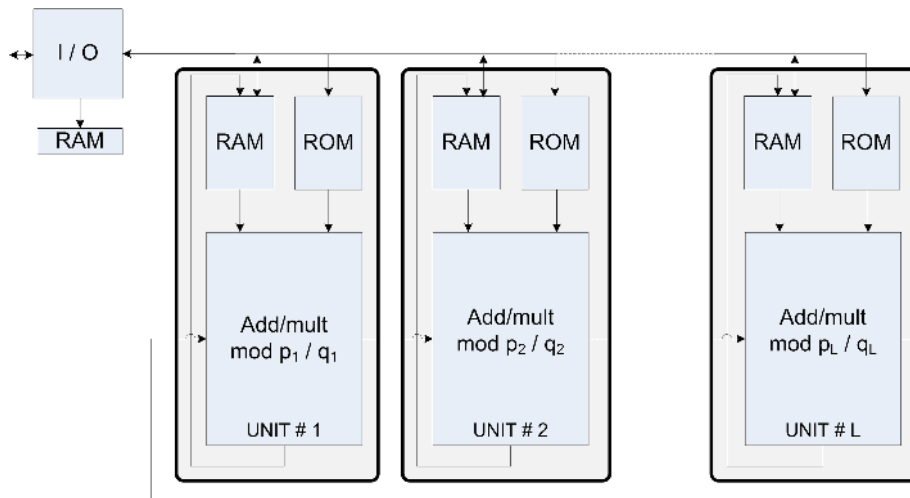


Figure 4.2: The proposed MRC-based RNSMMM architecture

add/multiply operations, each one dedicated to an RNS modulus channel. The lines that connect each unit are used for the base conversion and MRC realization, since according to (4.10), the outcome of a unit must be added to the outcome of its subsequent one. Obviously, the proposed architecture, if used repeatedly, executes modular exponentiation and inversion algorithms [MVO96] with no need for extra hardware.

The architecture preserves the efficient input/output conversions presented in the previous section for the binary-to-RNS and RNS-to-binary conversions respectively.

Table 4.5: ROM requirements of the proposed RNSMMM architecture

Operation	Parameters stored in ROM	ROM (bits)
Binary-to-RNS	$\langle 2^{rj} \rangle_{p_i}, \langle 2^{rj} \rangle_{q_i}$	$2Lr$
RNSMMM	N_B^{-1}, B_A^{-1}, N_A	$3Lr$
BC	$q_{j,i}^{-1}, p_{j,i}^{-1}, p_j, q_j$	$2Lr(\frac{L+1}{2})$
RNS-to-binary	–	–

Table 4.6: Number of operations in RNSMMM algorithms

	Alg. RNSMMM		Conversions	
	Alg. BC	Others	Binary-to-RNS	RNS-to-binary
MRC-based [SS11]	$L(L-1) + 2L^2$	$5L$	L^2	L^2
[KKSS00]	$2L^2 + 4L$	$5L$	L^2	$L(2L+1)$
[BI04]	$2L^2 + 3L$	$5L$	N/A	N/A
[GLP ⁺ 12]	$2L^2 + 3L$	$2L$	L^2	$L(2L+1)$

4.2.2 Performance and Comparisons

4.2.2.1 Memory requirements

Table 4.5 summarizes the memory requirements of the proposed architecture, where r is the radix, and therefore the bit-length, of each modulus. If area is not an issue, two RNSMMM architectures can be exploited in parallel, each one dedicated to a single base. Each RAM module in Figure 4.2 stores an r -bit result of a multiply-accumulate unit, which is equivalent to a total $2Lr$ -bit RAM, if two RNSMMM architectures operate in parallel. Note also that all parameters for the RNS-to-binary conversion are identical to the parameters used for the BC algorithm, thus no extra memory is required.

4.2.2.2 Frequency

Table 4.6 summarizes the number of operations required by the RNSMMM algorithm, along with the binary-to-RNS and RNS-to-binary conversions in terms of modular multiplications. Comparisons are made with the works presented in [KKSS00, BI04, GLP⁺12], which also present an RNS Montgomery multiplication, but they are based on CRT. It is apparent that due to the use of MRC instead of CRT, the proposed algorithm requires slightly more operations for the BC but less operations in total. The reason is that savings are achieved in the RNS-to-binary case, since the upper part (steps 2-7) of the BC algorithm is identical

for the conversion case as well. Thus, only L^2 multiplications are required for steps 8-13. In Chapter 5 the algorithm is further optimized to minimize the modular multiplications for the BC as well.

Note that in [BI04] no estimations are given for the converters, since an unusual splitting of the input to blocks of size equal to the size of the RNS moduli is issued, instead of standard RNS-to-binary conversion. However, this method produces an intermediate value that has to be corrected at the end of the algorithm, by adding several multiples of the modulus N to the final result.

Let Δ , f , T denote the total number of operations, frequency, and throughput of a modular exponentiation that utilizes RNSMMM multiplication and requires $\frac{3Lr}{2} + 2$ modular multiplications [KKSS00]. Δ is then roughly estimated by the sum of RNSMMM algorithm's operations multiplied with $\frac{3nr}{2} + 2$, plus the operations for the conversions and the final correction step. Under these assumptions, $T = f \cdot Lr / (\Delta/L)$ and Δ is divided by L , since L processing units work in parallel.

As an example, for a 1024-bit exponentiation used in RSA, if 33 32-bit moduli are chosen and the frequency is set to $f = 100$ MHz, then T is about 3 [Mbit/sec], which is a reasonable choice for deep sub-micron CMOS technologies, such as $0.35 - 0.18\mu m$. For the same scenario, the throughput reported in [KKSS00] was 890 [kbit/sec].

4.2.2.3 Area requirements

Each add/multiply unit encompasses an r -bit modular adder, consisting of 2 r -bit adders and a multiplexer, and an r -bit multiplier. Assuming A_{add} , A_{mult} and A_{mux} be the area of an r -bit adder, multiplier and multiplexer respectively, then the total area of the proposed architecture is roughly estimated to be $L(2A_{add} + A_{mult} + A_{mux})$.

4.3 New CRT-based Montgomery modular multiplication in $GF(2^n)$

Details on $GF(2^n)$ arithmetic were presented in Chapter 2, section 1.1.2, thus we focus on the applicability of PRNS in $GF(2^n)$ arithmetic. The authors in [KA98] proposed a Montgomery multiplication algorithm, presented as Algorithm 4.11, suitable for polynomials in $GF(2^n)$. Instead of computing the product $c(x) = a(x) \cdot b(x) \pmod{N(x)}$, the algorithm computes $c(x) = a(x) \cdot b(x) \cdot r^{-1}(x) \pmod{N(x)}$, where $r(x)$ is a special fixed element in $GF(2^n)$. The selection of $r(x) = x^n$ is the most appropriate, since modular reduction and division by x^n are simple shifts [BSS02, HVM04].

The Montgomery multiplication method requires that $r(x)$ and $N(x)$ are relatively prime, i.e., $\gcd\{r(x), N(x)\} = 1$. This assumption always holds, since $N(x)$ is an irreducible polynomial in $GF(2)$, thus it is not divisible by x . Since $r(x)$ and $N(x)$ are relatively prime, there exist two polynomials $r^{-1}(x)$ and $N^{-1}(x)$ such that

$$r(x) \cdot r^{-1}(x) + N(x) \cdot N^{-1}(x) = 1, \tag{4.13}$$

where $r^{-1}(x)$ is the inverse of $r(x)$ modulo $N(x)$. The polynomials $r^{-1}(x)$ and $N^{-1}(x)$ can be computed using the extended Euclidean algorithm [LN86, McE87]. The Montgomery multiplication of $a(x)$ and $b(x)$ is then defined as

$$c(x) = a(x) \cdot b(x) \cdot r^{-1}(x) \pmod{N(x)}, \quad (4.14)$$

which can be computed according to Algorithm 4.11.

Algorithm 4.11 Montgomery Multiplication in $GF(2^n)$

Input: $a(x), b(x), r(x), N(x), N^{-1}(x)$

Output: $c(x) = a(x)b(x)r^{-1}(x) \pmod{N(x)}$

- 1 $s(x) \leftarrow a(x)b(x)$
 - 2 $t(x) \leftarrow s(x)N^{-1}(x) \pmod{r(x)}$
 - 3 $u(x) \leftarrow t(x)N(x)$
 - 4 $v(x) \leftarrow s(x) + u(x)$
 - 5 $c(x) \leftarrow v(x)/r(x)$
-

The algorithm is similar to the Montgomery multiplication for integers presented already as Algorithm 4.1. The only difference is that the final subtraction step required in the integer case is not necessary in polynomials, as it has been proved that the degree of the resulting polynomial $c(x)$ is less than n [KA98].

4.3.1 The proposed PRNS Montgomery modular multiplication

A modification of the Montgomery algorithm for multiplication in $GF(2^n)$ that encompasses PRA is proposed. This work extends the work in [BIJ05], which encompasses trinomials for the modulus set, by employing general, any-degree polynomials that achieve larger dynamic ranges. Also, the problem of PRNS-to-polynomial and polynomial-to-PRNS conversions are addressed, as opposed to [BIJ05].

Similar to the integer case, two PRNS bases $\mathcal{A} = (p_1, p_2, \dots, p_L)$ and $\mathcal{B} = (q_1, q_2, \dots, q_L)$ are introduced, such that $\gcd\{p_i, q_j\} = 1, \forall i, j \in [1, L]$. The 5 steps of the Montgomery algorithm are translated to PRNS computations in both bases, denoted from now on as $\mathcal{T} = \mathcal{A} \cup \mathcal{B}$.

Initially, the input polynomials $a(x), b(x)$ are transformed to PRNS representation in both bases as $a_{\mathcal{T}}$ and $b_{\mathcal{T}}$. Steps 1,3, and 4 involve addition and multiplication operations, thus their transformation to PRNS is straightforward. For step 2, the constant $r(x)$ is replaced by

$B(x) = \prod_{i=1}^L q_i(x)$, which is the range of \mathcal{B} . Then, $t(x)$ in PRNS format is computed in base \mathcal{B}

by $t_{\mathcal{B}} = s_{\mathcal{B}} \cdot (N^{-1})_{\mathcal{B}}$. Nevertheless, the computations in base \mathcal{B} can't be continued for steps 3, 4 and 5, since in step 5 we would need to compute the quantity $B^{-1}(x) \pmod{B(x)}$, which does not exist. Thus, a base conversion from base \mathcal{B} to base \mathcal{A} is inserted to compute $t_{\mathcal{A}}$. $t_{\mathcal{A}}$ is then used to execute steps 3, 4 and 5 in base \mathcal{A} . The result at the end of the proposed algorithm is a quantity $c_{\mathcal{T}}$ in PRNS format such that $c(x) = a(x)b(x)B^{-1}(x) \pmod{N(x)}$.

Algorithm 4.12 depicts the proposed $GF(2^n)$ PRNS Montgomery Modular Multiplication (PRNSMMM). In the proposed algorithm, the degree of input and output polynomials are

both less than n , so that input and output are compatible with each other. This allows to construct a modular exponentiation algorithm by repetition of the PRNS Montgomery multiplication. Base extension in step 7 is utilized for the same reason. Note that it is unnecessary to realize step 5 in base \mathcal{B} . Also, base \mathcal{B} representation in step 4 can be ignored as well, since v is always a multiple of B and thus $v_{\mathcal{B}} = 0_{\mathcal{B}}$.

Algorithm 4.12 The proposed PRNSMMM algorithm

Input: $a_{\mathcal{T}}, b_{\mathcal{T}}, N_{\mathcal{B}}^{-1}, B_{\mathcal{A}}^{-1}, N_{\mathcal{A}}$, {such that $\deg\{a(x)\} < n$ and $\deg\{b(x)\} < n$ }

Output: $c_{\mathcal{T}}$, {such that $\deg\{c(x)\} < n$ and $c(x) = a(x)b(x)B^{-1}(x) \pmod{N(x)}$ }

- 1 $s_{\mathcal{T}} \leftarrow a_{\mathcal{T}} \cdot b_{\mathcal{T}}$
 - 2 $t_{\mathcal{B}} \leftarrow s_{\mathcal{B}} \cdot N_{\mathcal{B}}^{-1}$
 - 3 $t_{\mathcal{A}} \leftarrow t_{\mathcal{B}}$
 - 4 $u_{\mathcal{A}} \leftarrow t_{\mathcal{A}} \cdot N_{\mathcal{A}}$
 - 5 $v_{\mathcal{A}} \leftarrow s_{\mathcal{A}} + u_{\mathcal{A}}$
 - 6 $c_{\mathcal{A}} \leftarrow v_{\mathcal{A}} \cdot B_{\mathcal{A}}^{-1}$
 - 7 $c_{\mathcal{B}} \leftarrow c_{\mathcal{A}}$
-

Algorithm 4.13 Base Conversion (BC) algorithm for PRNSMMM

Input: $z_{\mathcal{B}}$

Output: $z_{\mathcal{A}}$

Precompute: $B_i^{-1}(x), \langle B_i(x) \rangle_{p_j}, \{ \forall i, j \in [1, L] \}$

- 1 $\xi_i = \langle z_i \cdot B_i^{-1}(x) \rangle_{q_i}, \{ \forall i \in [1, L] \}$
 - 2 $z_{0,i} = 0$
 - 3 **for** $i = 1, \dots, L$ **do**
 - 4 $z_{j,i} = z_{j-1,i} + \xi_i \cdot \langle B_i(x) \rangle_{p_j}, \{ \forall j \in [1, L] \}$
 - 5 **end for**
 - 6 $z_{\mathcal{A}} = \langle z_{j,L} \rangle_{p_j}, \{ \forall j \in [1, L] \}$
-

4.3.2 Base Conversion (BC) algorithm for PRNSMMM

Assuming a base conversion from base \mathcal{B} to base \mathcal{A} , one can calculate (2.35) to transform a PRNS vector expressed in \mathcal{B} to polynomial representation and then perform a Polynomial-to-PRNS conversion to base \mathcal{A} to obtain the PRNS representation in the new base. The process is shown in Algorithm 4.13.

4.3.2.1 Proof of PRNSMMM's algorithm validity

Theorem 1: If (1) $\gcd\{N, B\} = 1$, (2) $\gcd\{A, B\} = 1$, (3) $\deg\{A\} > n$, and (4) $\deg\{B\} > n$, then Algorithm 4.12 outputs $c_{\mathcal{T}}$, for which $c = abB^{-1} \pmod{N}$ and $\deg\{c\} < n$.

Proof. Since $\gcd\{N, B\} = 1$ and $\gcd\{A, B\} = 1$, N is relatively prime to B and B is relatively prime to A . Thus, the quantities $\langle N^{-1} \rangle_{q_i}$ and $\langle B^{-1} \rangle_{p_i}$ exist $\forall i \in [1, L]$, and therefore $(N^{-1})_{\mathcal{B}}$ and $(B^{-1})_{\mathcal{A}}$ exist.

Assume that the polynomial v is a multiple of B , i.e., $v \bmod B = 0$. Then, $s + t \cdot N = 0 \bmod B$, which means that $t = s \cdot N^{-1} \bmod B$. This corresponds to step 2 of the PRNSMMM algorithm, which means that step 6 is error-free since base conversion in step 3 is error-free, therefore PRNSMMM holds.

Furthermore, it must be proved that the resulting polynomial c of Algorithm 4.12 is a polynomial of degree less than n . It holds that

$$\begin{aligned}
 \deg\{c\} &\leq \deg\left\{\frac{v}{B}\right\} \Rightarrow \\
 \deg\{c\} &\leq \deg\left\{\frac{ab + tN}{B}\right\} \Rightarrow \\
 \deg\{c\} &\leq \deg\{ab + tN\} - \deg\{B\} \Rightarrow \\
 \deg\{c\} &\leq \max\{\deg\{ab\}, \deg\{tN\}\} - \deg\{B\} \Rightarrow \\
 \deg\{c\} &\leq \deg\{tN\} - \deg\{B\} \Rightarrow \\
 \deg\{c\} &\leq \deg\{B\} - 1 + n - \deg\{B\} \Rightarrow \\
 \deg\{c\} &\leq n - 1.
 \end{aligned} \tag{4.15}$$

Since v is the maximum intermediate value of Algorithm 4.12, its degree must be less than the degree of the polynomial AB . Under this assumption, we get

$$\begin{aligned}
 \deg\{v\} &< \deg\{AB\} \Rightarrow \\
 \deg\{cB\} &< \deg\{AB\} \Rightarrow \\
 \deg\{c\} + \deg\{B\} &< \deg\{A\} + \deg\{B\} \Rightarrow \\
 \deg\{c\} &< \deg\{A\}.
 \end{aligned} \tag{4.16}$$

From (4.15) and (4.16) we have

$$\left. \begin{aligned}
 \deg\{c\} &< n \\
 \deg\{c\} &< \deg\{A\}
 \end{aligned} \right\} \Rightarrow \deg\{A\} > n. \tag{4.17}$$

Finally, note that 4.16 is independent of $\deg\{B\}$, thus selecting $\deg\{B\} > n$ is sufficient. \square

4.3.3 The proposed PRNSMMM architecture

Fig. 4.3 depicts a suitable architecture that implements the proposed PRNSMMM algorithm. Due to the algorithm's internal structure, all calculations are decomposed to simple multiply-accumulate operations. This allows the use of identical parallel multiply-accumulate units, each one dedicated to a PRNS modulus. The lines that connect each unit are used for the base conversion and CRT realization, since the outcome of a unit must be added to the outcome of its subsequent unit. Note the resemblance to the corresponding architecture in Figure 4.2 for the integers case. Obviously, the proposed architecture, if used repeatedly, executes the exponentiation algorithm [BSS02, H MV04] with no need for extra hardware.

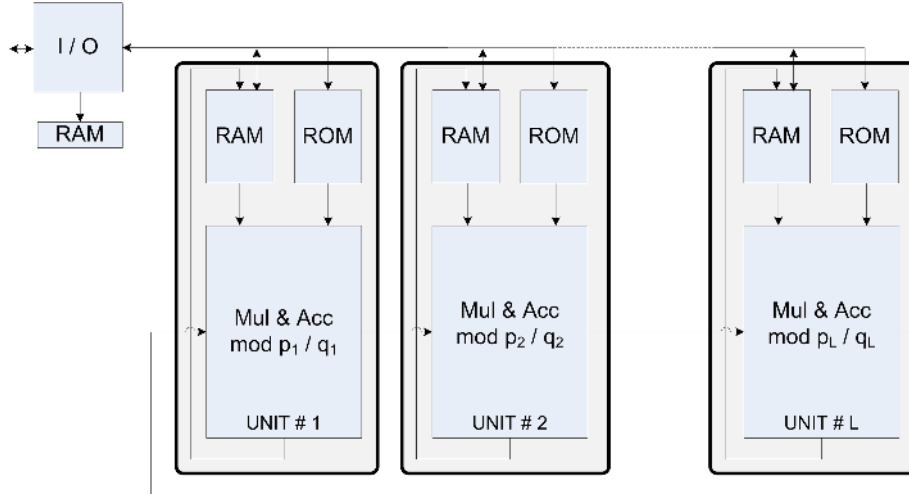


Figure 4.3: The proposed PRNSMMM architecture

4.3.3.1 Polynomial-to-PRNS conversion

Any polynomial $z(x) \in GF(2^n)$ can be written in the form

$$z(x) = (z_{(L-1)}, \dots, z_{(1)}, z_{(0)}) = \sum_{i=0}^{L-1} z_{(i)} x^{ri} \quad (4.18)$$

considering r -bit word split. Based on (4.18), a method must be devised that matches the proposed architecture and converts any polynomial $z(x) \in GF(2^n)$ to PRNS format. By applying the $\text{mod } p_j$ operation in (4.18) we get

$$z_{\mathcal{A}} = \left\langle \sum_{i=0}^{L-1} z_{(i)} x^{ri} \right\rangle_{p_j} = \left\langle \sum_{i=0}^{L-1} z_{(i)} \langle x^{ri} \rangle_{p_j} \right\rangle_{p_j}, \forall j \in [1, L]. \quad (4.19)$$

If constants $\langle x^{ri} \rangle_{p_j}$ are precomputed, this computation is well-suited to the proposed architecture and can be computed in L steps, when executed by L units in parallel.

4.3.3.2 PRNS-to-Polynomial conversion

As (2.35) is the basis of the proposed PRNSMMM algorithm, it would be useful to employ it also for the PRNS-to-Polynomial conversion. Considering the polynomial representation $A_{i(j)}, j \in [0, r(L-1) - 1]$ of $A_i(x)$, the PRNS-to-polynomial conversion can be computed as

$$z(x) = (x^{L-1}, \dots, x^1, 1) \sum_{i=1}^L \left\{ \delta_i \cdot \begin{bmatrix} A_{i(L-1)} \\ \vdots \\ A_{i(1)} \\ A_{i(0)} \end{bmatrix} \right\}, \quad (4.20)$$

where $\delta_i = \langle z_i \cdot A_i^{-1}(x) \rangle_{p_i}$. Each row of (4.20) can be computed in parallel by using the proposed architecture.

Table 4.7: ROM requirements of the proposed PRNSMMM architecture

Operation	Parameters stored in ROM	ROM (bits)
Polynomial-to-PRNS	$x_{p_i}^{r_i}, x_{q_i}^{r_i}$	$L(r + g)$
PRNSMMM	$(N^{-1})_{\mathcal{B}}, (B^{-1})_{\mathcal{A}}, N_{\mathcal{A}}$	$L(2r + g)$
BC	$\langle B_i^{-1}(x) \rangle_{q_i}, B_i(x)_{\mathcal{A}},$ $\langle A_i^{-1}(x) \rangle_{p_i}, A_i(x)_{\mathcal{B}}$	$(r + g)(L^2 + L)$
PRNS-to-Polynomial	$A_i(x)$	$L^2 r$

Table 4.8: Number of modular multiplications in PRNSMMM algorithm

	Alg. PRNSMMM		Conversions	
	Alg. BC	Others	Polynomial-to-PRNS	PRNS-to-Polynomial
# of ops	$2L^2 + 4L$	$5L$	$2L^2$	$L^2 + L$

4.3.4 Performance

4.3.4.1 Memory requirements

Table 4.7 summarizes the memory requirements of the proposed architecture, in terms of L , r , and g . Recall that, in the general case, base \mathcal{A} utilizes r -bit moduli, while base \mathcal{B} consists of g -bit moduli. It is always possible to select $g \geq r$, so that g -bit units can perform r -bit calculations as well. In this case, the proposed architecture supports both bases with the use of time sharing. If area is not an issue, two PRNSMMM architectures can be exploited in parallel, each one dedicated to a single base. Each RAM module in Fig.1 stores an r - or g -bit result of a multiply-accumulate unit, which is equivalent to a total $L(r + g)$ -bit RAM, if two PRNSMMM architectures operate in parallel.

4.3.4.2 Frequency

Table 4.8 summarizes the number of operations, in terms of modular multiplications, required by the PRNSMMM algorithm, along with the Polynomial-to-PRNS and PRNS-to-Polynomial conversions.

Let Δ , f , T denote the total number of operations, frequency, and throughput of an ECC implementation in $GF(2^n)$. In this case, approximately $\theta = 26 \lceil n/2 \rceil$ modular multiplications have to be performed [BSS02, HMV04]. Δ is then roughly estimated by the sum of all operations for the PRNSMMM algorithm multiplied by θ , plus the number of operations for the conversions. The throughput is then estimated by $T = f \cdot Lr / (\Delta/L)$. Δ is divided by L , since L processing units work in parallel.

As an example, for a $GF(2^{39})$ multiplication used in ECC, if 8, 32-bit moduli are chosen and the frequency is set to $f = 100$ MHz, then T is about 620 [kbit/sec], which is a reasonable choice for deep sub-micron CMOS technologies, such as $0.35 - 0.18\mu m$.

4.3.4.3 Area requirements

In the worst case, each unit performs in parallel operations for both bases thus, the accumulate part encompasses a $GF(2^r)$ and a $GF(2^g)$ adder, consisting of $(r + g)$ XOR gates. The multiply part consists of a $GF(2^r)$ and a $GF(2^g)$ multiplier. For the multiplier, several alternatives exist in the literature [DBS06]. Considering for example a MSB-multiplier, $2(r + g)$ XOR gates and $2(r + g)$ AND gates are required per unit plus some control logic [DBS06]. This results to a total number of $L [3(r + g) \text{ XOR} + 2(r + g) \text{ AND}]$ gates for an L -unit, PRNSMMM architecture.

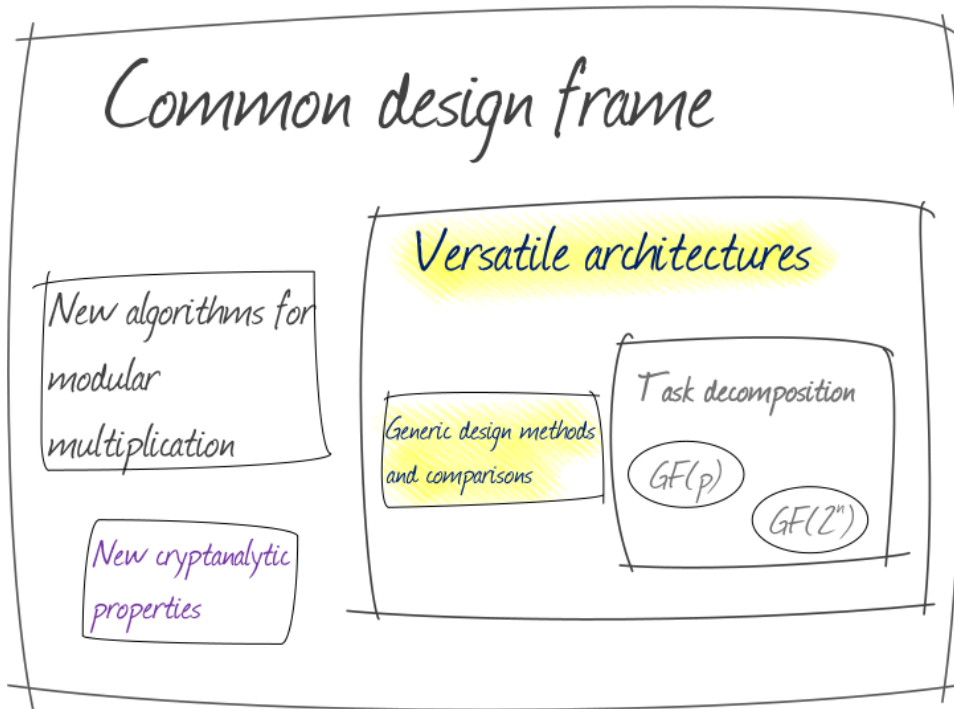
4.4 Summary

In this chapter, an overview of state-of-the-art RNS Montgomery multiplication algorithms was presented, along with algorithmic and architectural comparisons. Following, new algorithms for modular multiplication that combine Montgomery multiplication and RNS-PRNS for $GF(p)$ and $GF(2^n)$ arithmetic were proposed.

Especially for $GF(2^n)$, a methodology for incorporating PRA in the Montgomery multiplication algorithm for polynomials in $GF(2^n)$ was presented. The mathematical conditions that need to be satisfied, in order for this incorporation to be valid were examined and performance results were given in terms of the field characteristic n , the number of moduli elements L , and the moduli word-length r, g .

Both applications of residue arithmetic are flexible due to the decomposition of all internal calculations to simple add/multiply operations. The architectures perform binary-to-RNS/PRNS and RNS/PRNS-to-binary conversion, RNS/PRNS Montgomery multiplication, as well as MRC for integers and CRT for polynomials in the same hardware, thus making them suitable for a variety of cryptographic applications. A methodology to unify these algorithms into a common dual-field frame, VLSI designs, further optimizations and detailed performance estimations are examined in the next chapter.

Novel versatile architectures



This chapter presents the design methodology for incorporating RNS and PRNS in MMM in $GF(p)$ or $GF(2^n)$ respectively. An analysis of input/output conversions to/from residue representation, along with the proposed residue Montgomery multiplication algorithm, reveals common multiply-accumulate data paths both between the converters and between the two residue representations. A versatile architecture is derived that supports all operations of Modular Multiplication (MM) in $GF(p)$ and $GF(2^n)$, input/output conversions, MRC for integers and polynomials, dual-field modular exponentiation and inversion in the same hardware. Detailed comparisons with state-of-the-art implementations prove the potential of residue arithmetic exploitation in dual-field modular multiplication.

5.1 Decomposition of operations

In Chapter 4 two algorithms for RNS-PRNS Montgomery multiplication based on MRC and CRT (RNSMMM and PRNSMMM respectively), were presented. Both algorithms maintain multiply-accumulate characteristics and share common logic and algorithmic parts. In this chapter the RNSMMM algorithm is further optimized, the PRNSMMM algorithm is transformed to an MRC-based structure, and the two algorithms are unified to a common Dual-field Residue Arithmetic Montgomery Modular Multiplication (DRAMMM) algorithm.

5.1.1 Optimizing RNSMMM

The MRC-based algorithm that avoids the evaluation of the γ factor of (2.29) forms the basis of the proposed RNS-based Montgomery multiplication algorithm. The derived algorithm is identical to Algorithm 4.2, however the BC algorithm is now based on the modified version of MRC shown in (2.30) and (2.32). Comparing the previous approach employing (2.31), which requires $L \frac{L-1}{2}$ modular multiplications, the optimized MRC requires only $L-2$ modular multiplications. The methodology is further extended for the case of $GF(2^n)$.

Algorithm 5.1 depicts the proposed base conversion process that converts an integer ζ expressed in RNS base \mathcal{B} as $\zeta_{\mathcal{B}}$ to the RNS representation of another base \mathcal{A} . As will be shown in next sections, Algorithm 5.1 offers better opportunities for parallelization of operations. It implements 2.32 in steps 1-8 to obtain the mixed-radix digits U_i of ζ . In steps 9-15, 2.30 is realized, while the whole summation is computed modulo each modulus p_i of the new base \mathcal{A} .

Algorithm 5.1 The proposed optimized base conversion for the RNSMMM

Input: $\zeta_{\mathcal{B}} = (\zeta_{\mathcal{B}_1}, \zeta_{\mathcal{B}_2}, \dots, \zeta_{\mathcal{B}_L}), \mathcal{A}, \mathcal{B}$

Output: $\zeta_{\mathcal{A}} = (\zeta_{\mathcal{A}_1}, \zeta_{\mathcal{A}_2}, \dots, \zeta_{\mathcal{A}_L})$

```

1   $W_1 \leftarrow 0$ 
2   $U_1 \leftarrow \zeta_{\mathcal{B}_1}$ 
3  for all  $i = 2, \dots, L$  do
4       $U_i \leftarrow \zeta_{\mathcal{B}_i} - \zeta_{\mathcal{B}_1}$ 
5      for  $j = 1$  to  $i - 1$  do
6           $U_i \leftarrow \langle U_i - W_j U_j \rangle_{q_i}$ 
7      end for
8  end for
9   $\zeta_{\mathcal{A}_1} \leftarrow 0$ 
10 for all  $i = 1, \dots, L$  do
11     for  $j = 1$  to  $L$  do
12          $\kappa_{ij} = \langle W_j A_j \rangle_{p_i}$ 
13          $\zeta_{\mathcal{A}_i} \leftarrow \langle \kappa_{ij} + \zeta_{\mathcal{A}_i} \rangle_{p_i}$ 
14     end for
15 end for

```

The two base conversions in the RNSMMM algorithm are error-free, contrasted to other algorithms that employ CRT and utilize approximation methods to compute the correction

Algorithm 5.2 PRNSMMM in $GF(2^n)$

Input: $a_{\mathcal{T}}, b_{\mathcal{T}}, N_{\mathcal{B}}^{-1}, B_{\mathcal{A}}^{-1}, N_{\mathcal{A}}, \{\deg\{a\} < n, \deg\{b\} < n\}$

Output: $c_{\mathcal{T}}, \{\deg\{c\} < n, c = abB^{-1} \pmod{N}\}$

- 1 $s_{\mathcal{T}} \leftarrow a_{\mathcal{T}} \cdot b_{\mathcal{T}}$
 - 2 $t_{\mathcal{B}} \leftarrow s_{\mathcal{B}} \cdot N_{\mathcal{B}}^{-1}$
 - 3 $t_{\mathcal{A}} \leftarrow t_{\mathcal{B}}$
 - 4 $u_{\mathcal{A}} \leftarrow t_{\mathcal{A}} \cdot N_{\mathcal{A}}$
 - 5 $v_{\mathcal{A}} \leftarrow s_{\mathcal{A}} + u_{\mathcal{A}}$
 - 6 $c_{\mathcal{A}} \leftarrow v_{\mathcal{A}} \cdot B_{\mathcal{A}}^{-1}$
 - 7 $c_{\mathcal{B}} \leftarrow c_{\mathcal{A}}$
-

factor γ of 2.29 [KKSS00, BI04, GLP⁺12, GLMB11]. Conditions $\gcd(B, N) = 1$ and $\gcd(A, B) = 1$ are sufficient for the existence of $(-N^{-1})_{\mathcal{B}}$ and $B_{\mathcal{A}}^{-1}$, respectively. As it holds that

$$c = \frac{v}{B} = \frac{ab + tN}{B} < \frac{(2N)^2 + BN}{B} = \left(\frac{4N}{B} + 1\right)N \leq 2N, \quad (5.1)$$

it follows that $4N \leq B$ is sufficient for $c < 2N$ to hold when $a, b < 2N$. Finally, (5.1) also shows that $2N \leq A$ is sufficient for $c < B$ and $v < AB$. Since v is the maximum intermediate value, all values are less than AB [KKSS00, SS11].

5.1.2 Embedding PRNS in $GF(2^n)$ Montgomery Multiplication

A modification of the Montgomery algorithm for multiplication in $GF(2^n)$ that encompasses PRNS is proposed next. The proposed algorithm employs general polynomials of any degree, and is an extension of an algorithm [BIJ05], which employs trinomials for the PRNS modulus set. Additionally, the proposed algorithm addresses the problem of converting data to/from PRNS representation. In contrast to a similar algorithm in [SSS12], which employed CRT for polynomials for the BC algorithm, the proposed architecture employs MRC. This allows for dual-field RNS/PRNS implementation, which is not supported in [SSS12], and a new methodology to implement RNS-to-binary conversion as will be shown in the following subsections.

The proposed algorithm for $GF(2^n)$ PRNSMMM is presented below as Algorithm 5.2. The corresponding algorithm for base conversion in $GF(2^n)$ is identical to Algorithm 5.1, depicted below as 5.3. The only difference is that integer additions/subtractions and multiplications are replaced by polynomial ones. Again, the degree of input and output polynomials are both less than n , which allows the construction of a modular exponentiation algorithm by repetition of the PRNSMMM. Base conversion in step 7 is employed for the same reason. For the proof of PRNSMMM algorithm's validity see Section 4.3.

Algorithm 5.3 $GF(2^n)$ base conversion algorithm for PRNSMMM

Input: $\zeta_{\mathcal{B}} = (\zeta_{\mathcal{B}_1}, \zeta_{\mathcal{B}_2}, \dots, \zeta_{\mathcal{B}_L}), \mathcal{A}, \mathcal{B}$

Output: $\zeta_{\mathcal{A}} = (\zeta_{\mathcal{A}_1}, \zeta_{\mathcal{A}_2}, \dots, \zeta_{\mathcal{A}_L})$

Precompute: $\langle W_i(x) \rangle_{p_j}, \{ \forall i, j \in [1, L] \}$

```

1  $W_1 \leftarrow 0$ 
2  $U_1 \leftarrow \zeta_{\mathcal{B}_1}$ 
3 for all  $i = 2, \dots, L$  do
4    $U_i \leftarrow \zeta_{\mathcal{B}_i} + \zeta_{\mathcal{B}_1}$ 
5   for  $j = 1$  to  $i - 1$  do
6      $U_i \leftarrow \langle U_i + W_j U_j \rangle_{q_i}$ 
7   end for
8 end for
9  $\zeta_{\mathcal{A}_1} \leftarrow 0$ 
10 for all  $i = 1, \dots, L$  do
11   for  $j = 1$  to  $L$  do
12      $\kappa_{ij} = \langle W_j A_j \rangle_{p_i}$ 
13      $\zeta_{\mathcal{A}_i} \leftarrow \langle \kappa_{ij} + \zeta_{\mathcal{A}_i} \rangle_{p_i}$ 
14   end for
15 end for

```

5.1.3 The Proposed Versatile Architectures

A careful examination of RNSMMM and PRNSMMM algorithms reveals potential for unification into a common DRAMMM algorithm and a common Dual-field Base Conversion (DBC) algorithm. The unified algorithms are depicted below as Algorithms 5.4 and 5.5, where \oplus represents a dual-field addition/subtraction and \odot represents a dual-field multiplication.

Algorithm 5.4 The proposed DRAMMM algorithm

Input: $a_{\mathcal{T}}, b_{\mathcal{T}}, (-N^{-1})_{\mathcal{B}}, B_{\mathcal{A}}^{-1}, N_{\mathcal{A}}, \{a, b < 2N\}$

Output: $c_{\mathcal{T}}, \{c < 2N \text{ and } c \equiv abB^{-1} \pmod{N}\}$

```

1  $s_{\mathcal{T}} \leftarrow a_{\mathcal{T}} \odot b_{\mathcal{T}}$ 
2  $t_{\mathcal{B}} \leftarrow s_{\mathcal{B}} \odot (-N^{-1})_{\mathcal{B}}$ 
3  $t_{\mathcal{A}} \leftarrow t_{\mathcal{B}}$  { base conversion step }
4  $u_{\mathcal{A}} \leftarrow t_{\mathcal{A}} \odot N_{\mathcal{A}}$ 
5  $v_{\mathcal{A}} \leftarrow s_{\mathcal{A}} \oplus u_{\mathcal{A}}$ 
6  $c_{\mathcal{A}} \leftarrow v_{\mathcal{A}} \odot B_{\mathcal{A}}^{-1}$ 
7  $c_{\mathcal{B}} \leftarrow c_{\mathcal{A}}$  { base conversion step }

```

An important aspect is that all operations within the DRAMMM and the DBC algorithms are now decomposed into simple MAC operations of word-length equal to the modulus word length r . This allows for a fully-parallel hardware implementation, employing parallel MAC units, each dedicated to a modulus of the RNS/PRNS base.

Finally, the conditions from (4.16) and (4.17), for a valid RNS/PRNS transformation of the

Algorithm 5.5 DBC algorithm

Input: $\zeta_{\mathcal{B}} = (\zeta_{\mathcal{B}_1}, \zeta_{\mathcal{B}_2}, \dots, \zeta_{\mathcal{B}_L}), \mathcal{A}, \mathcal{B}$
Output: $\zeta_{\mathcal{A}} = (\zeta_{\mathcal{A}_1}, \zeta_{\mathcal{A}_2}, \dots, \zeta_{\mathcal{A}_L})$

```

1   $W_1 \leftarrow 0$ 
2   $U_1 \leftarrow \zeta_{\mathcal{B}_1}$ 
3  for all  $i = 2, \dots, L$  do
4       $U_i \leftarrow \zeta_{\mathcal{B}_i} \oplus \zeta_{\mathcal{B}_1}$ 
5      for  $j = 1$  to  $i - 1$  do
6           $U_i \leftarrow \langle U_i \oplus W_j \odot U_j \rangle_{q_i}$ 
7      end for
8  end for
9   $\zeta_{\mathcal{A}_1} \leftarrow 0$ 
10 for all  $i = 1, \dots, L$  do
11     for  $j = 1$  to  $L$  do
12          $\kappa_{ij} = \langle W_j \odot U_j \rangle_{p_i}$ 
13          $\zeta_{\mathcal{A}_i} \leftarrow \langle \kappa_{ij} \oplus \zeta_{\mathcal{A}_i} \rangle_{p_i}$ 
14     end for
15 end for
    
```

Montgomery algorithm yield

$$\left. \begin{array}{l} \deg\{A\} > n \\ \deg\{B\} > n \\ A \geq 2N \\ B \geq 4N \end{array} \right\}, \quad (5.2)$$

which means that one should select RNS/PRNS ranges of word length

$$\begin{array}{l} \delta_{\mathcal{A}} \geq \max\{\lceil \log 2N \rceil, n\} \\ \delta_{\mathcal{B}} \geq \max\{\lceil \log 4N \rceil, n\} \end{array} \quad (5.3)$$

for the bases \mathcal{A} and \mathcal{B} , respectively. Algorithms 5.4 and 5.5 along with conditions (5.2) and (5.3) form the complete framework for a dual-field residue arithmetic Montgomery multiplication.

As described before, the structure of the proposed algorithm allows it to be reused in the context of any exponentiation algorithm. A possible implementation is depicted in Algorithm 5.6, requiring in total $2n + 2$ DRAMMM multiplications [LN86, McE87]. Using Fermat's little theorem, field inversion can be realized by field exponentiation as described in [HMV04], thus it can be efficiently mapped to the proposed architecture as well without extra hardware.

Algorithm 5.6 Proposed DRAMMM modular exponentiation

Input: $z_T, e = (e_{n-1} \dots e_1 e_0)_2$

Output: $b_T, b \equiv \langle z^e \rangle_N$

```

1  $b \leftarrow 1$ 
2 for  $i = n - 1, \dots, 0$  do
3    $b \leftarrow \text{DRAMMM}(b, b)$ 
4   if  $e_i = 1$  then
5      $b \leftarrow \text{DRAMMM}(b, z)$ 
6   end if
7 end for
8 return  $b$ 

```

5.1.4 Input-Output (IO) Conversions

In the following discussion, base $\mathcal{A} = (p_1, p_2, \dots, p_L)$ shall be used as an example to analyze the conversions to/from residue representations, without loss of generality.

5.1.4.1 Binary-to-Residue conversion

A radix- 2^r representation of an integer z as an L -tuple $(z^{(L-1)}, \dots, z^{(0)})$ satisfies

$$z = \sum_{i=0}^{L-1} z^{(i)} 2^{ri} = (2^{r(L-1)}, \dots, 2^r, 1) \begin{bmatrix} z^{(L-1)} \\ \vdots \\ z^{(1)} \\ z^{(0)} \end{bmatrix}, \quad (5.4)$$

where $0 \leq z^{(i)} \leq 2^r - 1$. A method to compute $z_{\mathcal{A}}$ must be devised, that matches the proposed DRAMMM's multiply-accumulate structure. By applying the modulo p_j operation in (5.4) we obtain

$$\langle z \rangle_{p_j} = \left\langle \sum_{i=0}^{L-1} z^{(i)} \langle 2^{ri} \rangle_{p_j} \right\rangle_{p_j}, \quad \forall j \in [1, L]. \quad (5.5)$$

If constants $\langle 2^{ri} \rangle_{p_j}$ are precomputed, this computation is well-suited to the proposed MAC structure and can be computed in L steps, when executed by L units in parallel.

Similar to the integer case, a polynomial $z(x) \in GF(2^n)$ can be written as

$$z = \sum_{i=0}^{L-1} z^{(i)} x^{ri} = (x^{r(L-1)}, \dots, x^r, 1) \begin{bmatrix} z^{(L-1)} \\ \vdots \\ z^{(1)} \\ z^{(0)} \end{bmatrix}. \quad (5.6)$$

Applying the modulo p_j operation in 5.6 it yields

$$\langle z \rangle_{p_j} = \left\langle \sum_{i=0}^{L-1} z^{(i)} \langle x^{ri} \rangle_{p_j} \right\rangle_{p_j}, \quad \forall j \in [1, L] \quad (5.7)$$

which is a similar operation to 5.5, if $\langle x^{r^i} \rangle_{p_j}$ are precomputed.

From 5.5 and 5.7, it is deduced that conversions in both fields can be unified into a common conversion method, if dual-field circuitry is employed, as already mentioned for the case of the DRAMMM and DBC. In the rest of this thesis, the radix vectors $(2^{r(L-1)}, \dots, 2^r, 1)$ and $(x^{r(L-1)}, \dots, x^r, 1)$ of both fields shall be denoted as a common radix vector V , without loss of generality.

5.1.4.2 Residue-to-Binary Conversion

As all operands in (2.32) are of word length r , they can be efficiently handled by an r -bit MAC unit. However, (2.30) employs multiplications with large values, namely the W_i s.

To overcome this problem, (2.30) can be rewritten in matrix notation, as in (5.8), which implies a fully parallel implementation of the conversion process. The inner products of a row i are calculated in parallel in each MAC unit. Each MAC then propagates its result to subsequent MAC, so that at the end the last MAC(L) outputs the radix- 2^r digit $z^{(i)}$ of the result. In parallel with this summation, inner products of the next row $i + 1$ can be formulated, since the adder and multiplier of the proposed MAC architecture may operate in parallel.

$$\begin{aligned}
 z &= \left\{ U_1 \odot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus U_2 \odot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ W_2^{(1)} \\ W_2^{(0)} \end{bmatrix} \oplus U_3 \odot \begin{bmatrix} 0 \\ \vdots \\ W_3^{(2)} \\ W_3^{(1)} \\ W_3^{(0)} \end{bmatrix} \oplus \dots \oplus U_L \odot \begin{bmatrix} W_L^{(L-1)} \\ \vdots \\ W_L^{(2)} \\ W_L^{(1)} \\ W_L^{(0)} \end{bmatrix} \right\} \odot V \\
 &= \begin{bmatrix} 0 \oplus 0 \oplus 0 \oplus \dots \oplus U_L \odot W_L^{(L-1)} \\ \vdots \oplus \vdots \oplus \vdots \oplus \vdots \oplus \vdots \\ 0 \oplus 0 \oplus U_3 \odot W_3^{(2)} \oplus \dots \oplus U_L \odot W_L^{(2)} \\ 0 \oplus U_2 \odot W_2^{(1)} \oplus U_3 \odot W_3^{(1)} \oplus \dots \oplus U_L \odot W_L^{(1)} \\ U_1 \oplus U_2 \odot W_2^{(0)} \oplus U_3 \odot W_3^{(0)} \oplus \dots \oplus U_L \odot W_L^{(0)} \end{bmatrix} \odot V = \begin{bmatrix} z^{(L-1)} \\ \vdots \\ z^{(2)} \\ z^{(1)} \\ z^{(0)} \end{bmatrix} \odot V \quad (5.8)
 \end{aligned}$$

5.2 Versatile architectures - hardware design

5.2.1 Dual-Field Addition/Subtraction

A Dual-field Full Adder (DFA) cell is basically a Full Adder (FA) cell equipped with a field-select signal ($fsel$), that controls the operation mode [STK00]. When $fsel = 0$, the carry output is forced to 0 and the sum outputs the XOR operation of the inputs. As already mentioned, this is equivalent to the addition operation in $GF(2^n)$. When $fsel = 1$, $GF(p)$ mode is selected and the cell operates as a normal FA cell. Obviously, dual-field adders in various configurations (carry-propagate, carry-skip, etc) can be mechanized by utilizing

DFA cells. In the proposed implementation, 3-level, CLA with 4-bit Carry Lookahead Generator (CLG) groups are employed [EL04]. An example of a 4-bit dual-field CLA adder is shown in Figure 5.2. The GAP modules generate the signals $p_i = x_i \text{ XOR } y_i$, $g_i = x_i \text{ AND } y_i$, $\alpha_i = x_i \text{ OR } y_i$, and AND gates along with a f_{sel} signal control whether to eliminate carries or not. The carry-lookahead generator is an $AND - OR$ network based on (5.9) [EL04].

$$c_{i+1} = \text{OR}_{j=0}^i \left(\text{AND}_{k=j+1}^i \alpha_k \right) g_j \text{ OR } \left(\text{AND}_{k=0}^i \alpha_k \right) c_0 \quad (5.9)$$

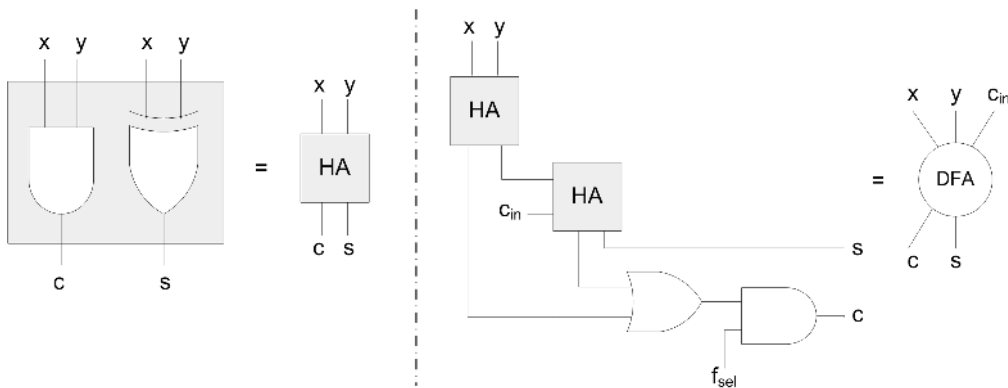


Figure 5.1: Dual-field full-adder cell

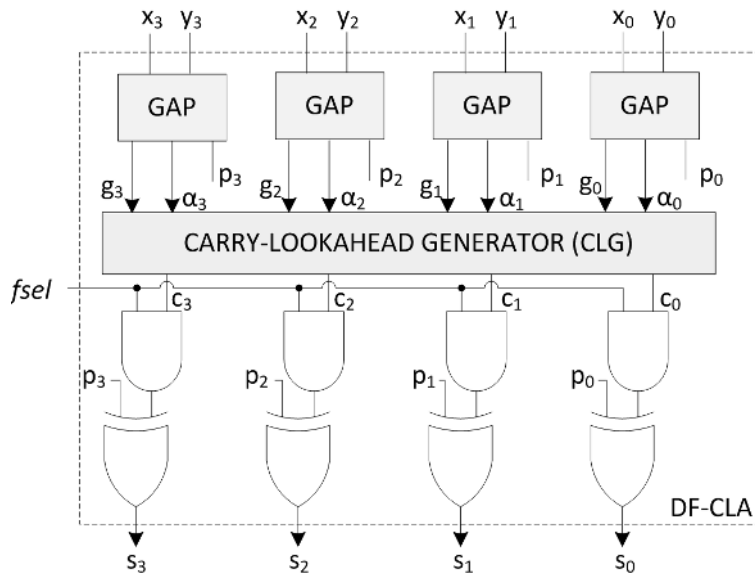


Figure 5.2: Dual-field CLA

5.2.1.1 Dual-Field Modular/Normal Addition/Subtraction

With trivial modifications of algorithms in 2.1 and 2.2 for modular addition/subtraction in $GF(p)$ [DBS06, LN86], a Dual-field Modular Adder-Subtractor (DMAS) shown in Figure 5.3 can be mechanized using CLA adders. There, c_r s represent the carry-out signals of each adder at position r , while the boolean function $\overline{add/sub} \cdot (c_1 \vee c_2) \vee add/sub \cdot \overline{c_1}$ controls

whether addition or subtraction is performed when in $GF(p)$ mode. When $fsel = 0$, the circuit is in $GF(2^n)$ mode and the output is derived directly from the top adder which performs a $GF(2^n)$ addition. When $fsel = 1$, the circuit may operate either as a normal $(2r + \log_2 L)$ -bit adder/subtractor ($conv_mode=0$) or as a modular adder/subtractor ($conv_mode=1$). In the first case, the output is the concatenation of the outputs of the two adders. This is required during residue-to-binary conversion, since (5.8) dictates that L , $(2r)$ -bit quantities need to be added recursively via a normal adder.

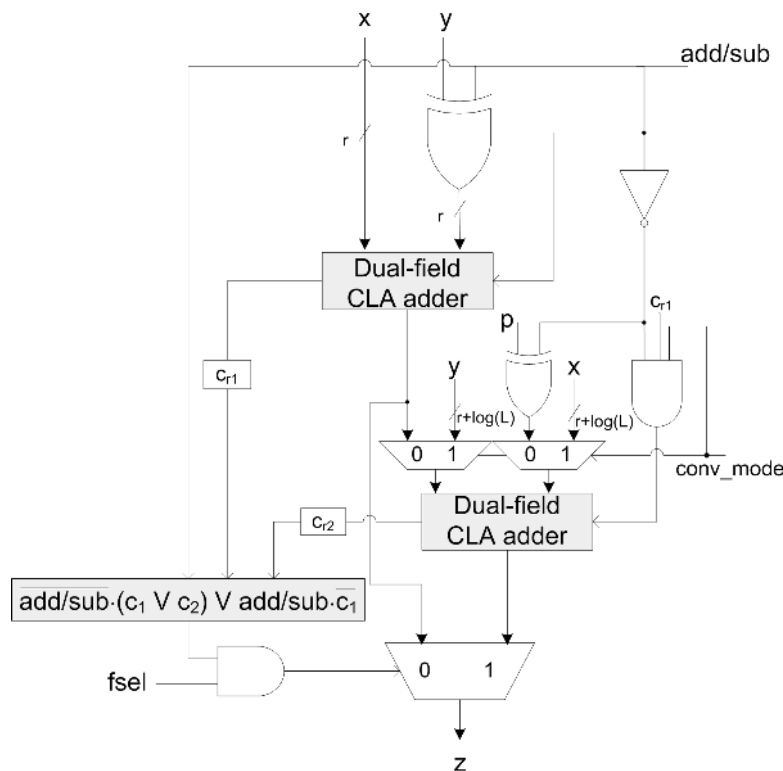


Figure 5.3: Dual-field modular/normal adder/subtractor (DMAS)

5.2.2 Dual-Field Multiplication

A parallel tree multiplier, which is suitable for high-speed arithmetic and requires little modification to support both fields, is considered in the proposed architecture. Regarding input operands, either integers or polynomials, partial product generation is common for both fields, i.e., an *AND* operation among all operand bits. Consequently, the addition tree that sums the partial products must support both formats. In $GF(2^n)$ mode, if DFA cells are used, all carries are eliminated and only *XOR* operations are performed among partial products. In $GF(p)$ mode, the multiplier acts as a conventional tree multiplier. A 4×4 -bit example of the proposed Dual-field modular Multiplier (DM) with output in carry-save format is depicted in Figure 5.4.

5.2.3 Dual-Field Modular Reduction

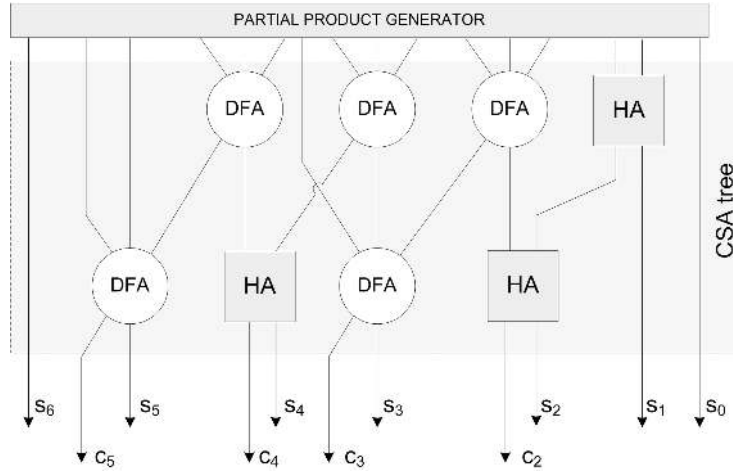


Figure 5.4: Dual-field multiplier (DM)

A final modular reduction by each RNS/PRNS modulus is required, for each multiplication outcome, within each MAC unit. From several modular reduction strategies [DBS06], a method based on careful modulus selection is utilized, since, not only it offers efficient implementations but also provides the best unification potential at a low area penalty.

Assume a $2r$ -bit product c that needs to be reduced modulo an integer modulus p_i . By selecting p_i of the form $2^r - \mu_i$, where the h -bit $\mu_i \ll 2^r$, the modular reduction process can be simplified as

$$\begin{aligned} \langle c \rangle_{p_i} &= \left\langle \sum_{i=0}^{r-1} c_i 2^i + 2^r \sum_{i=0}^{r-1} c_{r+i} 2^i \right\rangle_{p_i} = \left\langle E + 2^r F \right\rangle_{p_i} \\ &= \left\langle \sum_{i=0}^{r-1} d_i 2^i + \mu_i \sum_{i=0}^h d_{r+i} 2^i \right\rangle_{p_i}. \end{aligned} \quad (5.10)$$

From (5.10), it is apparent that

$$\langle c \rangle_{p_i} = \begin{cases} \sum_{i=0}^{r-1} \xi_i 2^i, & \xi < 2^r - \mu \\ \sum_{i=0}^r \xi_i 2^i + \mu_i, & 2^r - \mu < \xi < 2^r. \end{cases} \quad (5.11)$$

The same decomposition can be applied to polynomials and consequently, if dual-field adders and dual-field multipliers are employed, a Dual-field Modular Reduction (DMR) unit can be mechanized as shown in Figure

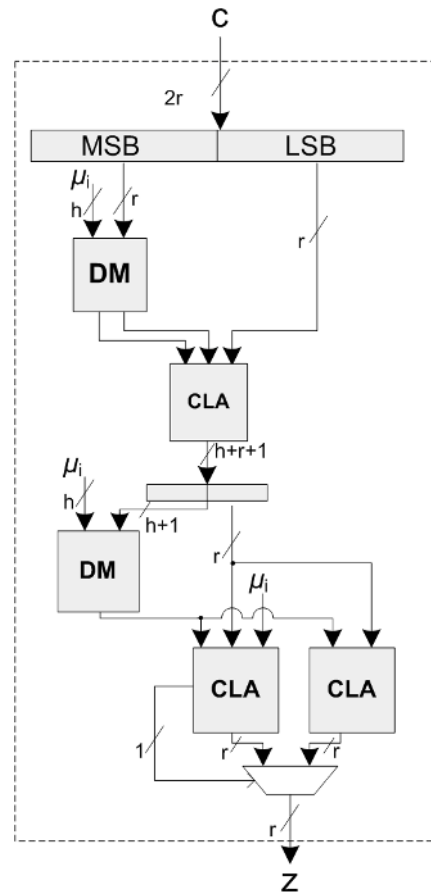


Figure 5.5: Dual-field modular reduction unit (DMR)

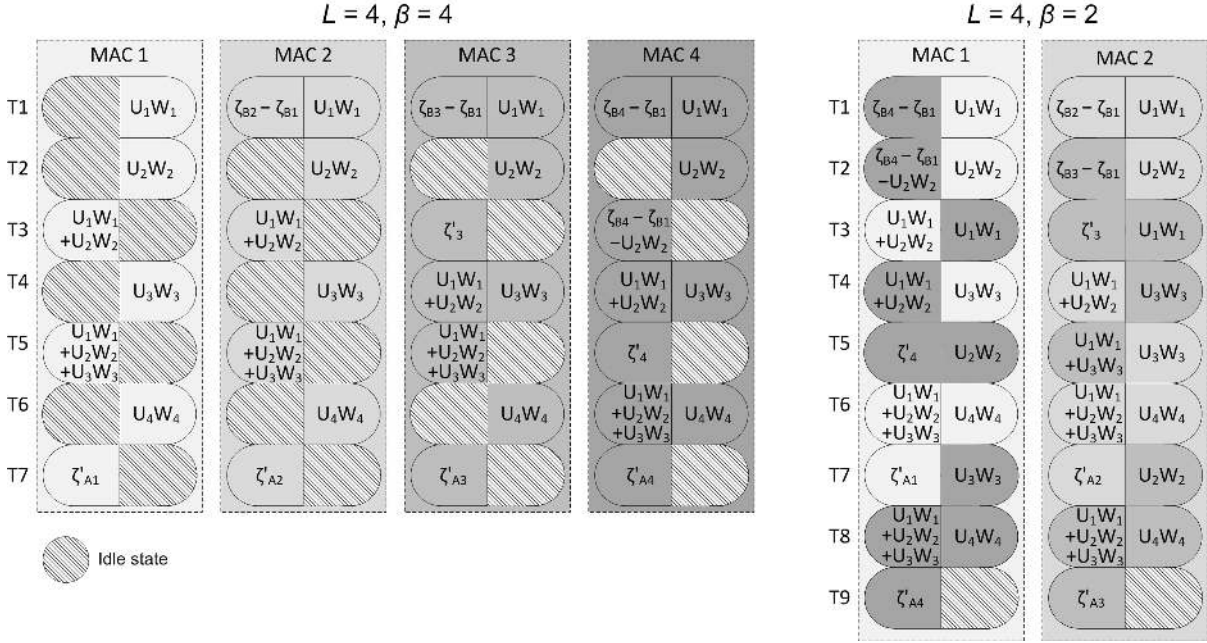


Figure 5.6: Task distribution in the proposed DRAMMM

5.5. The word length h of μ_i can be limited to a maximum of 10 bits for a base with 66 elements [KKSS00].

5.2.4 MAC Unit

The circuit organization of the proposed MAC unit is shown in Figure 5.7. Its operation is analyzed below in three steps, corresponding to the three phases of the calculations it handles, i.e., binary-to-residue conversion, RNS/PRNS Montgomery multiplication, and residue-to-binary conversion.

5.2.4.1 Binary-to-residue conversion

Initially, r -bit words of the input operands, as implied by (5.5), are cascaded to each MAC unit and stored in RAM1 at the top of Figure 5.7. These words serve as the first input to the multiplier, along with the quantities $\langle 2^{r_i} \rangle_{p_i, q_i}$ or $\langle x^{r_i} \rangle_{p_i, q_i}$, which are stored in a ROM. Their multiplication produces the inner products of (5.5) or (5.7) which are added recursively in the DMAS unit. The result is stored via the bus in RAM1. The process is repeated for the second operand and the result is stored in RAM2, so that when the conversion is finished, each MAC unit holds the residue digits of the two operands in the two RAM. The conversion requires L steps to be executed.

Table 5.1: Normalized area and delay of the proposed DRAMMM architecture

Module	Area	Critical path delay
DFA	$A_{FA} + A_{AND}$	$T_{FA} + T_{AND}$
DF-CLA ¹	$3r A_{DFA}$	$2\log_4(r) T_{DFA}$
DMAS ⁷	$2A_{DF-CLA} + (3r + \log_2 L) A_{mux(2-1)}$	$2\log_4(r + \log_2(L) + 1) T_{DFA} + T_{mux(2-1)}$
DM ⁸	$(r - 2)A_{DFA} + r^2 A_{AND} + 2A_{DF-CLA}$	$T_{DF-CLA} + (\log r) T_{DFA} + T_{AND}$
DMR	$A_{DM}^2 + A_{DM}^3 + A_{DF-CLA}^4 + A_{DF-CLA}^5 + A_{DF-CLA}^6 + (h + r + 1)A_{FF} + r A_{mux(2-1)}$	$(\log h + 6\log_4(h + r + 1)) T_{DFA}$
MAC	$A_{DM} + A_{DMAS} + A_{DMR} + 2r A_{mux(2-1)} + (2r) A_{FF}$	$T_{DM} + T_{DMR}$
DRAMMM	$L A_{MAC}$	T_{MAC}

¹ r -bit DF-CLA ² $(h \times r)$ -bit ³ $(h \times (h + 1))$ -bit ⁴ 3-input $(h + r + 1)$ -bit DF-CLA
⁵ 3-input r -bit DF-CLA ⁶ 2-input r -bit DF-CLA ⁷ $(r + \log L)$ -bit adders
⁸ $(r \times r)$ -bit

5.2.4.2 Montgomery multiplication

The first step of the proposed DRAMMM is a modular multiplication of the residue digits of the operands. Since these digits are immediately available by the two RAMs, a modular multiplication is executed and the result in R_1 is stored in RAM1 for base \mathcal{B} and RAM2 for base \mathcal{A} . Step 2 of DRAMMM is a multiplication of the previous result with a constant provided by the ROM. The results correspond to $\zeta_{\mathcal{B}_i}$ inputs of the DBC algorithm and are stored again in RAM1. All MAC units are updated through the bus with the corresponding RNS digits of all other MACs and a DBC process is initiated.

To illustrate the DBC process, a task distribution graph is presented in Figure 5.6 for a DRAMMM requiring $L = 4$ moduli. Two cases are represented; the first corresponds to a fully parallel architecture with $\beta = 4$ units and the second shows how the tasks can be overlapped when only $\beta = 2$ MAC units are available. Each MAC unit has been assigned to a different color, thus in the overlapped case the color codes signify when a MAC unit performs operations for other units. In the example of Figure 5.6, MAC(1) handles MAC(4) and MAC(2) handles MAC(3).

In each cycle, modular additions and multiplications are performed in parallel in each MAC. To depict this, each cycle is split in two parts: the operations on the left correspond to modular additions and on the right to modular multiplications. The results obtained by

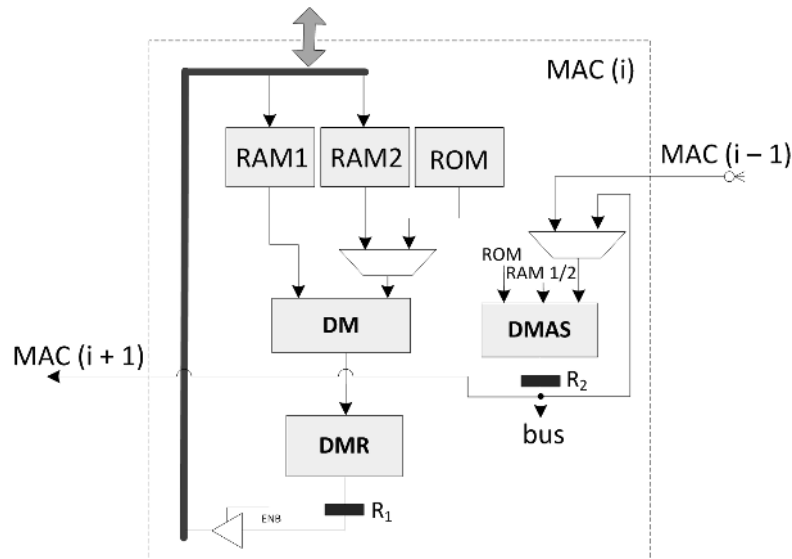


Figure 5.7: The proposed MAC unit

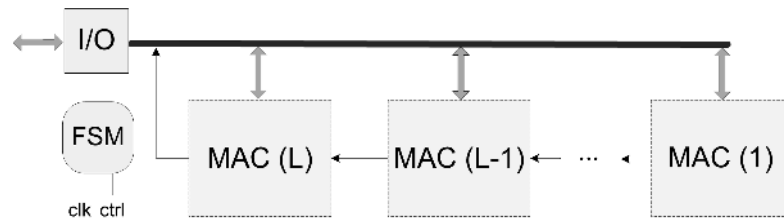


Figure 5.8: The proposed DRAMMM architecture

each operation are depicted in each cycle (they correspond to Alg.5.5), while idle states are denoted by dashed lines. An analysis on the number of clock cycles required, and how MAC units can be efficiently paired is presented in the next section.

The remaining multiplications, additions, and the final base conversion operation required by the DRAMMM algorithm are computed in the same multiply-accumulate manner and the final residue Montgomery product can be either driven to the I/O interface, or it can be reused by the MAC units to convert the result to binary format.

5.2.4.3 Residue-to-binary conversion

Residue-to-binary conversion is essentially a repetition of the DBC algorithm, except for steps 9-14, which are no longer modulo operations. Instead, (5.8) has been developed to map efficiently the conversion process to the proposed architecture. An important observation is that, whenever the preceding operation of a residue-to-binary conversion is a DRAMMM, which ends with a DBC execution, time savings are achieved since the upper part of the DBC algorithm (steps 1-8) is common with the conversion. Thus, the intermediate results from steps 1-8 can be stored until DBC is finished and then reused to implement (5.8).

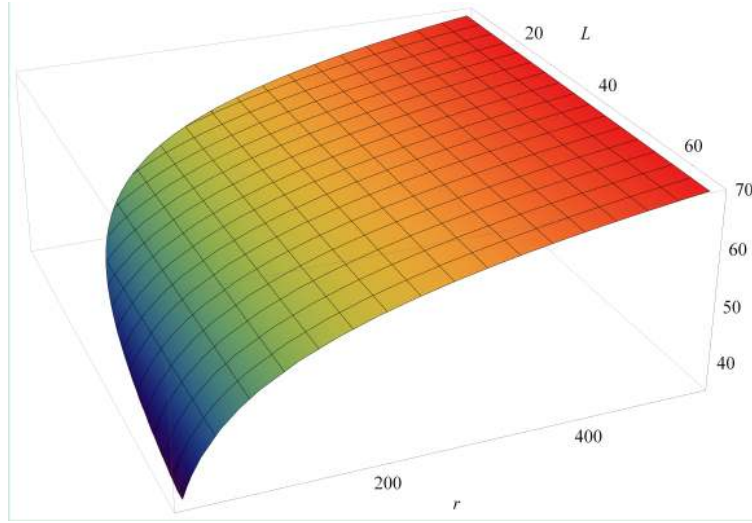


Figure 5.9: Normalized time complexity function $f(r, L)$

To illustrate the conversion process assume the generation of the inner products in row 1 of (5.8). Each product is calculated in parallel in each MAC unit and a “*carry-propagation*” from MAC(1) to MAC(L) is performed to add all inner products. When summation finishes the first digit $z^{(0)}$ of the result is produced in MAC(L). In parallel with this “*carry-propagation*”, the inner products of line 2 are calculated. As soon as a MAC unit completes an addition of carry-propagated inner products for line 1, a new addition for line 2 is performed. The process continues for all lines of (5.8) and the result is available after L steps. The complete DRAMMM architecture is depicted in Figure 5.8.

5.3 Performance results

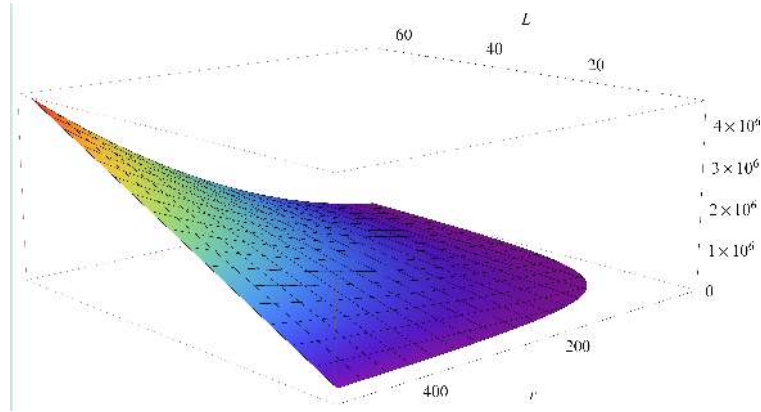
5.3.1 Area and Delay Estimations

Table 5.1 summarizes the area and delay complexity of the proposed architecture, in terms of the parameters L, r, h . In general, delay and area of a cell G shall be denoted with T_G and A_G respectively.

Regarding the area of a r -bit, dual-field CLA adder, with 4-bit CLG modules and three CLG levels it holds that [EL04]

$$A_{DF-CLA} = \frac{r-1}{4-1} \overbrace{(10A_{OR} + 20A_{AND})}^{4\text{-bit CLG}} + rA_{FA}. \quad (5.12)$$

Based on (5.12) it is easy to show that an r -bit CLA is approximately 3 times larger than an r -bit DFA, which explains the factor 3 regarding the area of the dual-field CLA in Table 5.1 [EL04].

Figure 5.10: Normalized area complexity function $g(r, L)$

5.3.1.1 Number of clock cycles

Assume that $\beta \leq L$ parallel MAC units are utilized in a real implementation. To simplify the discussion it is assumed that β is a multiple of L , i.e., $L = k\beta$. This means that each one of the first MAC(i), $i = 1, 2, \dots, \beta$ will provide results for $k - 1$ more channels. By construction of the MRC, the DBC process in Figure 5.6 requires $2L - 1$ clock cycles in the full parallel case. Each channel $1 \leq i \leq \beta$ requires L cycles for multiplications and $L + i - 2$ cycles for additions, thus each channel has $L - 1$ free slots for multiplications and $L - i + 1$ free slots for additions (idle states in Figure 5.6).

Let us assume for simplicity that $k = 3$. The free slots in each MAC(i) will accommodate operations from one MAC(j), $j = \beta + 1, \dots, 2\beta$ and one MAC(l), $l = 2\beta + 1, \dots, 3\beta$. Since each MAC(j) requires L multiplications and $L + j - 2$ additions and each MAC(l) requires L cycles for multiplications and $L + l - 2$ cycles for additions, then the cycles required to accommodate the results are $2L + (j + l) - 4$. The free slots in each MAC(i) are $L - i + 1$ thus the extra cycles to produce all results in each MAC(i) are $2L + (j + l) - 4 - (L - i + 1) = L + (i + j + l) - 5$. The problem transposes to finding the best combinations for (i, j, l) so that the quantity $L + (i + j + l) - 5$ minimizes. The problem can be described in terms of the following pseudo-code:

```

1  for all  $i = 1, \dots, \beta$  do
2    for  $j = \beta + 1, \dots, 2\beta$  do
3       $\alpha_{ijl} \leftarrow L + (i + j + l) - 5 \{\forall l \in [2\beta + 1, 3\beta]\}$ 
4    end for
5  end for
6  find  $v = \max(\alpha_{ijl})$  common  $\forall i$ 
7  for all  $i = 1, \dots, \beta$  do
8    match  $i$  with  $j$  and  $l$  such that  $\alpha_{ijl} \leq v$ 
9  end for

```

The pseudo-code calculates all possible values of extra cycles for all combinations of (i, j, l) and in step 6 we select v as the maximum common value for all MAC(i). For every dis-

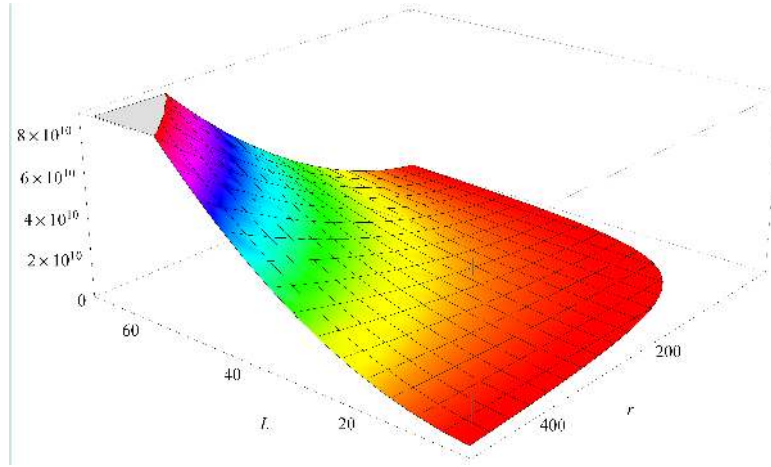


Figure 5.11: Area × time product function $\sigma(r, L)$

Table 5.2: Parameters of the proposed DRAMMM stored in ROM

Operation	Parameters stored in ROM	ROM (bits)
Binary-to-residue	$\langle 2^{ri} \rangle_{p_j, q_j}, \langle x^{ri} \rangle_{p_j, q_j}$	$4L^2r$
DRAMMM	$-p_B^{-1}, Q_A^{-1}, p_A$	$6Lr$
DBC	$\langle W_j \rangle_{p_i, q_i}$	$2L(L-1)r$
Residue-to-binary	W_i	$L(L-1)r$

tinct combination (i, j, l) that satisfies $\alpha_{ijl} < v$ we match the corresponding units until all distinct pairs of units in positions (j, l) are assigned to a distinct unit in position i . The remaining 6 steps of the DRAMMM require $6k$ cycles in total.

5.3.1.2 Memory Requirements

Table 5.2 summarizes the ROM requirements of the proposed DRAMMM architecture. As far as RAM is concerned, the worst case occurs during DBC and input/output conversions. This amounts to a $(2L-1)r$ -bit RAM1/2 per MAC unit, thus in total a $L(4L-2)r$ -bit RAM is required by the proposed DRAMMM architecture.

5.3.2 Comparisons with RNS implementations

The proposed architecture introduces the concept of dual-field RNS Montgomery multiplication, which is not supported by existing RNS solutions [KKSS00, BI04, NMSK01, PP95, Gui10, TjZbXHQj10, BDK98, GLP⁺12]. The architecture in [PP95] requires additional bits in the mantissa of the arithmetic units employed, in order to accurately compute the final

Table 5.3: Number of modular multiplications in the DRAMMM algorithm

	Alg. DRAMMM		Conversions	
	Alg. DBC	Others	Input	Output
This work	$2L(L+1) - 4$	$5L$	L^2	$\frac{L(L-1)}{2}$
[KKSS00, NMSK01]	$2L(L+2)$	$5L$	L^2	$L(2L+1)$
[BI04]	$2L(L + \frac{3}{2})$	$5L$	N/A	N/A
[Gui10]	$2L(L+2)$	$5L$	L^2	$L(2L+1)$
[GLP ⁺ 12]	$2L(L + \frac{3}{2})$	$2L$	L^2	$L(2L+1)$

modular product. Although the authors in [PP95] do not provide accurate metrics of the hardware complexity, the bit precision of the proposed MAC unit is equal to the modulus word-length, implying a simpler VLSI structure and less hardware. In [BI04], no hardware details are discussed, while the proposed algorithm is impractical in real cryptosystem implementations. Data sent from one party to another are in RNS format which, as a concept, poses limitations on the hardware architecture of the communicating parties.

Compared to the most efficient and practical RNS implementations in [KKSS00, NMSK01, Gui10, GLP⁺12], the proposed architecture further reduces the number of modular multiplications for the base conversion and the RNS-to-binary conversion, as depicted in Table 5.3. Other algorithms that employ MRC also perform worse. For example, the work in [TjZbXHQj10] requires $2L^2 + 5L$ modular multiplications while the work in [BDK98] is a predecessor of [BI04], which also performs worse, as shown in Table 5.3. This is also due to the simplified version of MRC employed in this work that requires $L - 2$ multiplication to implement (2.32), while [BDK98] requires $L \frac{L-1}{2}$ multiplications for the same conversion to implement (2.31) [BDK98, TjZbXHQj10].

5.3.3 Complexity comparisons with non-RNS implementations

None of the RNS implementations presented in [KKSS00, BI04, NMSK01, PP95, Gui10] provided comparisons with non-RNS solutions. In the following, a systematic, technology-independent, complexity comparison between RNS and non-RNS architectures is attempted for the first time, plus a function-based approach for the calculation of the optimal values for the parameters L, r is developed. As common constants for all comparisons, the TSMC 0.13 μm library of standard cells in Table 5.7 is employed [SCWL08].

The references in Tables 5.4 and 5.5, refer to both scalable [SL10, TK03, HKA⁺05, STK00, PH08, HGEG11] and non-scalable architectures [MMM04, SKS07, SCWL08]. Regarding the scalable implementations which encompass multiple Processing Elements (PEs), if n -bit input operands are employed, then $\epsilon = \lceil n/r \rceil$ or $\epsilon = \lceil (n+1)/r \rceil$ (if carry-save format is used or not accordingly), θ is the number of PEs, and $\lambda = \lceil n/\theta \rceil$.

Table 5.4: Normalized time and area complexity comparisons in $GF(p)$

Work	Cycles	Critical path delay	Time complexity	Area	Area complexity
This work	see section 5.3.1.1	T_{MAC}	$f(r, L)^8$	$L A_{MAC}$	$g(r, L)^{11}$
[MMM04] ¹	$n + 1$	$3T_{FA} + 2T_{XOR} + T_{AND}$	4.11	not reported	$22.68n^{10}$
[MMM04] ²	$n + 2$	$2T_{FA} + T_{mux(4-1)} + 2T_{XOR} + T_{AND}$	3.76	not reported	$27.12n^{10}$
[SCWL08]	$n + 4$	$2T_{FA} + T_{mux(2-1)} + T_{AND}$	2.94	$10nA_{FF} + 2nA_{FA} + 3nA_{AND} + nA_{mux(2-1)} + nA_{mux(3-1)}$	$12.66n$
[SL10] ³	$\lambda\theta + \epsilon + 3$	$2T_{AND} + 2T_{FA} + T_{mux(2-1)}$	3.39	$2r\theta A_{AND} + (2r\theta + 1)A_{FA} + (r - 1)A_{HA} + [(4r + 4)\theta + 4r + 1]A_{FF}$	$1.33 + 4.17r + \theta(3.6 + 5.98r)$
[SL10] ⁷	$\lambda(\epsilon + 1) + \theta + 2$				
[TK03] ⁴	$2\lambda\theta + \epsilon - 1$	$2T_{AND} + 2T_{FA} + T_{mux(2-1)}$	3.39	$2r\theta(A_{AND} + A_{FA}) + \theta(8r + 2)A_{FF}$	$\theta(1.8 + 9.58r)$
[TK03] ⁷	$\lambda(\epsilon + 1) + 2(\theta - 1)$				
[HKA ⁺ 05] ⁵	$\lambda(\theta + \lceil\theta/r\rceil) + \epsilon - 1$	$2T_{AND} + 2T_{FA} + T_{mux(2-1)}$	3.39	$2r\theta(A_{AND} + A_{FA}) + \theta(4r + 6)A_{FF}$	$\theta(5.4 + 5.98r)$
[HKA ⁺ 05] ⁷	$\lambda(\epsilon + \lceil\theta/r\rceil + 1) + \theta - 1$				
[STK00] ⁴	$2\lambda\theta + \epsilon - 1$	$3T_{AND} + 2T_{FA} + T_{mux(2-1)}$	3.84	$2r\theta(2A_{AND} + A_{FA}) + \theta(8r + 2)A_{FF}$	$\theta(1.8 + 9.96r)$
[STK00] ⁷	$\lambda(\epsilon + 1) + 2(\theta - 1)$				
[SKS07] ⁶	$n + 2$	$T_{mux(4-1)} + 2T_{FA} + T_{mux(2-1)} + 2T_{AND}$	4.04	$2r\theta A_{AND} + 6r\theta A_{FF} + 2r\theta(A_{FA} + A_{mux(4-1)}) + r\theta A_{mux(2-1)}$	$9.96r\theta$
[SKS07] ⁷	$n\lambda + 2$				
[PH08] ⁹	$\lambda(\theta + 4\frac{\theta}{r} + 1)$	$2T_{FA} + T_{mux(4-1)}$	2.49	$2(r + 1)\theta A_{FA} + (6r - 9)\theta A_{FF} + \theta A_{mux(2-1)} + \theta A_{XOR} + 2(r + 1)\theta A_{mux(4-1)}$	$r(3.8\theta + 5.4) + 0.38\theta + 9$
[HGEG11] ¹²	$n + \epsilon - 1$	$T_{AND} + 4T_{FA} + T_{mux(2-1)}$	4.49	$2(r\theta + \theta - 1)A_{mux(2-1)} + r\theta A_{HA} + \theta(2r + 6)A_{FA} + A_{XOR} + (3r\theta - 2r + \theta + 1)A_{FF}$	$r(4.46\theta - 1.8) + 9.66\theta - 7.33$
[HGEG11] ⁷	$n + \lambda(\epsilon - \theta) + \epsilon - 1$				

¹ design with 5-to-2 CSA ² design with 4-to-2 CSA ³ $\epsilon + 1 \leq \theta$ ⁴ $\epsilon + 1 \leq 2\theta$ ⁵ $\epsilon + 1 \leq \theta \lceil\theta/r\rceil$ ⁶ $n \leq \theta$, each PE is bit-sliced
⁷ otherwise ⁸ $f(r, L) = 6.27 \log_2(r + 8) + 4.18 \log_2(r) + 5.97$ ⁹ fastest case for time complexity, area calculated by the authors
¹⁰ calculated by the authors in [SL10] ¹¹ $g(r, L) = L(0.19r^2 + 33.6r + 13.1 \log_2(L) + 94.24)$, $h = 7$ ¹² $\epsilon \leq \theta$

Table 5.5: Normalized area-time complexity comparisons for a 1024-bit $GF(p)$ Montgomery multiplication (CPA delays included)¹

	Parameters	Delay	Area	area \times time
This work	$\beta = 6, r = 64$	2,440.8	18,278.1	4.46×10^7
[PH08]	$\theta = 64, r = 16$	2,659.8	4,010.92	1.07×10^7
[SCWL08]	$n = 1024$	4,046.3	12,963.8	5.25×10^7
[HKA ⁺ 05]	$\theta = 64, r = 16$	4,980.1	6,469.12	3.22×10^7
[SKS07]	$n = 1024$	5,169	10,199	5.27×10^7
[MMM04]	$n = 1024$	5,236.7	23,224.3	1.22×10^7
[HGEG11]	$\theta = 65, r = 16$	5,326.7	5,230.17	2.79×10^7
[SL10]	$\theta = 257, r = 4$	5,386.9	7,090.6	3.82×10^7
[TK03]	$\theta = 40, r = 8$	13,786.6	3,137.6	4.33×10^7
[STK00]	$\theta = 7, r = 32$	20,294.4	2,243.64	4.55×10^7

¹ Only the fastest versions of other works are considered

Table 5.4 offers a detailed, technology-independent comparison with the most up-to-date non-RNS solutions. The time complexity of the proposed architecture is a function $f(r, L)$, calculated according to Table 5.1, while it is a constant for each of the other works. Selecting $r \in [16, 512]$ and $L \in [2, 66]$, a plot for the time complexity $f(r, L)$ can be sketched, as shown in Figure 5.9.

The normalized area of the proposed architecture is provided as a 2-variable function $g(r, L)$, by selecting a typical value $h = 7$ for the word length of μ_i s. The other works provide functions of r , θ , and n . A plot of $g(r, L)$ is depicted in Fig. 5.10. The area of the proposed architecture, although larger, is of comparable magnitude as the other works. Unfortunately, complexity comparisons are not offered for the RNS implementation in [Gui10]. Instead, only metrics in terms of the number of Adaptive Logic Modules (ALM) of the Stratix II FPGA are given. Altera does not provide the number of gates per ALM, thus a direct comparison is infeasible.

To depict the trade-offs between RNS and non-RNS implementations, the area-time product is given in Figure 5.11 as a function $\sigma(r, L) = f(r, L)g(r, L)$ for a 1024-bit implementation. Table 5.5 was created by assigning values to the parameters L, r, θ, n in Table 5.4 used in other works to obtain numerical values for the number of cycles, normalized delay and area. The normalized areas are calculated for the complete designs and total execution time is calculated by the product cycles \times critical path, all under the common TMSO 0.13 μ m technology.

Based on the derived functions $f(r, L), g(r, L), \sigma(r, L)$, one can parametrize the proposed RNS architecture for minimum delay, area, or area \times time product, according to the requirements.

Table 5.6: Area-time comparisons for 1024-bit modular exponentiation

	Platform	Parameters	Cycles (per mult.)	Max Freq. (MHz)	Exponentiation time (ms)	Area (LUT)	Area (Gates)	Comments
This work	Xilinx XC4VFX60	$\beta = 22, r = 16$	624	142.23	8.99	29,379		RNS
		$\beta = 11, r = 32$	316	107.66	6.02	37,723	N/A	dual-field
		$\beta = 6, r = 64^5$	176	81.63	4.42	30,467		
[PH08]	Xilinx XC2V2000	$\theta = 64$ $r = 16$	651	248	5.38	8,428	N/A	non-RNS not dual-field
[NMSK01]	0.25 μm CMOS	$\beta = 11, r = 32$	248	80	6.35 ⁶	N/A	333,000	RNS not dual-field
[SL10]	Xilinx Virex-II-6	$\theta = 257$ $r = 4$	1,287	254.55	10.36 ¹	5,430	N/A	non-RNS not dual-field
[SCWL08]	Xilinx XC2V6000	$n = 1024$	1,028	152.49	13.82 ¹	25,074 ²	N/A	non-RNS not dual-field
[SKS07]	Xilinx 2VP100	$n = 1024$	1,026	140	15	5,892	N/A	non-RNS dual-field
[HKA+05]	Xilinx XC2V250-6	$\theta = 64$ $r = 16$	1,167	144	16.61	5,598	N/A	non-RNS dual-field
[HGEG11]	Xilinx XC2V6000	$\theta = 65$ $r = 16$	1,088	116.4	19.17 ¹	9,319	N/A	non-RNS not dual-field
[MMM04]	Xilinx XC2V6000	$n = 1024$	1,025	95.9	21.9 ³	23,208	165,048	non-RNS not dual-field
[TK03]	0.5 μm CMOS	$\theta = 40$ $r = 8$	3,458	80	88.61	N/A	28,000 ⁴	non-RNS not dual-field
[STK00]	1.2 μm CMOS	$\theta = 7$ $r = 32$	5,010	90	114.12 ¹	not reported	not reported	non-RNS dual-field

¹ Calculated by the authors

² Original value is 12,537 slices. Each slice contains 2 LUTs in the XC2V6000 device [Xil12a].

³ Result in CSA format, CPA delay should be included.

⁴ Equivalent area only for the multiplier

⁵ optimized for area and power

⁶ original value is 4.2 ms using a 4-bit window method exponentiation algorithm [NMSK01]

Table 5.7: Normalized area and delay of standard cells

	FF	FA	HA	OR	AND	XOR	MUX (2-1)	MUX (3-1)	MUX (4-1)
Delay	–	1	0.45	0.45	0.45	0.33	0.49	0.61	0.65
Area	0.9	1	0.57	0.19	0.19	0.33	0.38	0.71	0.9

5.3.4 Area-time-power comparisons

To verify the complexity comparisons presented in the previous paragraphs, a prototype of the proposed DRAMMM architecture was synthesized in Xilinx Virtex 4 (XC4VFX60) FPGA device using VHDL. Post-layout results were obtained from the Xilinx 14.4 ISE tool [Xil12b] and are presented in Table 5.6 for three combinations of β, r . Note that the last configuration for $\beta = 6, r = 64$ would not fit in the FPGA device, so it was optimized for area and power minimization, while the other two were optimized for timing performance using the corresponding strategies of the Xilinx 14.4 ISE tool.

In cases where other works reported area in terms of CLB slices, the equivalent area in terms of LUT was calculated according to the Xilinx FPGA datasheets [Xil12a]. Exponentiation execution times were calculated assuming that a single exponentiation requires $2n + 2$ Montgomery multiplications. Works in [LH08a, Gui10, TjZbXHQj10] provide results for precisions up to 512-bit, thus direct comparisons are infeasible. The works in [BDK98, Gro01] were also not included since no experimental results are presented. Finally, the work in [GLP⁺12] was omitted since it offers speed and area results for the fully-parallel case of $\beta = L = 33$, which is unfair to compare with the proposed configuration that employs time-sharing among MAC units. Additionally, area results are offered in terms of μm^2 which is incompatible with the values offered by other works.

Finally, power consumption measurements were performed using the XPower Analyzer functionality of Xilinx 14.4 ISE tool [Xil12b]. For the three configurations of the proposed DRAMMM in Table 5.6 the corresponding consumptions are 1,656/588 mW ($\beta = 22, r = 16$), 1,265/598 mW ($\beta = 11, r = 32$) and 1,076/595 mW ($\beta = 6, r = 64$) respectively, where the first value is the total consumption and the second is the leakage power. From the considered works only [HKA⁺05] reports that the complete unit draws 69/23 mW . In terms of power/throughput efficiency (throughput is in terms of exponentiations/sec) the corresponding values are 14.9 ($\beta = 22, r = 16$), 7.61 ($\beta = 11, r = 32$) and 4.75 ($\beta = 6, r = 64$) for the proposed architecture and 1.14 for [HKA⁺05].

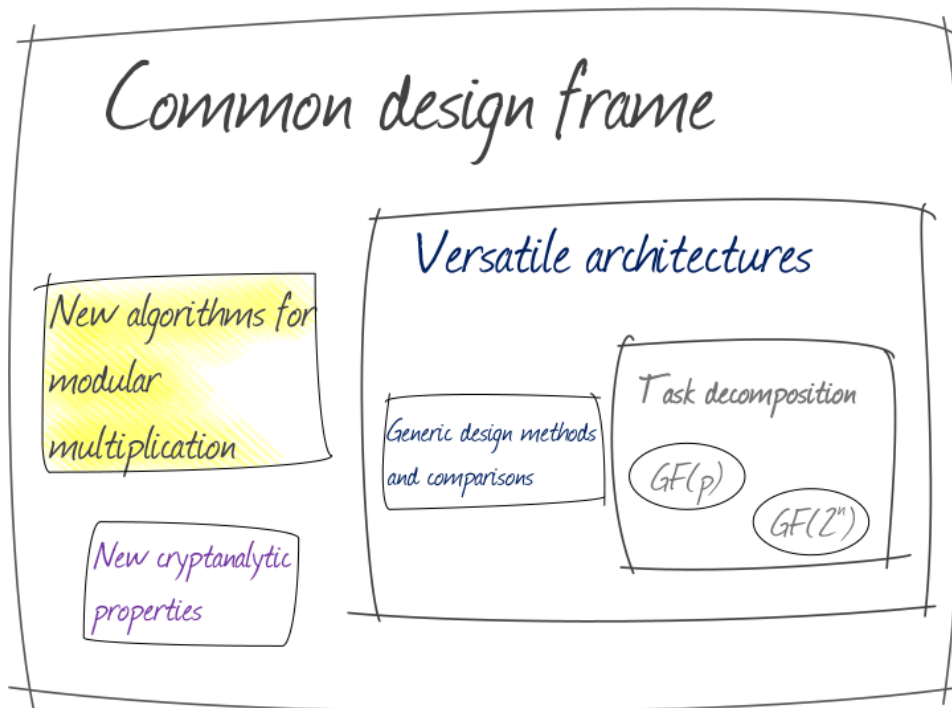
Table 5.6 verifies that the generic complexity comparisons presented in Table 5.4 can be a useful tool for comparing architectures of different implementation platforms and underlying arithmetic. The proposed architecture outperforms existing implementations in terms of total execution time for a modular exponentiation, with an overhead in area. Both Table 5.5 and Table 5.6 are sorted from the fastest to the slowest design for easy comparisons. The results of the complexity analysis are in accordance with the results obtained from synthe-

sis, since five out of nine references appear in the same ranking ([NMSK01] can be ignored since no complexity results are provided). Deviations exist mainly due to the fan-out factor, which is not included in the complexity analysis, plus due to the completely different characteristics of each implementation platform, even between FPGA packages of the same FPGA family [Xil12a].

5.4 Summary

The mathematical framework and a flexible, dual-field, residue arithmetic architecture for Montgomery multiplication in $GF(p)$ and $GF(2^n)$ is developed and the necessary conditions for the system parameters (number of moduli channels, modulus word-length) are derived. The proposed DRAMMM architecture supports all operations of Montgomery multiplication in $GF(p)$ and $GF(2^n)$, residue-to-binary and binary-to-residue conversions, MRC for integers and polynomials, dual-field modular exponentiation and inversion, in the same hardware. A generic, technology-independent methodology to evaluate the optimal system parameters (number of moduli, modulus word-length) was also presented. Generic complexity and real performance comparisons with state-of-the-art works prove the potential of residue arithmetic exploitation in Montgomery multiplication.

Novel RNS algorithms for modular multiplication



A new RNS modular multiplication algorithm based on Barrett's technique is presented in this chapter. The necessary conditions for the algorithm's validity, as well as the conditions to apply the algorithm in modular exponentiation are also derived. Algorithmic and architectural comparisons with the state-of-the-art solutions for RNS Montgomery multiplication are also offered.

6.1 New RNS modular multiplication algorithm based on Barrett's technique

6.1.1 Barrett Modular Multiplication

Barrett's modular reduction method requires the pre-computation of one parameter $\mu = \left\lfloor \frac{2^{2n}}{N} \right\rfloor$ which remains constant as long as the n -bit reduction modulus N does not change. In this case, like in Montgomery's algorithm, modular multiplication is realized by multiplying the input operands and then reducing the result. Assuming two n -bit input operands x, y and their product $s = x \cdot y$, then Barrett Modular Multiplication (BMM) is realized as

$$w = s - \left\lfloor \left\lfloor \frac{s}{2^n} \right\rfloor \frac{\mu}{2^n} \right\rfloor N, \quad (6.1)$$

where $w < 3N$ and $w \equiv s \pmod{N}$, so that the modulus N may be needed to be subtracted once or twice to obtain the exact result [Bar87].

6.1.2 Proposed RNSBMM algorithm

The first step towards the proposed RNSBMM algorithm is to transform (6.1) to an RNS-friendly form [SS13]. For this reason, (6.1) is rewritten in 6 steps, depicted below as Algorithm 6.1.

Algorithm 6.1 The proposed RNSBMM algorithm

Input: x, y in $\mathcal{A} \cup \alpha_r$, $x, y < 3N$

Output: $w \equiv xy \pmod{N}$, $w < 3N$ in $\mathcal{A} \cup \alpha_r$

Precompute: μ, N in $\mathcal{A} \cup \alpha_r$

- 1 $s = x \cdot y$ in $\mathcal{A} \cup \alpha_r$
 - 2 $t = SR(s)$ in $\mathcal{A} \cup \alpha_r$
 - 3 $u = t \cdot \mu$ in $\mathcal{A} \cup \alpha_r$
 - 4 $v = SR(u)$ in $\mathcal{A} \cup \alpha_r$
 - 5 $p = v \cdot N$ in $\mathcal{A} \cup \alpha_r$
 - 6 $w = s - p$ in $\mathcal{A} \cup \alpha_r$
-

The proposed algorithm is executed in a base \mathcal{A} along with an extra modulus channel corresponding to a redundant modulus α_r for reasons to be explained later on. Steps 1, 3, 5, and 6 of the proposed RNSBMM algorithm are normal subtractions and multiplications, thus they can be executed in parallel in the RNS base $\mathcal{A} \cup \alpha_r$. Steps 2 and 4 include scaling of RNS numbers by 2^n and then rounding of the result based on the floor function. In the context of RNSBMM an SR operation by 2^n is defined, that is $SR(x) = \left\lfloor \frac{x}{2^n} \right\rfloor$.

Observe that the upper bound for the inputs was modified from $x, y < N$ to $x, y < 3N$ so that inputs and outputs are compatible with each other. This allows recursive use of the proposed RNSBMM to construct modular exponentiation as well [Des09]. Due to this change, there are two issues that need to be addressed. First, the conditions so that the result w

remains bounded by $w < 3N$ should be derived, and secondly, based on the previous result, the minimum range A for a valid RNS incorporation to the BMM algorithm should be evaluated.

In the following, the word-length $n^* = n + \xi$ is used to denote that ξ more bits are assigned to represent all internal calculations in the proposed RNSBMM and N is considered as an n -bit modulus. The proposed technique attempts to evaluate the appropriate conditions for n^* so that for inputs $x, y < 3N$ the output is upper-bounded by $w < 3N$.

Our proof is based on a fundamental inequality for the floor function, i.e., $\lfloor x \rfloor = m$ if and only if $x - 1 < m \leq x$ for $m \in \mathbb{Z}$ and $x \in \mathbb{R}$. Let $t = \left\lfloor \frac{(3N)^2}{2^{2n^*}} \right\rfloor$ denote the maximum value for t for inputs $x = y = 3N$ (t corresponds to step 2 of the RNSBMM). For the quantities t, μ it holds that

$$\begin{aligned} \frac{(3N)^2}{2^{2n^*}} - 1 < t \leq \frac{(3N)^2}{2^{2n^*}} \\ \frac{2^{2n^*}}{N} - 1 < \mu \leq \frac{2^{2n^*}}{N}. \end{aligned} \quad (6.2)$$

By multiplying both terms in (6.2) we formulate $u = t\mu$ (step 3 of RNSBMM). It holds that

$$\begin{aligned} \left(\frac{(3N)^2}{2^{2n^*}} - 1 \right) \left(\frac{2^{2n^*}}{N} - 1 \right) < t\mu \leq \frac{(3N)^2}{2^{2n^*}} \frac{2^{2n^*}}{N} \Rightarrow \\ \frac{(3N)^2}{2^{2n^*}} \frac{2^{2n^*}}{N} - \left(\frac{(3N)^2}{2^{2n^*}} + \frac{2^{2n^*}}{N} - 1 \right) < u \leq \frac{(3N)^2}{2^{2n^*}} \frac{2^{2n^*}}{N} \Rightarrow \\ 9N - \overbrace{\left(\frac{(3N)^2}{2^{2n^*}} + \frac{2^{2n^*}}{N} - 1 \right)}^{\zeta} \leq \frac{u}{2^{2n^*}} \leq 9N \Rightarrow \\ 9N - \zeta \leq \frac{u}{2^{2n^*}} \leq 9N \Rightarrow \\ 9N - \lfloor \zeta \rfloor < v \leq 9N \Rightarrow \\ -9N^2 \leq -vN < \lfloor \zeta \rfloor N - 9N^2 \Rightarrow \\ 9N^2 - 9N^2 \leq s - vN < 9N^2 + \lfloor \zeta \rfloor N - 9N^2 \Rightarrow \\ 0 \leq w < \lfloor \zeta \rfloor N. \end{aligned} \quad (6.3)$$

At this point it is proven that w is upper-bounded by $w < \lfloor \zeta \rfloor N$. It is required to evaluate the conditions so that $\lfloor \zeta \rfloor \leq 3$, in order for the condition $w < 3N$ to hold. Based on another fundamental inequality for the floor function, i.e., $\lfloor x \rfloor = m$ if and only if $m \leq x < m + 1$ for $m \in \mathbb{Z}$ and $x \in \mathbb{R}$, it holds that

$$\begin{aligned} \lfloor \zeta \rfloor \leq 3 \Rightarrow \zeta < 4 \Rightarrow \frac{9N^2}{2^{2n^*}} + \frac{2^{2n^*}}{N} - \frac{1}{2^{2n^*}} < 4 \Rightarrow \\ 9N^2 < 2^{2n^*} \left(4 - \frac{2^{2n^*}}{N} + \frac{1}{2^{2n^*}} \right). \end{aligned} \quad (6.4)$$

Taking the base-2 logarithms on both sides of (6.4) to obtain a relation for the word-length of n^* (assuming both sides of (6.4) are positive) we get

$$\begin{aligned}
 \lceil \log_2 9 + 2n \rceil &< \left\lceil 2n^* + \log_2 \left(4 - \frac{2^{n^*}}{N} + \frac{1}{2^{n^*}} \right) \right\rceil \Rightarrow \\
 2n + 4 &< 2n^* + \left\lceil \log_2 \left(4 - \frac{2^{n^*}}{N} + \frac{1}{2^{n^*}} \right) \right\rceil \Rightarrow \\
 2n + 4 &< 2n + 2\xi + \overbrace{\left\lceil \log_2 \left(4 - \frac{2^{n+\xi}}{N} + \frac{1}{2^{n+\xi}} \right) \right\rceil}^C \Rightarrow \\
 \xi &> 2 - \frac{C}{2}.
 \end{aligned} \tag{6.5}$$

We make an observation on the quantity C , that is $\max\{2 - \frac{C}{2}\} = 1$, thus $\xi > 1$. Since it is required that $n^* > n$, that is the available word-length for calculations is always larger than the modulus word-length, then from (6.5) $2 - \frac{C}{2} \geq 0$ should also hold. In that case we get

$$\begin{aligned}
 2 - \frac{C}{2} \geq 0 &\Rightarrow C \leq 4 \Rightarrow \\
 \left\lceil \log_2 \left(4 - \frac{2^{n+\xi}}{N} + \frac{1}{2^{n+\xi}} \right) \right\rceil &\leq 4 \Rightarrow \\
 4 - \frac{2^{n+\xi}}{N} + \frac{1}{2^{n+\xi}} \leq 16 &\Rightarrow n + \xi > 0 \text{ and } 0 < N \leq 2^{n+\xi}
 \end{aligned} \tag{6.6}$$

which holds by definition for all n, ξ, N , assuming that the logarithmic function elimination is greater than 0. Under this restriction we finally get

$$\begin{aligned}
 4 - \frac{2^{n+\xi}}{N} + \frac{1}{2^{n+\xi}} &\geq 1 \Rightarrow \\
 \frac{1}{N} 2^{2n^*} - 3 \cdot 2^{n^*} - 1 &\leq 0 \stackrel{\rho=2^{n^*}}{\Rightarrow} \\
 \rho^2 - 3N\rho - N &\leq 0 \Rightarrow \rho > 0 \text{ and } N \geq \frac{\rho^2}{3\rho + 1}.
 \end{aligned} \tag{6.7}$$

Figure 6.1 depicts the solutions of the inequality in (6.7). Note that only positive values are of interest. Under this restriction, which can be easily met for positive values of ρ, N , all previous inequalities hold and thus the proposed algorithm is valid. Conditions in (6.7) provide also the desired word-length for ξ . Consider relaxing the expression for N in (6.7) as

$$N \geq \frac{\rho^2}{3\rho} \Rightarrow N \geq \frac{2^{2n^*}}{3 \cdot 2^{n^*}} \Rightarrow N \geq \frac{2^{n^*}}{3}, \tag{6.8}$$

which simplifies to $\xi \leq 2$ after logarithmic elimination. In combination with the result from (6.5) it is required that $1 < \xi \leq 2$. Therefore by selecting an RNS base that handles operands

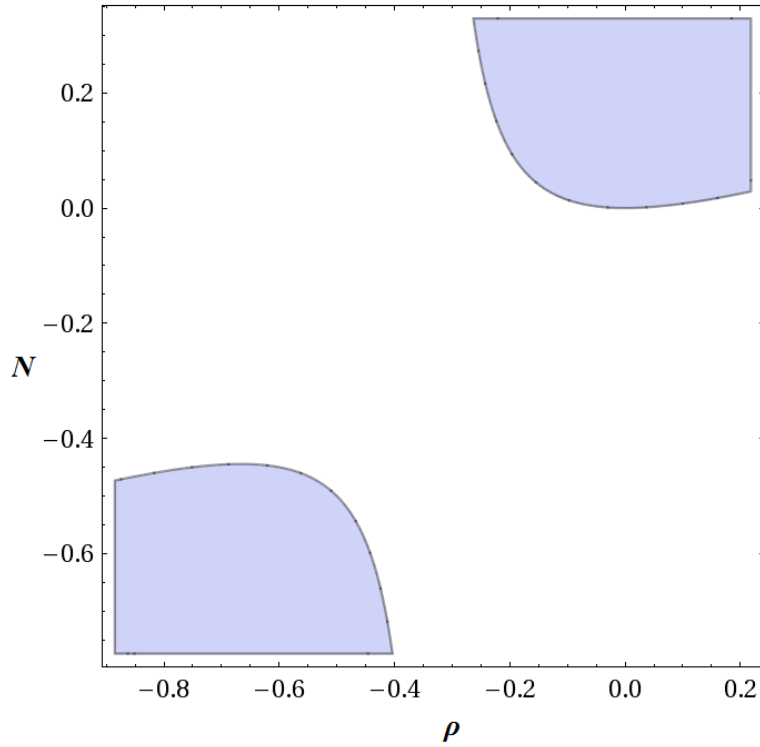


Figure 6.1: Region plot for inequality (6.7)

of at most $n^* = n + \xi$ -bit long it is assured that (6.4) also holds and the result w will always be upper-bounded by $w < 3N$ for inputs $x, y < 3N$.

This proof is one of the main contributions of this work, since it is proved that the proposed RNSBMM can be executed repeatedly in the context of any modular exponentiation algorithm to achieve modular exponentiation in RNS. More importantly, the correction step that subtracts once or twice the modulus N according to (6.1) can be executed after the exponentiation is completed. An example of an RNSBMM exponentiation algorithm is shown below as Algorithm 6.2.

It remains to evaluate an appropriate range A so that all calculations within the RNSBMM are valid. Assuming that each modulus is r -bit long, then for the base \mathcal{A} it holds that

$$\begin{aligned}
 A &> (3n^*)^2 \Rightarrow \\
 \log_2 A &> \left\lceil \log_2(9) + \log_2 n^{*2} \right\rceil \Rightarrow \\
 rk &> 2n^* + 4.
 \end{aligned} \tag{6.9}$$

Equation (6.9) holds because the maximum internal value in the RNSBMM corresponds to single multiplication step, since multiplications in steps 1, 3 and 5 are always followed by scaling or subtraction operations, thus the intermediate results never exceed $(3n^*)^2$. (6.9) also provides the conditions for selecting an appropriate number of RNS moduli and their word-length so that RNSBMM is valid. For example, for $n^* = 1026$ -bit calculations (N a

1024-bit modulus) and assuming that each modulus channel handles $r = 32$ -bit operands, an RNS base with at least $k = 65$ moduli is sufficient so that (6.5), (6.6), (6.9) hold.

Similar to the state-of-the-art RNSMMM algorithms in [KKSS00, BDK01, GLP⁺12] where BC operations define the total complexity of the algorithms, SR operations define the complexity and critical path of the proposed RNSBMM as will be shown in the following sections.

Algorithm 6.2 Proposed RNSBMM modular exponentiation

Input: $x_{\mathcal{A} \cup \alpha_r}, e = (e_{n-1} \dots e_1 e_0)_2$

Output: $b_{\mathcal{A} \cup \alpha_r}, b \equiv \langle b^e \rangle_N$

Precompute: μ, N in $\mathcal{A} \cup \alpha_r$

```

1  $b \leftarrow 1$  in  $\mathcal{A} \cup \alpha_r$ 
2 for  $i = n - 1, \dots, 0$  do
3    $b \leftarrow \text{RNSBMM}(b, b)$ 
4   if  $e_i = 1$  then
5      $b \leftarrow \text{RNSBMM}(b, x)$ 
6   end if
7 end for
8 return  $b$ 

```

6.1.3 Scaling and rounding of an RNS number

The proposed SR technique is based on a scaling-by-2 algorithm developed in [MBS03]. We assume the scaling of an integer x by a scaling constant f . The algorithm is based on the following theorems:

Theorem 1: (Exact Division) $\langle x/f \rangle_{\mathcal{A}} = \langle x f^{-1} \rangle_{\mathcal{A}} \leftrightarrow (\langle x f^{-1} \rangle_{\alpha_1}, \langle x f^{-1} \rangle_{\alpha_2}, \dots, \langle x f^{-1} \rangle_{\alpha_k})$ if, and only if, $f \mid x \leftrightarrow (x_1, x_2, \dots, x_k)$ without remainder and $\gcd(f, \alpha_i) = 1, \forall i \in [1, 2, \dots, k]$.

Proof. For proof see [ST67], p.38. □

For RNS sets with odd moduli the division by two is substituted by a multiplication with the multiplicative inverse of two using the following corollary:

Theorem 2: (Odd-Modulus Set Division Without Remainder) If $f = 2 \mid x \leftrightarrow (x_1, x_2, \dots, x_k)$, then $\langle x/2 \rangle_{\mathcal{A}} = \langle 2^{-1}x \rangle_{\mathcal{A}} \leftrightarrow (\langle 2^{-1}x \rangle_{\alpha_1}, \langle 2^{-1}x \rangle_{\alpha_2}, \dots, \langle 2^{-1}x \rangle_{\alpha_k})$ implements a scaling by two, where $\langle 2^{-1} \rangle_{\alpha_i}$ is the multiplicative inverse of two with respect to α_i .

Proof. For proof see [MBS03]. □

Theorem 3: (Odd-Modulus Set Division With Remainder) If $f = 2 \nmid x \leftrightarrow (x_1, x_2, \dots, x_k)$, then $\langle 2^{-1}(x+1) \rangle_{\mathcal{A}} \leftrightarrow (\langle 2^{-1}(x+1) \rangle_{\alpha_1}, \langle 2^{-1}(x+1) \rangle_{\alpha_2}, \dots, \langle 2^{-1}(x+1) \rangle_{\alpha_k})$ implements a scaling by two.

Proof. For proof see [MBS03]. □

The theorems above imply that for an RNS system with odd moduli, a scaling-by-2 scheme can be devised by first checking whether 2 divides x . If not, x is odd thus $(x+1)$ is even so $(x+1)$ is selected to be multiplied with the multiplicative inverse of 2, i.e., $\langle \times 2^{-1} \rangle_{\alpha_i}, \forall i \in [1, k]$.

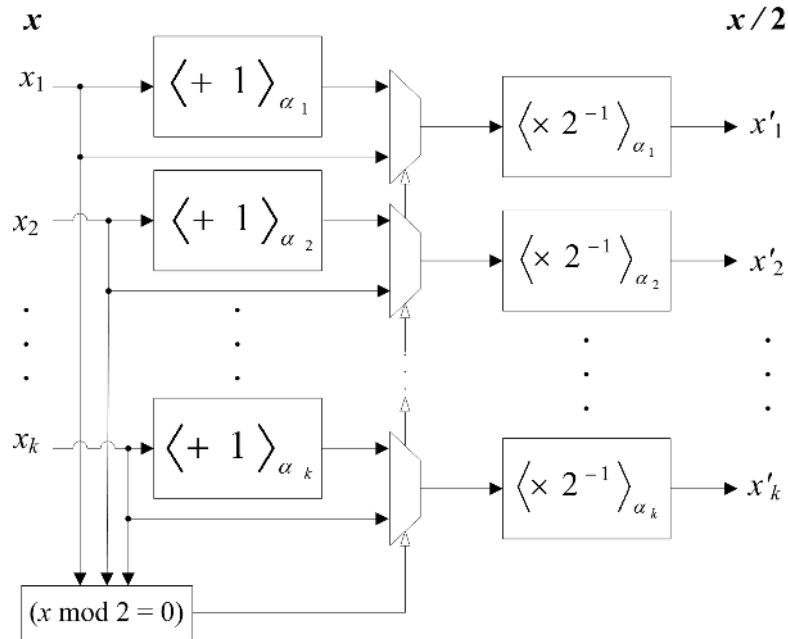


Figure 6.2: Scaling by two scheme [MBS03]

A suitable architecture for the scheme is shown in Fig.6.2. Note that, this scheme performs the “round-to-nearest” method [EL04]. If a rounding to next smaller integer is required, i.e., the floor function, the increment in Fig.6.2 should be replaced by a decrement, that is $\langle -1 \rangle_{\alpha_i}$. This solves the problem of floor function evaluation in RNS, which is required by the proposed RNSBMM.

However, in order to incorporate the scheme to the proposed RNSBMM, the technique should be enhanced to perform scaling by higher powers of 2. In general, two approaches are possible. Either the scheme in Fig.6.2 may be iteratively used n times, or a larger look-up table may be used, in order to get the right offset O , so that 2^n divides $(x - O)$ (just like 2 divides $(x - 1)$ in the scaling-by-2 case with rounding-to-next-smaller). Since it is desirable to avoid iterations of the scheme in order to achieve a parallel architecture and since look-up-tables can grow too large for higher powers of 2 (> 32), a technique for scaling by 2^n based on [SK89] is developed in the following subsection.

6.1.3.1 Divisibility check of an RNS number by 2^n

The proposed scaling-by- 2^n scheme is shown in Fig.6.3 and is an extension of the scheme in Fig.6.2. The architecture checks whether 2^n divides x and if not it calculates the correct offset O . The value $(x - O)$ is then multiplied by the multiplicative inverse of 2^n to complete the scaling. Apparently, the module that checks whether 2^n divides x determines the critical path of the architecture.

We make an observation on the offset O . From the fundamental equation for integer divi-

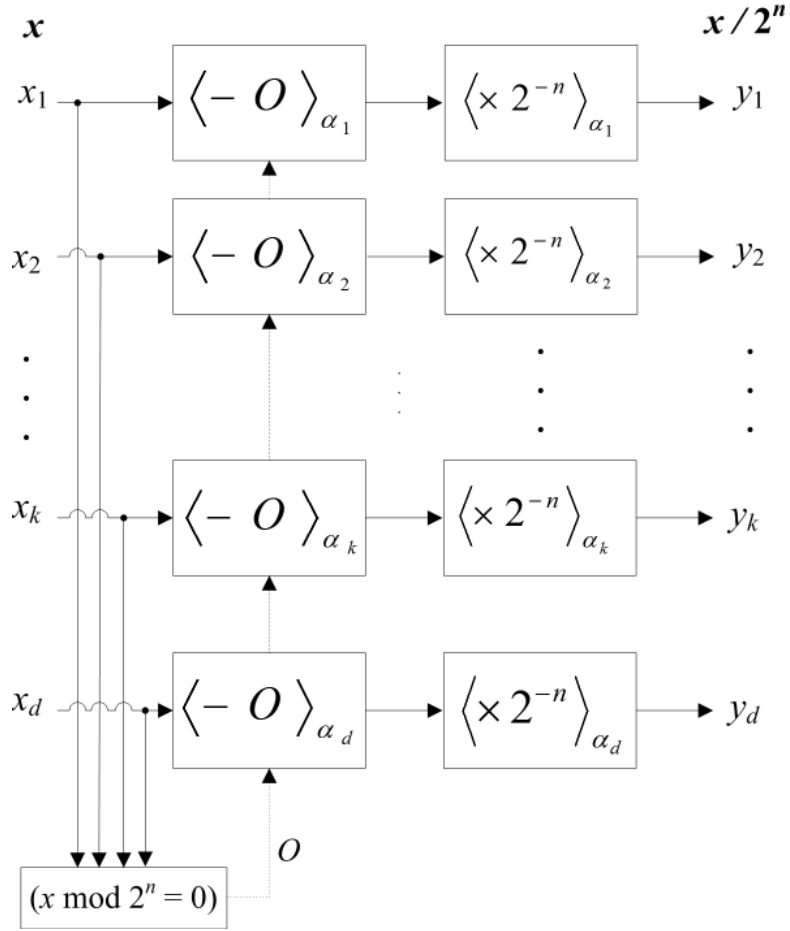


Figure 6.3: The proposed SR scheme for RNSBMM

sion it holds that

$$y = \left\lfloor \frac{x}{2^n} \right\rfloor = \frac{x - \langle x \rangle_{2^n}}{2^n} \Rightarrow x - \langle x \rangle_{2^n} = y \cdot 2^n \quad (6.10)$$

or $SR(x) = y$. By taking both sides of (6.10) modulo $\alpha_i, \forall i \in [1, k]$ we obtain that

$$\begin{aligned} \langle y \rangle_{\alpha_i} &= \langle \langle x_{\alpha_i} - \langle x \rangle_{2^n} \rangle_{\alpha_i} \cdot \langle 2^{-n} \rangle_{\alpha_i} \rangle_{\alpha_i} \Rightarrow \\ y_i &= \langle \langle x_i - O \rangle_{\alpha_i} \cdot \langle 2^{-n} \rangle_{\alpha_i} \rangle_{\alpha_i}, \forall i \in [1, k] \end{aligned} \quad (6.11)$$

which is the mathematical expression of the proposed scheme in Figure 6.3. The condition for the existence of $\langle 2^{-n} \rangle_{\alpha_i}$ is $\gcd(2^n, \alpha_i) = 1, \forall i \in [1, k]$. (6.11) implies that the offset O that needs to be subtracted from x is

$$O = \langle x \rangle_{2^n} \quad (6.12)$$

or, in other words, the n LSB of x . Thus the problem of evaluating the offset O transposes to the problem of finding the n LSB of x .

An algorithm developed in [SK89] may be employed at this point to evaluate the offset O . The algorithm requires a redundant modulus $\alpha_r \geq k$ so that the RNS base \mathcal{A} is extended to

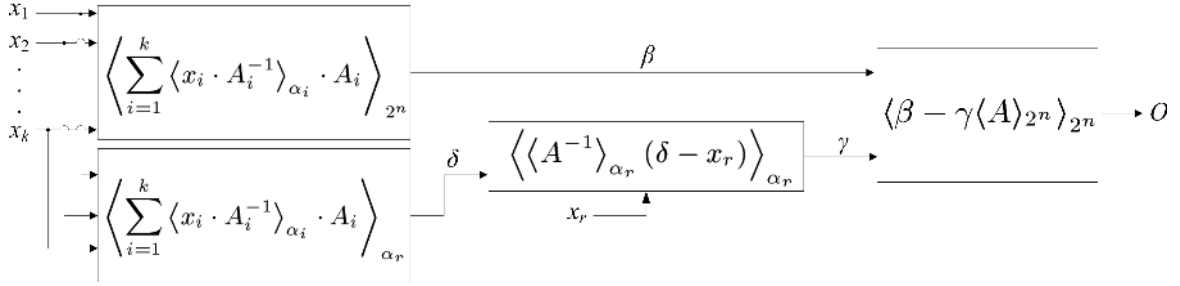


Figure 6.4: The proposed offset evaluation block

$\mathcal{A} = (\alpha_1, \alpha_2, \dots, \alpha_k | \alpha_r)$. This redundant channel will be available from now on throughout the calculations of the RNSBMM. Let x be an integer with an RNS representation $x_{\mathcal{A}} = (x_1, x_2, \dots, x_k | x_r)$, where $x_r = \langle x \rangle_{\alpha_r}$. By reducing both sides of (2.29) $\text{mod } \alpha_r$ we obtain that

$$\begin{aligned} \langle x \rangle_{\alpha_r} &= \left\langle \left\langle \sum_{i=1}^k \langle x_i \cdot A_i^{-1} \rangle_{\alpha_i} \cdot A_i \right\rangle_{\alpha_r} - \langle \gamma A \rangle_{\alpha_r} \right\rangle_{\alpha_r} \Rightarrow \\ \langle \gamma \rangle_{\alpha_r} &= \left\langle \left\langle A^{-1} \right\rangle_{\alpha_r} \left(\left\langle \sum_{i=1}^k \langle x_i \cdot A_i^{-1} \rangle_{\alpha_i} \cdot A_i \right\rangle_{\alpha_r} - \langle x \rangle_{\alpha_r} \right) \right\rangle_{\alpha_r} \\ &= \left\langle \left\langle A^{-1} \right\rangle_{\alpha_r} (\delta - \langle x \rangle_{\alpha_r}) \right\rangle_{\alpha_r}, \end{aligned} \quad (6.13)$$

where $\delta = \left\langle \sum_{i=1}^k \langle x_i \cdot A_i^{-1} \rangle_{\alpha_i} \cdot A_i \right\rangle_{\alpha_r}$. Since $\gamma < k$ and $\alpha_r \geq k$ it follows that $\gamma = \langle \gamma \rangle_{\alpha_r}$ [SK89]. All terms on the right hand side of (6.13) are known, thus the correction factor γ can be substituted in (2.29) to obtain x and then use the result in (6.12) to calculate O . Let us rewrite (6.12) as

$$\begin{aligned} O = \langle x \rangle_{2^n} &= \left\langle \left\langle \sum_{i=1}^k \langle x_i \cdot A_i^{-1} \rangle_{\alpha_i} \cdot A_i \right\rangle_{2^n} - \langle \gamma A \rangle_{2^n} \right\rangle_{2^n} = \\ &= \langle \beta - \langle \gamma A \rangle_{2^n} \rangle_{2^n} = \langle \beta - \gamma \langle A \rangle_{2^n} \rangle_{2^n} = \\ &= \left\langle \beta - \left\langle \left\langle A^{-1} \right\rangle_{\alpha_r} (\delta - x_r) \right\rangle_{\alpha_r} \langle A \rangle_{2^n} \right\rangle_{2^n}, \end{aligned} \quad (6.14)$$

where δ can be efficiently computed by choosing α_r to be of the convenient form $2^l \pm 1$, $l < r$. A block diagram for the offset evaluation is shown in Figure 6.4.

After scaling of an RNS number it is required to update the redundant modulus channel in order to make the result compatible with subsequent scalings. By reducing $\text{mod } \alpha_r$ both sides of (6.10) we obtain

$$\begin{aligned} \langle y \rangle_{\alpha_r} &= \langle (x - \langle x \rangle_{2^n}) \cdot \langle 2^{-n} \rangle_{\alpha_r} \rangle_{\alpha_r} \Rightarrow \\ y_r &= \langle \langle x_r - O \rangle_{\alpha_r} \cdot \langle 2^{-n} \rangle_{\alpha_r} \rangle_{\alpha_r}, \end{aligned} \quad (6.15)$$

where x_r and O have already been computed during the scaling phase and $\langle 2^{-n} \rangle_{\alpha_r}$ is a constant which can be precomputed. Obviously, it should hold that $\gcd(2^n, \alpha_r) = 1$ to allow the existence of $\langle 2^{-n} \rangle_{\alpha_r}$.

6.1.4 Numerical examples

Let us demonstrate the validity of the proposed RNSBMM by two numerical examples for a modular multiplication and a modular exponentiation respectively. Assume the 16-bit modulus $N = 65521$, $N < 2^{16}$ a prime. We select $x, y < 3N$ that is $x = 3N - 65 = 196498$, $y = 2N - 1 = 131005$. By substituting the values for the constant μ we get $\mu = \left\lfloor \frac{2^{32}}{65521} \right\rfloor = 65551$ and from (6.1) the result is $w = 196498 \cdot 131005 - \left\lfloor \left\lfloor \frac{196498 \cdot 131005}{2^{16}} \right\rfloor \frac{\mu}{2^{16}} \right\rfloor 65521 = 67926 < 3N$. The exact result can be obtained by subtracting once the modulus N , i.e., $w = 67926 - 65521 = 2405$.

For the proposed RNSBMM the modulus set $\mathcal{A} = (131, 137, 139, 149, 151||5)$ is chosen, where the redundant modulus $\alpha_r = 5 \geq k = 5$ and $\gcd(2^{16}, 5) = 1$. It is easy to check that $A = \prod_{i=1}^5 \alpha_i = 56126747867 > [3(65521 - 1)]^2$ as dictated by (6.9). The inputs x, y and the constants μ, N are computed in $\mathcal{A} \cup \alpha_r$ as following:

- $x = (129, 40, 91, 116, 47||3)$
- $y = (5, 33, 67, 34, 88||0)$
- $\mu = (51, 65, 82, 140, 17||1)$
- $N = (21, 35, 52, 110, 138||1)$
- $A_i = A/\alpha_i = (428448457, 409684291, 403789553, 376689583, 371700317)$
- $A_i^{-1} = (120, 130, 16, 1, 2)$
- $\langle 2^{-16} \rangle_{\mathcal{A} \cup \alpha_r} = (91, 74, 83, 31, 76||1)$

Executing Alg.6.1 with the previous values yields

- 1 $s = (129, 40, 91, 116, 47||3) \cdot (5, 33, 67, 34, 88||0) = (121, 87, 120, 70, 59||0)$
- 2 $t = (57, 16, 120, 31, 44||0)$ with $\beta = 47963, \gamma = 3, \delta = 1, O = 7370$
- 3 $u = (57, 16, 120, 31, 44||0) \cdot (51, 65, 82, 140, 17||1) = (25, 81, 110, 19, 144||0)$
- 4 $v = (15, 105, 70, 120, 133||4)$ with $\beta = 34278, \gamma = 3, \delta = 1, O = 59221$
- 5 $p = (15, 105, 70, 120, 133||4) \cdot (21, 35, 52, 110, 138||1) = (53, 113, 26, 88, 83||4)$
- 6 $w = (121, 87, 120, 70, 59||0) - (53, 113, 26, 88, 83||4) = (68, 111, 94, 131, 127||1)$

By applying (2.29) on the result w it can be verified that $w = 67926$, which is the same result obtained by the original BMM execution. Subtracting once the modulus and applying again (2.29) we get $w = (68, 111, 94, 131, 127||1) - (21, 35, 52, 110, 138||1) = (47, 76, 42, 21, 140||0) = 2405_{10}$, which is the exact result.

Assuming an exponent $e = 2^{16} - 564$ and applying Algorithm 6.2 for the same input $x = (129, 40, 91, 116, 47||3)$ the algorithm outputs $b = (70, 70, 11, 107, 111||0) = 89805_{10}$. Subtract-

ing once the modulus $N = (21, 35, 52, 110, 138 || 1)$ we obtain $b = (49, 35, 98, 146, 124 || 4) = 24282_{10}$ which is the exact result. All calculations were done with Mathematica [Wol13].

6.2 Complexity analysis - comparisons

6.2.1 Complexity Comparisons

Evaluations for the algorithmic complexity of the proposed RNSBMM is presented in this section. The metric is the same followed by [KKSS00, BDK01, GLP⁺12], i.e., the number of small r -bit modular multiplications required to perform one RNSBMM. In cases where normal multiplications are required we consider the cost of a normal multiplication to be a fraction of the cost of a modular multiplication. Assuming that the normalized cost of a modular multiplication is $C_{MM} = 1$, then the cost of a normal multiplication will be $C_{NM} = \frac{1}{\epsilon} C_{MM}$, where ϵ an arbitrary positive variable. We proceed by examining the proposed RNSBMM step-by-step:

- Steps 1, 3, 5: these steps are performed completely in parallel in all moduli channels. Each step requires $k + 1$ parallel modular multiplications, thus in total $3k + 3$ modular multiplications are required.
- Step 2: the complexity of the SR step corresponds to the complexity implied by Figure 6.3. There, $k + 1$ parallel modular multiplications are required for multiplications by $\langle 2^{-n} \rangle_{\alpha_i}$ in $\mathcal{A} \cup \alpha_r$. The complexity of the $x \bmod 2^n = 0$ module corresponds to the complexity of the offset evaluation block in Figure 6.4 (values β, γ, δ, O).
 - β evaluation: this step requires k modular multiplications to evaluate the r -bit partial products $\langle x_i \cdot A_i^{-1} \rangle_{\alpha_i}$ and another k normal multiplications of these partial products by $\langle A_i \rangle_{2^n}$. By considering that $\langle A_i \rangle_{2^n}$ is n -bit long and assuming we want to use only r -bit operators in each RNS channel, we can split an $r \times n$ -bit multiplication in $k/2$ r -bit multiplications according to the restriction in (6.9). In total this amounts to $k + k \times (k/2) \frac{1}{\epsilon} = \frac{1}{2\epsilon} k^2 + k$ modular multiplications for this step.
 - γ evaluation: 1 small ($\bmod \alpha_r$) multiplication between the pre-computed value $\langle A^{-1} \rangle_{\alpha_r}$ and $(\delta - x_r)$ is required.
 - δ evaluation: the k partial products $\langle x_i \cdot A_i^{-1} \rangle_{\alpha_i}$ have already been calculated from β evaluation thus only k modular multiplications by the pre-computed values $\langle A_i \rangle_{\alpha_r}$ should be considered in this case.
 - O evaluation: the complexity is the summation of complexity values for β, γ, δ , plus $k/2$ normal multiplications for the product $\gamma \langle A \rangle_{2^n}$, i.e., $\frac{1}{2\epsilon} k^2 + (2 + \frac{1}{2\epsilon})k + 1$ modular multiplications in total.
 - SR evaluation: the complexity corresponds to the complexity for computing O , plus $k + 1$ modular multiplications by $\langle 2^{-n} \rangle_{\mathcal{A} \cup \alpha_r}$, plus $(k + 1) \frac{k}{2} \frac{\epsilon - 1}{\epsilon}$ modular re-

ductions for the $\langle -O \rangle_{\alpha_i}$ calculations, i.e., $\frac{1}{2}k^2 + \frac{7}{2}k + 2$ modular multiplications in total.

Putting all the above together, the proposed algorithm requires

$$\overbrace{(3k+3)}^{\text{steps 1,3,5}} + 2 \overbrace{\left(\frac{1}{2}k^2 + \frac{7}{2}k + 2\right)}^{\text{steps 2,4}} = k^2 + 10k + 7 \quad (6.16)$$

modular multiplications in total.

Steps in RNSMMM	[KKSS00]	[BDK01]	[GLP ⁺ 12] applied in [KKSS00]	[GLP ⁺ 12] applied [BDK01]
1, 3, 4	$5k$	$5k$	$2k$	$2k$
First BC	$k^2 + 2k$	$k^2 + k$	$k^2 + 3k$	$k^2 + 2k$
Second BC	$k^2 + 2k$	$k^2 + 2k$	$k^2 + k$	$k^2 + k$
Total	$2k^2 + 9k$	$2k^2 + 8k$	$2k^2 + 6k$	$2k^2 + 5k$

Table 6.1: Number of modular multiplications in state-of-the-art RNSMMM

Tables 6.1 and 6.2 compare the proposed algorithm with the state-of-the-art RNSMMM algorithms in [KKSS00, BDK01, GLP⁺12]. An important remark is that, all values for the proposed RNSBMM correspond to the values calculated previously, but with k replaced by $2k$. The reason is that the works in [KKSS00, BDK01, GLP⁺12] present complexity results with k corresponding to the number of moduli in one of the two bases required to implement RNSMMM, since in these cases the BC operations are performed in only one of the two bases (thus $2k$ moduli are required in these algorithms). Since the proposed RNSBMM utilizes only one base and all operations are performed on all channels, k should be replaced by $2k$ to achieve an accurate comparison.

From Tables 6.1 and 6.2 it is clear that the proposed RNSBMM requires twice the number of modular multiplications per RNS modular multiplication compared to existing solutions.

Steps in RNSBMM	This work
1, 3, 5	$3(2k + 1)$
First SR	$2k^2 + 7k + 2$
Second SR	$2k^2 + 7k + 2$
Total	$4k^2 + 20k + 7$

Table 6.2: Number of modular multiplications in the proposed RNSBMM

Operation	Base	# multiplications	# cycles
$s = xy$	\mathcal{B}	k	ϵ
$\hat{s} = \hat{x}\hat{y}$	\mathcal{A}	k	$\lfloor \frac{1}{\epsilon+M-1} \rfloor$
$q = s(-N^{-1}B_i^{-1})$	\mathcal{B}	k	ϵ
$\hat{w} = \hat{s}B_A^{-1}A_j$	\mathcal{A}	k	$\lfloor \frac{1}{\epsilon+M-1} \rfloor$
$\hat{w}_i = B_iNB_A^{-1}A_j^{-1}$	\mathcal{A}	k^2	$\lfloor \frac{k}{M} \rfloor - 1 + \epsilon$
$w = \hat{w}_jA_j$	\mathcal{B}	k^2	$\lfloor \frac{k}{M} \rfloor - 1 + \epsilon$

Table 6.3: Number of multiplication steps per RNS modular multiplication in state-of-the-art RNSMMM ([GLP⁺12] without BC correction)

Apparently, a more efficient *SR* technique that would be executed in one of two bases, as in the case of the BC operation, would dramatically reduce the total complexity and this is currently a focus of our research.

6.2.2 Architectural Study

6.2.2.1 Modular reduction by the RNS moduli

The modular reduction technique by each RNS modulus is the same used in [KKSS00, BDK01, GLP⁺12], since not only it offers simple implementations but also allows for fair comparisons. Assuming moduli of the form $\alpha_i = 2^r - c_i$, where $c_i < 2^h$ and $h < \frac{r-1}{2}$, the reduction of an integer $x < 2^{2r}$ requires two multiplications and three additions according to

$$y = x \bmod 2^r + ((x \ll r) \bmod 2^r) \cdot c_i + (x \ll 2r) \cdot c_i^2, \quad (6.17)$$

where \ll denotes a left-shift operation, $x < 2^r$, $z > 2r$, and $c_i < 2^h$ [GLP⁺12].

6.2.2.2 Conversions to/from RNS

To allow handling of large integers in each modulus channel, it is useful to employ high-radix representations so that each high-radix digit can be assigned to an RNS channel. A radix- 2^r representation of an integer x as a k -tuple $(x^{(k-1)}, \dots, x^{(0)})$ satisfies

$$x = \sum_{i=0}^{k-1} x^{(i)} 2^{ri} = \left(2^{r(k-1)}, \dots, 2^r, 1 \right) \begin{bmatrix} x^{(k-1)} \\ \vdots \\ x^{(1)} \\ x^{(0)} \end{bmatrix}, \quad (6.18)$$

Operation	Base	# multiplications	# cycles
$s = xy$	$\mathcal{A} \cup \alpha_r$	$2k + 1$	ϵ
$t = SR(s)$	$\mathcal{A} \cup \alpha_r$	$2k^2 + 7k + 2$	$\left\lceil \frac{k\epsilon}{2M} \right\rceil + 4\epsilon + 1$
$u = t\mu$	$\mathcal{A} \cup \alpha_r$	$2k + 1$	ϵ
$v = SR(u)$	$\mathcal{A} \cup \alpha_r$	$2k^2 + 7k + 2$	$\left\lceil \frac{k\epsilon}{2M} \right\rceil + 4\epsilon + 1$
$p = vN$	$\mathcal{A} \cup \alpha_r$	$2k + 1$	ϵ

Table 6.4: Number of multiplication steps in the proposed RNSBMM

where $0 \leq x^{(i)} \leq 2^r - 1$. By applying the modulo α_j operation in (6.18) we can convert the integer x to its associated RNS representation by

$$\langle x \rangle_{\alpha_j} = \left\langle \sum_{i=0}^{k-1} x^{(i)} \langle 2^{ri} \rangle_{\alpha_j} \right\rangle_{\alpha_j}, \forall j \in [1, k]. \quad (6.19)$$

If constants $\langle 2^{ri} \rangle_{\alpha_j}$ are precomputed, this computation is a typical multiply-accumulate operation and can be computed in k steps, when executed by k units in parallel.

As (2.29) is the basis of the proposed RNSBMM algorithm, it would be useful to employ it also for the RNS-to-decimal conversion. Let us rewrite (2.29) as

$$\begin{aligned} x &= \sum_{i=1}^k \langle x_i \cdot A_i^{-1} \rangle_{\alpha_i} \cdot A_i - \gamma A = \\ &= \left(2^{r(k-1)}, \dots, 2^r, 1 \right) \sum_{i=1}^k \left\{ \sigma_i \cdot \begin{bmatrix} A_{i(k-1)} \\ \vdots \\ A_{i(1)} \\ A_{i(0)} \end{bmatrix} - \gamma \begin{bmatrix} A_{(k-1)} \\ \vdots \\ A_{(1)} \\ A_{(0)} \end{bmatrix} \right\}, \end{aligned} \quad (6.20)$$

where $\sigma_i = \langle x_i \cdot A_i^{-1} \rangle_{\alpha_i}$. As soon as γ has been evaluated using the methods of section 6.1.3.1, each row of (6.20) can be computed in parallel in each cell by means of multiply-accumulate operations. In this case, carry should be propagated from cell 1 until cell k [KKSS00].

6.2.2.3 Architectural comparisons

In [NMSK01] and [GLP⁺12], cell-based architectures for implementing the algorithms in [KKSS00] and [BDK01, GLP⁺12] respectively were presented. Each cell corresponds to a single RNS modulus and utilizes a multiply-accumulate unit followed by a modular reduction unit which performs reduction by the corresponding RNS modulus using (6.17). Actually, with slight modifications, the architecture in [GLP⁺12] supports both algorithms in

[KKSS00, BDK01].

The cell structure is shown in Figure 6.5 [GLP⁺12]; a common bus that connects the cells and lines connecting one cell to a subsequent one are omitted, for simplicity reasons. The multiply-accumulate unit is depicted at the top of the cell and the modular reduction units at the bottom are a straightforward implementation of (6.17). Again, the prospective reader is instructed to refer to [GLP⁺12] for a detailed architectural analysis of the state-of-the-art RNS MM algorithms.

Most importantly, the proposed RNSBMM can also be mapped to the latest architecture in [GLP⁺12], since, as will be shown, the algorithms share many common parts. For example, both the proposed RNSBMM and the state-of-the-art RNSMMM have pure parallel multiplication or addition (subtraction) steps (for example steps 1, 3, 4, 6 of RNSBMM and steps 1, 2, 4, 5, 6 of RNSMMM in Algorithm 3.1 are identical operations).

What needs to be verified is that BC and SR operations can be mapped to the same cell architecture. Consider for example the execution of an SR operation in the cell architecture of Figure 6.5. Initially, each RNS digit x_i along with the corresponding constants A_i^{-1} and A_i stored in a ROM (not shown) are recursively multiplied and reduced in parallel in each cell, as dictated by the offset block in Figure 6.4. Each result is added to a subsequent one until β is derived serially in r -bit streams from the last cell unit in position k .

β is stored and another multiplication of the same inner products $\langle x_i \cdot A_i^{-1} \rangle_{\alpha_i}$ computed previously by $\langle A_i \rangle_{\alpha_r}$, also stored in ROM, is executed in each cell. The results are cascaded through a bus and are added recursively modulo α_r in the cell unit dedicated to the redundant modulus to compute δ . In the same cell unit γ is calculated by means of a small subtraction and multiplication modulo α_r , as dictated by (6.13).

Finally, the corresponding high-radix digits of β, γ, δ, A are combined in each unit to produce the quantity O using the same multiply-accumulate manner. The remaining subtractions by O and multiplications by $\langle 2^{-n} \rangle_{\alpha_i}$ in Figure 6.3 are carried out in parallel in each cell.

Tables 6.3 and 6.4 compare the number of multiplication steps between the latest and most

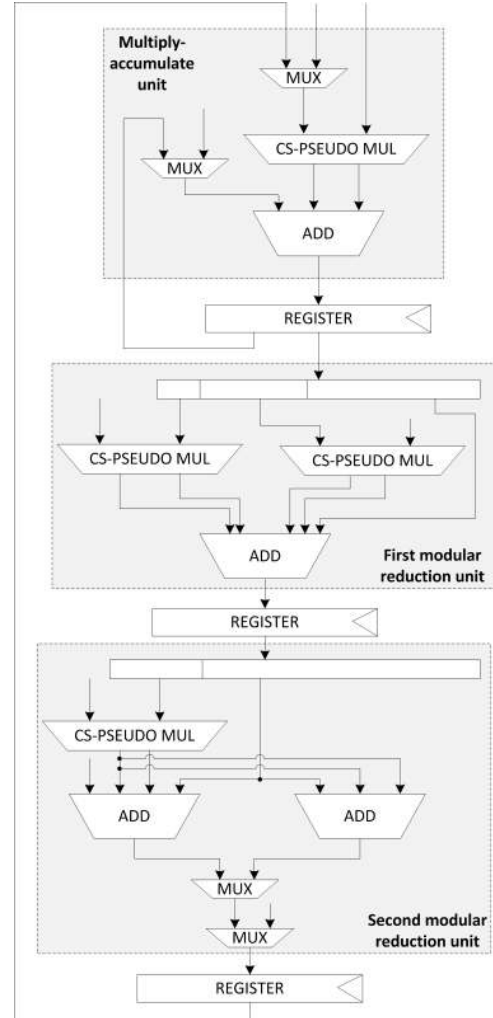


Figure 6.5: Multiply-accumulate cell architecture [GLP⁺12]

Algorithm	BC	# cycles	Cycle delay	MM delay	Expon. delay ($\times 2,050$)	Area ($\times 33$)
[KKSS00],[BDK01]	[KKSS00]	88	93	8,184	8,184	99,873
[GLP ⁺ 12]	[KKSS00]	76	93	7,068	$\cong 7,068$	99,873
[KKSS00, BI04]	[BI04]	89	86.6	7,707	7,707	99,840
[GLP ⁺ 12]	[BDK01]	77	86.6	6,669	$\cong 6,669$	99,840
This work	N/A	134	86.6	11,604.4	11,604.4	199,680

Table 6.5: Area and delay comparisons with $k = 33$, $r = 32$, $\epsilon = 3$, $M = 1$, $h = 11$

Gate	Area (transistors)	Delay (inverter)
Inverter	2	1
NAND	4	1.4
XOR	4	1.4
XNOR	12	3.2
NAND3	8	1.8
NAND4	10	2.2
REGISTER	15	4.8

Table 6.6: Basic logic library in CMOS technology (model from [Gaj97])

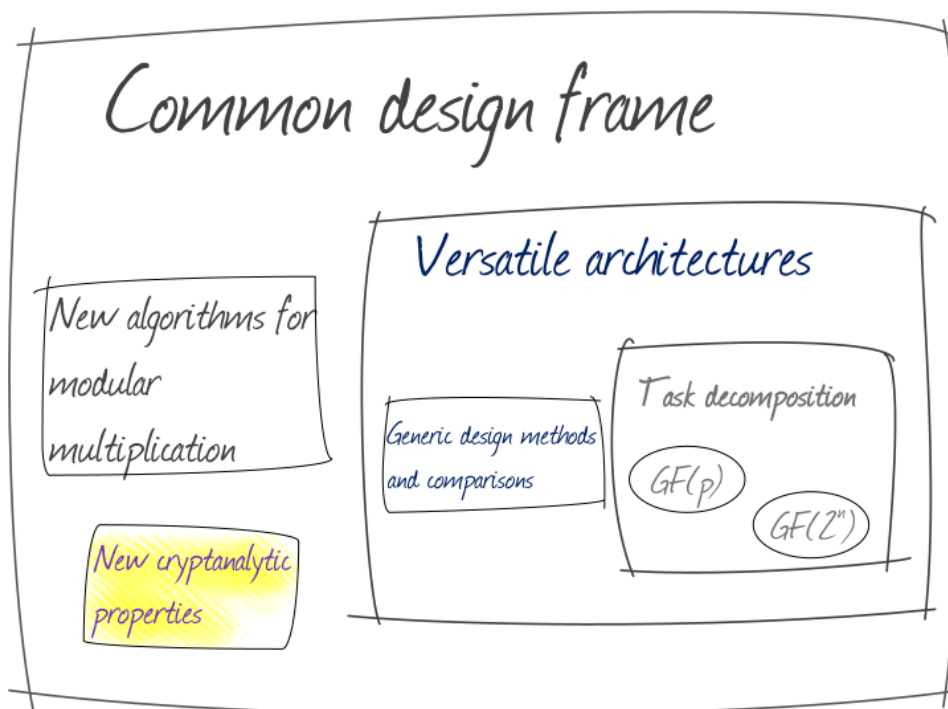
efficient architecture in [GLP⁺12] with the proposed RNSBMM. The metrics used are the same used in [GLP⁺12], i.e., the number of pipeline stages ϵ per cell, the number M of parallel multipliers per cell, and the word-length h associated with each RNS modulus. Table 6.5 summarizes complexity comparisons based on the model in Table 6.6 [Gaj97]. The table is provided in [GLMB11] which is a preliminary version of the work in [GLP⁺12]. It is assumed that the proposed RNSBMM is executed in the same architecture as [GLP⁺12], only it requires double number of cells since all computations are executed in one base of $2k$ elements. However, the total delay of the proposed RNSBMM is close to previous works, although, as in the case of total complexity, a more efficient SR technique would reduce the total delay of the proposed architecture as well.

6.3 Summary

A new algorithm for RNS modular multiplication based on Barrett’s technique was presented in this chapter. The algorithm’s validity was proven and the conditions to employ the proposed algorithm in the context of modular exponentiation were derived. Conditions for selecting the number and word-length of the RNS moduli were also provided. In the context of the proposed algorithm, methods to evaluate floor function and scaling by 2^n of an RNS number directly in RNS format were proposed. The proposed architecture requires approximately twice the number of modular multiplications compared to the state-of-the-art,

however the total delay is still competitive. Apparently, a more efficient *SR* technique that would be executed in one of two bases, as in the case of the *BC* operation, would dramatically reduce the total complexity and this is currently a focus of our research. The idea of merging both types of algorithms (*RNSBMM* and *RNSMMM*) into a common architecture was also considered.

Cryptanalysis



This chapter examines the security potentials offered by the proposed versatile hardware implementations. It attempts to prove that the use of a well-designed, residue-arithmetic, Montgomery multiplier overcomes hardware-fault attack threats, with no need to alter the basic RSA-CRT protocol while at the same time, the speed-gains offered by RSA-CRT are maintained.

7.1 Overview of side-channel attacks countermeasures

Let us briefly present some basic concepts of the RSA-CRT algorithm for easiness. In this scheme, the digital signature operation $S = M^d \pmod N$ is split in two operations $S_p = M^{d_p} \pmod p$ and $S_q = M^{d_q} \pmod q$, where $d_p = d \pmod{(p-1)}$ and $d_q = d \pmod{(q-1)}$. CRT ensures that the combination of these two values produces the signature S as

$$S = S_q + [(S_p - S_q) \cdot (q^{-1} \pmod p) \pmod p] \cdot q \quad (7.1)$$

denoted from now on as $S = CRT(S_p, S_q)$ [Knu97]. In this way, an approximate 4-time speedup of operations is achieved [Lab11b, Lab11a].

Despite this significant performance improvement, RSA-CRT was proved to be extremely vulnerable against hardware-fault attacks [ABF⁺02, Gir06, BDL01]. Assume an erroneous output generated randomly during the execution of a cryptographic operation. Without loss of generality, let the fault be in the modulus p channel, denoted as \tilde{S}_p . This will produce a faulty signature $\tilde{S} = CRT(\tilde{S}_p, S_q)$. An adversary can then factorize the public modulus n by computing its prime factor q as $q = \gcd\{(\tilde{S}^e - m) \pmod n, n\}$ and consequently obtain $p = n/q$.

In [Sha99], Shamir modified the basic RSA-CRT algorithm in (7.1) by introducing a random prime r so that $S_{pr} = m^{d \pmod{(p-1)(r-1)} \pmod{pr}$ and $S_{qr} = m^{d \pmod{(q-1)(r-1)} \pmod{qr}$. The method checks whether $S_{pr} \equiv S_{qr} \pmod r$ holds before combining them with CRT. If $S_{pr} \equiv S_{qr} \pmod r$ the computation is error-free, but the step of CRT combination is left unprotected. Moreover, Shamir's method requires the knowledge of the straightforward RSA private key d in an RSA-CRT context, which is unpractical since the key material is given in CRT format [Vig08].

The work in [ABF⁺02] exploited this weakness and broke Shamir's countermeasure. The authors proposed an improved implementation that included the protection of the CRT re-combination step. But random number generation is a problem in this scheme, since generating random numbers for each signature operation results in large time overhead.

In [Vig08], the authors proposed a method based on modulus expansion. It computes $m^d \pmod n$ in \mathbb{Z}_{Nr^2} , where r is a small random integer co-prime with n . The message m is transformed to \hat{m} such that $\hat{m} = m \pmod n$ and $\hat{m} = 1 + r \pmod{r^2}$. Then, S and \hat{S} are computed as $S = m^d \pmod n$ and $\hat{S} = \hat{m}^d \pmod{nr^2}$. If $\hat{S} \equiv S \pmod n$ then the protocol is error-free. However, the method did not improve much the performance overhead [MLW12].

The work in [Gir06] exploited the Montgomery Ladder Exponentiation (MLE) algorithm as a countermeasure scheme. Unlike the square-and-multiply algorithm which performs on average 1.5 modular multiplications per bit of the exponent, the MLE algorithm performs two modular multiplications for each bit of exponent, and thereby increases execution time.

The authors in [YJ00] provided an ingenious fault-attack based on the *safe-error* concept. They observed that during a modular exponentiation using typical Square and Multiply algorithms, if the exponent bit is 0 then the result of a modular multiplication is not used. By inducing an error during multiplication and by testing whether the result is correct or

not the attacker can deduce the bit of the secret exponent. However, a countermeasure was provided in [JY02] using MLE.

In [YKLM03, BOS03] a class of countermeasures based on “fault-infective” techniques are introduced. They are based on the idea of modifying the RSA-CRT computations in such a way that a faulty signature in one channel will infect the final signature after the CRT recombination. Unfortunately, like in [Sha99], not only the knowledge of d is required, but also the techniques rely on some very strong assumptions. For example, some parameters t_1, t_2 introduced in [BOS03] require that $\gcd(t_1, t_2) = \gcd(d, \varphi(t_1)) = \gcd(d, \varphi(t_2)) = 1$, where φ is the Euler’s totient function. t_1, t_2 should normally be generated once along with the RSA key and the same values should be used throughout the lifetime of the key. However, these values cannot be stored in such a personalized context, meaning that the generation of t_1, t_2 for each signature is not a negligible computational task.

7.2 Fault handling in RNS-based multipliers

The majority of the aforementioned countermeasures are based on modifications in the RSA-CRT protocol, which amount to extra operations and increased algorithmic complexity for the RSA-CRT execution. These solutions rely on the 2-modulus splitting of RSA calculations using a naive RNS consisting of just the moduli p and q . In the following, we deviate from 2-modulus splitting, and the multi-modulus RNS Montgomery multipliers presented in the previous chapters are examined from a hardware-fault tolerance point of view.

7.2.1 Hardware-fault tolerance in MRC-based RNS Montgomery multipliers

To simplify our discussion, the RNSMMM algorithm along with the MRC-based BC are presented for reference as Algorithm 7.1 and 7.2 respectively (see Section 4.1 and 4.2).

Algorithm 7.1 RNS Montgomery Modular Multiplication (RNSMMM)

Input: $a_{\mathcal{T}}, b_{\mathcal{T}} \{ a, b < 2N \}$

Output: $c_{\mathcal{T}}, \{ c < 2N \text{ and } c \equiv abQ^{-1} \pmod{N} \}$

Precompute: $(-N^{-1})_{\mathcal{B}}, Q_{\mathcal{A}}^{-1}, N_{\mathcal{A}}$

- 1 $s_{\mathcal{T}} \leftarrow a_{\mathcal{T}} \cdot b_{\mathcal{T}}$
 - 2 $t_{\mathcal{B}} \leftarrow s_{\mathcal{B}} \cdot (-N^{-1})_{\mathcal{B}}$
 - 3 $t_{\mathcal{A}} \leftarrow t_{\mathcal{B}} \{ \text{base conversion step} \}$
 - 4 $u_{\mathcal{A}} \leftarrow t_{\mathcal{A}} \cdot N_{\mathcal{A}}$
 - 5 $v_{\mathcal{A}} \leftarrow s_{\mathcal{A}} + u_{\mathcal{A}}$
 - 6 $c_{\mathcal{A}} \leftarrow v_{\mathcal{A}} \cdot Q_{\mathcal{A}}^{-1}$
 - 7 $c_{\mathcal{B}} \leftarrow c_{\mathcal{A}} \{ \text{base conversion step} \}$
-

It is apparent that steps 1,2,4,5,6 in Algorithm 7.1 are performed in parallel in each modulus channel. Clearly, if the algorithm was completely parallel, an error in modulus channel i would not influence the rest channels and thus the Greatest Common Divisor (GCD) attack

Algorithm 7.2 MRC-based base conversion (see section 4.2)

Input: $x_B = (x_1, x_2, \dots, x_L)$
Output: $x_A = (x'_1, x'_2, \dots, x'_L)$

- 1 $U_1 \leftarrow x_1$
- 2 **for all** $i = 2, \dots, L$ **do**
- 3 $U_i \leftarrow x_i$
- 4 **for** $j = 1$ to $i - 1$ **do**
- 5 $U_i \leftarrow \langle (U_i - U_j) q_{j,i}^{-1} \rangle_{q_i}$
- 6 **end for**
- 7 **end for**
- 8 **for all** $i = 1, \dots, L$ **do**
- 9 $x'_i \leftarrow \langle U_L \rangle_{p_i}$
- 10 **for** $j = L - 1$ to 1 **do**
- 11 $x'_i \leftarrow \langle x'_i q_j + U_j \rangle_{p_i}$
- 12 **end for**
- 13 **end for**

would hold. We prove that the base conversion provides the desired mechanism for fault tolerance.

Assume a permanent error \tilde{t}_i in modulus channel $1 \leq i \leq L$. Note that, since step 2 of Algorithm 7.1 uses the result of step 1, the faulty result will always amount to \tilde{t}_i . By observation, employing \tilde{t}_i in the base conversion of step 3 yields

$$\tilde{t}_i \leftarrow \langle (\tilde{t}_i - t_j) q_{j,i}^{-1} \rangle_{q_i}, i \in [2, L], \forall j \in [1, i - 1]. \quad (7.2)$$

(7.2) corresponds to the steps 1-7 of Algorithm 7.2 and implies that an error occurred in position i , will always cascade to next channels and produce a faulty \tilde{t}_L even if the error occurs at the very last step of calculations in channel L . This value is used in step 9 to continue the base conversion process. An important observation is that at this step, the faulty \tilde{t}_L is injected to all channels according to

$$\tilde{t}'_i \leftarrow \langle \tilde{t}_L \rangle_{p_i}, \forall i \in [1, L] \quad (7.3)$$

and similarly the faulty \tilde{t}'_i s produce

$$\tilde{t}'_i \leftarrow \langle \tilde{t}'_i \cdot q_j + \tilde{t}_j \rangle_{p_i}, \forall i \in [1, L], \forall j \in [1, L - 1]. \quad (7.4)$$

As a result, a faulty \tilde{t}_A is generated at step 3 of Algorithm 7.2 and injected in step 4 for subsequent calculations. Note that due to (7.3), it is assured that all channels after the first base conversion will be infected. Using a similar analysis, it is easy to show that even if the error occurs after the first base conversion, the second base conversion at step 7 of Algorithm 7.1 will infect all channels in the same manner, thus making the GCD attack infeasible.

A special case is when the error is not permanent and is inserted in a channel i , $i \in [1, L]$ during the base conversion. If the error is generated during steps 1-7 of Algorithm 7.2, step

9 will inject the error to all other channels, according to (7.3). The case that an error is inserted in channel $i, i \in [1, L]$ during step 11 of Algorithm 7.2 should also be examined. Although step 11 is executed in parallel for all channels, each channel calculation reuses the results from all other channels. This is also apparent from the recursive form of (7.3). Due to this, all channels are affected making GCD attack infeasible. A similar analysis may be conducted for the MRC-based BC in section 5.1.

7.2.2 Hardware-fault tolerance in CRT-based RNS Montgomery multipliers

In [KKSS00], the first practical and efficient implementation of RNS Montgomery multiplier based on CRT was presented. The CRT-based algorithm for RNSMMM is identical to Algorithm 7.1, thus only the BC is presented below as Algorithm 7.3.

Algorithm 7.3 Base Conversion (BC) algorithm by Kawamura et al. [KKSS00]

Input: $\zeta_{\mathcal{B}} = (\zeta_1, \zeta_2, \dots, \zeta_L), A, B, \alpha$
Output: $\zeta_{\mathcal{A}} = (\zeta'_1, \zeta'_2, \dots, \zeta'_L)$
Precompute: $(B_i^{-1})_{q_i}, (B_i)_{\mathcal{A}} (\forall i = 1 \dots L), (-B)_{\mathcal{A}}$

- 1 $\sigma_0 = \alpha$
- 2 **for all** $i = 1 \dots L$ **do**
- 3 $\xi_i = \langle \zeta_i \cdot B_i^{-1} \rangle_{q_i}$
- 4 $\delta_{i,0} = 0$
- 5 **end for**
- 6 **for all** $i = 1 \dots L$ **do**
- 7 **for** $j = 1 \dots L$ **do**
- 8 $\sigma_j = \sigma_{(j-1)} + \text{trunc}(\xi_j) / 2^r$
- 9 $\gamma_j^* = \lfloor \sigma_j \rfloor, \{\gamma_j^* = \{0, 1\}\}$
- 10 $\sigma_j = \sigma_j - \gamma_j^*$
- 11 $\delta_{i,j} = \delta_{i,(j-1)} + \xi_j \cdot \langle B_j \rangle_{p_i} + \gamma_j^* \cdot \langle -B \rangle_{p_i}$
- 12 **end for**
- 13 **end for**
- 14 **for all** $i = 1 \dots L$ **do**
- 15 $\zeta'_i = \langle \delta_{i,L} \rangle_{p_i}$
- 16 **end for**

Clearly, steps 1-5 and 14-16 in Algorithm 7.3 involve completely parallel operations in all channels, so fault-tolerance should be examined for the steps 6-13. In the case of a permanent error, a faulty $\tilde{\gamma}_j^*, j \in [1, L]$ is generated in steps 8-9 which consequently produces

$$\tilde{\delta}_{i,j} = \delta_{i,(j-1)} + \xi_j \cdot \langle B_j \rangle_{p_i} + \tilde{\gamma}_j^* \cdot \langle (-B) \rangle_{p_i}, \forall i, j \in [1, L]. \quad (7.5)$$

This means that all channels are affected by the error, thus the parallel operation of steps 14-16 is also affected.

In the case of an error induced in a timing manner, issues are raised. It is apparent that, if an adversary is able to insert an error during the steps 14-16, only one (or several) channels

can be affected, which makes the GCD attack easily mountable. To overcome this issue, an extra checking procedure is inserted in steps 14-16 of Algorithm 7.3 based on the following pseudo code:

```
1 for all  $i = 1, \dots, L$  do
2   if  $\delta_{i,L} == \delta_{i,L}$  of step 11 then
3      $\zeta'_i = \langle \delta_{i,L} \rangle_{p_i}$ 
4   else
5     error detected
6   end if
7 end for
```

The solution checks whether the quantities $\delta_{i,L}$ are identical to the values obtained in the previous step 11 and if not, a malicious error has been detected. The solution requires the storage of the L values of step 11 and a comparison with the $\delta_{i,L}$ s employed in step 15. Note that this solution does not issue significant overhead since the checking procedure can be executed only once at the end of an RSA exponentiation.

7.2.3 Remarks on Performance

Presenting performance metrics of RNS Montgomery multipliers is out of scope of this chapter. We have already presented comparative studies in chapters 4, 5 and in [SS14], while trade-offs between state-of-the art RNS solutions appear in [GLP⁺12].

There is, however, an important derivative of the presented hardware-fault analysis on RNS Montgomery multipliers. As described in Section 7.1, current countermeasures appearing in the literature provide immunity at the cost of extra operations or checking procedures in the RSA-CRT protocol itself, thus the 4-time speedup offered by the use of RSA-CRT is somehow sacrificed to achieve tolerance against hardware-fault attacks.

The presented analysis shows that if RNS Montgomery multipliers are employed, instead of typical non-RNS ones in crypto-hardware design, security is offered for free, with no need for extra checking procedures or modifications to the RSA-CRT protocol as in [Sha99, Vig08, ABF⁺02, MLW12, Gir06, YJ00, YKLM03, BOS03]. At the same time, since immunity comes for free, the 4-time speedups between RSA and RSA-CRT are maintained.

7.3 Summary

The cryptanalytic properties of RNS-based Montgomery multipliers against hardware fault attacks for RSA-CRT were analyzed. It was proved that, in contrast to previous solutions based on modifications to the basic RSA-CRT protocol, the use of MRC-based RNS Montgomery multipliers [SS14], is sufficient to provide security against such attacks with no need for modifications in protocol level. Weaknesses of CRT-based multipliers were also identified and countermeasures were proposed. The contribution of the base-conversion

process as the inherent mechanism of RNS-based Montgomery multipliers responsible for hardware-fault immunity was proved.

CHAPTER

8

Conclusions and Outlook

Being exposed in an unprecedented number of threats and frauds, safe connectivity for all network-based systems has become a predicate necessity. Cryptographic hardware plays a dominant role in the implementation of systems that could offer the desired levels of security. The prospective crypto-hardware designer should not only care for performance but also resistance against attacks. Under this perspective, cryptographic hardware design poses extra difficulties and challenges considering especially the fact that, as years pass by, the security standards need to be constantly strengthened.

This doctoral thesis attempted to approach the problem of cryptographic hardware design in a holistic manner, covering the aspects of new algorithms proposal, algorithmic analysis, mathematical validation, crypto-hardware design, and security validation of the proposed architectures. The proposed algorithms and architectures made use of the non-conventional representation of RNS and PRNS.

Initially, the first practical implementation of an elliptic curve processor using the RNS representation was presented. We approached the problem by evaluating an appropriate range for the calculations, and new task execution graphs for point doubling and point addition were proposed. The tasks were optimized to be resistant against power and timing attacks. The idea was further enhanced and more efficient designs based on pipelined RNS structures and moduli of special form were also proposed.

Next, an important class of algorithms that formed the basis of the proposed versatile architectures was presented, namely the RNSMMM and PRNSMMM algorithms. The most important features and characteristics of these algorithms were thoroughly analyzed. New, improved versions for both algorithms were proposed, while algorithmic and architectural analysis proved the superiority of the proposed solutions compared to existing ones.

The design methodology for incorporating RNS and PRNS in MMM in $GF(p)$ or $GF(2^n)$ respectively was subsequently presented. An analysis of input/output conversions to/from residue representation, along with the proposed residue Montgomery multiplication algorithm, revealed common multiply-accumulate data paths both between the converters and between the two residue representations. A novel versatile architecture was derived that supports all operations of MM in $GF(p)$ and $GF(2^n)$, input/output conversions, MRC for integers and polynomials, dual-field modular exponentiation and inversion in the same hardware. Detailed comparisons with state-of-the-art implementations proved the potential of residue arithmetic exploitation in dual-field modular multiplication.

Furthermore, one of the fundamental problems in VLSI design was considered, that is the problem of evaluating and comparing architectures using models independent from the underlying fabrication technology. Generic, function-based methods to evaluate the optimal operation parameters of the proposed architectures as well as methodologies to optimize the proposed architectures in terms of speed, area, or area \times speed based on the needs of the underlying application were proposed.

An important security property of the proposed residue arithmetic architectures was also revealed. It was proved that the use of a well-designed, residue-arithmetic, Montgomery

multiplier overcomes hardware-fault attack threats, with no need to alter the basic RSA-CRT protocol while at the same time, the speed-gains offered by RSA-CRT are maintained.

Finally, novel algorithms based on new mathematical and design problems for the crucial operation of modular multiplication using Barrett's technique were presented. The algorithms preserve the versatile characteristics discussed previously and it was proved that, along with existing algorithms in the literature, a large family of algorithms applicable in cryptography may be formed, unified under the common frame of the proposed versatile architectures.

Several directions towards future research are still left open. The proposed function-based methodologies for generic comparisons could be enhanced to include more design parameters, like the fan-out factor of logic gates, thus more detailed and accurate models could be derived. Equivalent models could also be devised to analyze and compare power consumption, in the same sense that area, speed and area \times speed product were compared in this thesis.

Hardware evaluation of the new RNSBMM algorithm should also be considered either by employing the proposed multiply-accumulate architectures, or RNS-specific architectures using special moduli sets like the ones proposed in Chapter 3. Optimization potentials for the SR operations employed in RNSBMM should also be investigated.

Moduli of special form, like the ones employed in Chapter 3, should be evaluated in the context of the proposed MAC architecture for RNSMMM and PRNSMMM. The possibility of replacing the proposed arithmetic circuits with more specific ones in order to further simplify and optimize the proposed architectures should be considered.

New parallelization prospects offered by state-of-the-art multi-processor systems could also be investigated. A possible scenario could be that parallel processors perform parallel multiplications on different data-sets of a single message. In such cases, the existence of equivalencies between a serial and a parallel algorithm, if any, should be mathematically proven. Also, in case parallelization is possible, any required algorithmic overhead should be carefully determined and assessed for performance impact. There are various design issues when it comes to multi-processor system design. A careful examination of the impact of interconnections, system I/O delays, etc., on the system's performance and area should be carried out. It should be also considered that these systems require full availability of all input data beforehand, which does not allow for real-time encryption/signing.

Finally, the cryptanalytic properties of RNS-based architectures can also be further extended, to include attacks other than hardware-fault related. The role of BC and SR operations should be meticulously analyzed to reveal new possibilities for cryptanalytic resistance. An interesting derivative of this thesis is the security potential offered by the proposed versatile architectures, by means of changing seamlessly the underlying cryptographic protocols during an established communication channel. Investigating the applicability of RNS to other PKC systems, like for example the emerging lattice-based cryptography [GGH97, Reg06], could also generate new and interesting cryptanalytic properties, architectures and algorithms.

In general, current solutions employing conventional binary arithmetic for modular multiplication, are based on Montgomery's algorithm (systolic, semi-systolic, etc). These architectures have been extensively analyzed and the optimizations proposed are so fine-grained, that the research space on the field steadily narrows. On the other hand, this doctoral thesis provided solid indications that non-conventional arithmetic like RNS and PRNS may provide new means for tackling design problems of crypto-hardware and further extend the research space in this active field.



Bibliography

- [ABF⁺02] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Counter-Measures. In *Proc. Int. Workshop Cryptographic Hardware and Embedded Systems (CHES '02)*, pages 260–275, 2002.
- [AH93] I. O. Aichholzer and H. Hassler. A fast method for modulus reduction and scaling in residue number system. In *Workshop on Economic Parallel Processing (EPP'93)*, pages 40–56, 1993.
- [AMV93] G.B. Agnew, R.C. Mullin, and S.A. Vanstone. An implementation of elliptic curve cryptosystems over $F_{2^{155}}$. *IEEE J. Sel. Areas Commun.*, 11(5):804–813, jun 1993.
- [Bar87] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology, CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer Berlin Heidelberg, 1987.
- [BCJL93] G. Brassard, C. Crepeau, R. Jozsa, and D. Langlois. A quantum bit commitment scheme provably unbreakable by both parties. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 362–371, 1993.
- [BDEM06] J. C. Bajard, Sylvain Duquesne, M. Ercegovac, and N. Meloni. Residue systems efficiency for modular products summation: application to elliptic curves cryptography. In *Proc. Advanced Signal Processing Algorithms, Architectures, and Implementations XVI*, volume 6313, August 2006.
- [BDK98] J. Bajard, L.-S. Didier, and P. Kornerup. An RNS Montgomery modular multiplication algorithm. *Computers, IEEE Transactions on*, 47(7):766–776, 1998.

- [BDK01] J.-C. Bajard, L. S. Didier, and P. Kornerup. Modular multiplication and base extensions in residue number systems. In *Proceedings of the 15th Symposium on Computer Arithmetic*, ARITH '01, pages 59–65, 2001.
- [BDL01] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14:101–119, 2001.
- [BI04] Jean-Claude Bajard and Laurent Imbert. A full RNS implementation of RSA. *IEEE Transactions on Computers*, 53:769–774, June 2004.
- [BIJ05] Jean-Claude Bajard, Laurent Imbert, and Graham A. Jullien. Parallel Montgomery Multiplication in $GF(2^k)$ Using Trinomial Residue Arithmetic. *Computer Arithmetic, IEEE Symposium on*, 0:164–171, 2005. doi:10.1109/ARITH.2005.34.
- [BKP09] J.C. Bajard, M. Kaihara, and T. Plantard. Selected RNS Bases for Modular Multiplication. In *19th IEEE International Symposium on Computer Arithmetic*, pages 25–32, 2009.
- [BL06] J. Buček and R. Lrencz. Comparing subtraction-free and traditional AMI. In *Proc. IEEE Design and Diagnostics of Electronic Circuits and systems, DDECS'06*, pages 95–97, April 2006.
- [BOS03] Johannes Blmer, Martin Otto, and Jean-Pierre Seifert. A new CRT-RSA algorithm secure against bellcore attacks. In *Proceedings of the 10th ACM conference on Computer and communications security, CCS '03*, pages 311–320, 2003.
- [BSS02] I. Blake, G. Seroussi, and N. Smart. *Elliptic curves in cryptography*. Cambridge University Press, 2002.
- [CBC07] G. Chen, G. Bai, and H. Chen. 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 II: Express Briefs*, 54(5):412–416, May 2007.
- [DBS06] Jean-Pierre Deschamps, Gery J. A. Bioul, and Gustavo D. Sutter. *Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems*. John Wiley & Sons, Hoboken, New Jersey, 2006.
- [dDBQ04] G. M. de Dormale, P. Bulens, and J.-J. Quisquater. An improved Montgomery modular inversion targeted for efficient implementation on FPGA. In *Proc. IEEE Int. Conf. Field Programm (FPT'04)*, pages 441–444, December 2004.
- [Des09] Jean-Pierre Deschamps. *Hardware Implementation of Finite-Field Arithmetic*. McGraw-Hill, Inc., New York, NY, USA, 2009.

- [DFSS05] I.B. Damgard, S. Fehr, L. Salvail, and C. Schaffner. Cryptography in the bounded quantum-storage model. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 449–458, 2005. doi:10.1109/SFCS.2005.30.
- [DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.
- [DMKP04] A. Daly, W. Marnane, T. Kerins, and E. Popovici. An FPGA implementation of a $GF(p)$ ALU for encryption processors. *Microprocessors and Microsystems*, 28(5-6):253–260, August 2004.
- [DMP03] A. Daly, L. Marnane, and E. Popovici. Fast modular inversion in the Montgomery domain on reconfigurable logic. *Technical report: University College Cork, Ireland*, 2003.
- [EL04] M. Ercegovic and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Publishers, San Francisco, 2004.
- [Elg85] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, 31(4):469–472, 1985. doi:10.1109/TIT.1985.1057074.
- [ESG⁺05] H. Eberle, S. Shantz, V. Gupta, N. Gura, L. Rarick, and L. Spracklen. Accelerating next-generation public-key cryptosystems on general-purpose CPUs. *IEEE Micro*, 25(2):52–59, March-April 2005.
- [ESJ⁺13] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi. Efficient RNS Implementation of Elliptic Curve Point Multiplication Over $GF(p)$. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(21):1545–1549, 2013.
- [FP99] W. L. Freking and K. K. Parhi. A unified method for iterative computation of modular multiplication and reduction operations. In *Proc. IEEE International Conference on Computer Design (ICCD'99)*, pages 80–87, October 1999.
- [Gaj97] D. Gajski. *Principles of Digital Design*. Prentice-Hall, 1997.
- [GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Jr. Kaliski, editor, *Advances in Cryptology CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer Berlin Heidelberg, 1997. URL: <http://dx.doi.org/10.1007/BFb0052231>, doi:10.1007/BFb0052231.

- [Gir06] C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, Sept. 2006.
- [GLMB11] F. Gandino, F. Lamberti, P. Montuschi, and J. Bajard. A General Approach for Improving RNS Montgomery Exponentiation Using Pre-processing. In *Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on*, pages 195–204, July 2011.
- [GLP⁺12] F. Gandino, F. Lamberti, G. Paravati, J.-C. Bajard, and P. Montuschi. An Algorithmic and Architectural Study on Montgomery Exponentiation in RNS. *Computers, IEEE Transactions on*, 61(8):1071–1083, Aug. 2012.
- [Gro01] Johann Großschädl. A Bit-Serial Unified Multiplier Architecture for Finite Fields $GF(p)$ and $GF(2^m)$. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems, CHES '01*, pages 202–219, London, UK, 2001. Springer-Verlag.
- [GT03] A. A. A Gutub and A.F. Tenca. Efficient scalable hardware architecture for Montgomery inverse computation in $GF(p)$. In *Proc. IEEE Workshop Signal Process. Syst. (SIPS'03)*, pages 93–98, August 2003.
- [GTK02] A. A.-A. Gutub, A. Tenca, and Ç. K. Koç. Scalable VLSI architecture for $GF(p)$ Montgomery modular inverse computation. In *Proc. IEEE Comput. Soc. Annual Symp. VLSI*, pages 53–58, April 2002.
- [Gui10] Nicolas Guillermin. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over F_p . In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 48–64. Springer Berlin / Heidelberg, 2010.
- [HGEG11] Miaoqing Huang, K. Gaj, and T. El-Ghazawi. New Hardware Architectures for Montgomery Modular Multiplication Algorithm. *IEEE Transactions on Computers*, 60(7):923–936, July 2011.
- [HGG07] William Hasenplaugh, Gunnar Gaubatz, and Vinodh Gopal. Fast modular reduction. In *Proc. 18th IEEE Int. Symposium on Computer Arithmetic (ARITH'07)*, pages 225–229, June 2007.
- [HKA⁺05] David Harris, Ram Krishnamurthy, Mark Anders, Sanu Mathew, and Steven Hsu. An Improved Unified Scalable Radix-2 Montgomery Multiplier. *Computer Arithmetic, IEEE Symposium on*, 0:172–178, 2005.
- [HMV04] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to elliptic curves cryptography*. Springer-Verlag & Hall/CRC, New York, 2004.

- [JY02] M. Joye and S.-M. Yen. The Montgomery powering ladder. In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES'02) LNCS*, pages 291–302, 2002.
- [KA98] Ç. K. Koç and T. Acar. Montgomery Multiplication in $GF(2^k)$. *Design, Codes and Cryptography*, 14(1):57–69, April 1998.
- [KAK96] Ç. K. Koç, T. Acar, and B. S. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
- [Kal] B. Kaliski. TWIRL and RSA Key Size. URL: <http://www.rsasecurity.com/rsalabs/node.asp?id=2004>.
- [Kal95] B. S. Kaliski. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
- [KF07] Hakim Khali and Ahcene Farah. Cost-Effective Implementations of $GF(p)$ Elliptic Curve Cryptography Computations. *International Journal of Computer Science and Network Security*, 7(8):29–37, August 2007.
- [KH98] Q. K. Kop and C. Y. Hung. Fast algorithm for modular reduction. In *Proc. IEEE Computer and Digital Techniques*, volume 145(4), pages 265–271, July 1998.
- [KKSS00] Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-Rower architecture for fast parallel Montgomery multiplication. In *EUROCRYPT'00: Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, pages 523–538, Berlin, Heidelberg, 2000. Springer-Verlag.
- [Knu97] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
- [Kob87] N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48:203–209, 1987.
- [KPH04] Hyun-Sung Kim, Hee-Joo Park, and Sung-Ho Hwang. Parallel Modular Multiplication Algorithm in Residue Number System. In *Proc. Workshop on High Performance Numerical Algorithms LNCS*, volume 3019, pages 1028–1033, April 2004.
- [Lab11a] RSA Lab. High-Speed RSA Implementation, 2011. URL: <ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf>.
- [Lab11b] RSA Lab. RSA Hardware Implementation, 2011. URL: <ftp://ftp.rsasecurity.com/pub/pdfs/tr801.pdf>.

- [LH08a] Jyu-Yuan Lai and Chih-Tsun Huang. Elixir: High-Throughput Cost-Effective Dual-Field Processors and the Design Framework for Elliptic Curve Cryptography. *IEEE Transactions on VLSI*, 16(11):1567–1580, nov. 2008.
- [LH08b] Ralf Laue and Sorin A. Huss. Parallel Memory Architecture for Elliptic Curve Cryptography over $GF(p)$ Aimed at Efficient FPGA Implementation. *Journal of Signal Processing Systems*, 51:39–55, 2008.
- [LN86] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, New York, NY, USA, 1986.
- [MBS03] U. Meyer-Base and T. Stouraitis. New power-of-2 RNS scaling scheme for cell-based IC design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(2):280–283, april 2003.
- [McE87] Robert J. McEliece. *Finite field for scientists and engineers*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.
- [Mic94] G. De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Inc., Singapore, 1994.
- [Mil86] V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology (CRYPTO'85) LNCS*, volume 218, pages 47–426, 1986.
- [MLW12] Kun Ma, Han Liang, and Kaijie Wu. Homomorphic Property-Based Concurrent Error Detection of RSA: A Countermeasure to Fault Attack. *Computers, IEEE Transactions on*, 61(7):1040–1049, july 2012.
- [MMM04] C. McIvor, M. McLoone, and J.V. McCanny. Modified Montgomery modular multiplication and RSA exponentiation techniques. *Computers and Digital Techniques, IEE Proceedings -*, 151(6):402–408, Nov. 2004.
- [MMM06] C. J. McIvor, M. McLoone, and J. V. McCanny. Hardware elliptic curve cryptographic processor over $GF(p)$. *IEEE Transactions on Circuits and Systems-Part I*, 53(9):1946–1957, September 2006.
- [Moh07] P.V.A. Mohan. RNS-To-Binary Converter for a New Three-Moduli Set $\{2^{n+1} - 1, 2^n, 2^n - 1\}$. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 54(9):775–779, 2007.
- [Mon85] P. L. Montgomery. Modular multiplication without trial division. *Math. Comput.*, 16:519–521, 1985.
- [MOVW88] R. Mullin, I. Onyszchuk, S. Vanstone, and R. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22:149–161, 1988.

- [MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.
- [NME11] K. Navi, A.S. Molahosseini, and M. Esmaeildoust. How to Teach Residue Number System to Computer Scientists and Engineers. *Education, IEEE Transactions on*, 54(1):156–163, 2011.
- [NMSK01] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura. Implementation of RSA Algorithm Based on RNS Montgomery Multiplication. In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES'01) LNCS*, volume 2162, pages 364–376, 2001.
- [OBPV03] Siddika Berna Örs, Lejla Batina, Bart Preneel, and Joos Vandewalle. Hardware implementation of an elliptic curve processor over $GF(p)$. In *Proc. IEEE Application-Specific Systems, Architectures, and Processors (ASAP'03)*, pages 433–443, June 2003.
- [OM86] J. Omura and J. Massey. Computational method and apparatus for finite field arithmetic, May 1986.
- [OP01] G. Orlando and C. Paar. A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware. In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES'01) LNCS*, volume 2162, pages 348–363, 2001.
- [Par97] B. Parhami. Modular reduction by multi-level table lookup. In *Proc. IEEE 40th Midwest Symposium on Circuits and Systems*, volume 1, pages 381–384, August 1997.
- [PH08] Nathaniel Ross Pinckney and David Money Harris. Parallelized radix-4 scalable Montgomery multipliers. *Integrated Circuits and Systems*, 3(1):39–45, nov. 2008.
- [Pie95] S. J. Piestrak. A high-speed realization of a residue to binary number system converter. *IEEE Transactions on Circuits and Systems II, Analog and Digital Signal Processing*, 42:661–663, October 1995.
- [PP95] K. Posch and R. Posch. Modulo reduction in residue number systems. *Trans. Parallel Distrib. Syst.*, 6(5):449–454, 1995.
- [Reg06] Oded Regev. Lattice-based cryptography. In *Advances in Cryptology CRYPTO '06*, Lecture Notes in Computer Science, pages 131–141. Springer Berlin Heidelberg, 2006.

- [RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for Obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, February 1978.
- [SBC98] T. Srikanthan, M. Bhardwaj, and C. T. Clarke. Area-time-efficient VLSI residue-to-binary converters. *IEE Proceedings Computers and Digital Techniques*, 145:229–235, May 1998.
- [SCWL08] Ming-Der Shieh, Jun-Hong Chen, Hao-Hsuan Wu, and Wen-Ching Lin. A new modular exponentiation architecture for efficient design of RSA cryptosystem. *IEEE Transactions on VLSI*, 16:1151–1161, September 2008.
- [SFKS06] D. M. Schinianakis, A. P. Fournaris, A. P. Kakarountas, and T. Stouraitis. An RNS architecture of an $F(p)$ elliptic curve point multiplier. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'06)*, pages 3369–3373, May 2006.
- [SFM⁺09] D.M. Schinianakis, A.P. Fournaris, H.E. Michail, A.P. Kakarountas, and T. Stouraitis. An RNS Implementation of an F_p Elliptic Curve Point Multiplier. *IEEE Transactions on Circuits and Systems I*, 56(6):1202–1213, jun. 2009.
- [Sha99] A. Shamir. Improved Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks, Nov 1999.
- [SK89] M.A.P. Shenoy and R. Kumaresan. A fast and accurate RNS scaling technique for high speed signal processing. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(6):929–937, jun 1989.
- [SK00] E. Savas and Ç. K. Koç. The Montgomery modular inverse-Revisited. *IEEE Transactions on Computers*, 49(7):763–766, July 2000.
- [SKS06] D. M. Schinianakis, A. P. Kakarountas, and T. Stouraitis. A new approach to elliptic curve cryptography: an RNS architecture. In *Proc. IEEE Mediterranean Electrotechnical Conference MELECON'06*, pages 1241–1245, May 2006.
- [SKS07] M. Sudhakar, R.V. Kamala, and M.B. Srinivas. A bit-sliced, scalable and unified Montgomery multiplier architecture for RSA and ECC. In *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on*, pages 252–257, oct. 2007. doi:10.1109/VLSISOC.2007.4402507.
- [SL10] Ming-Der Shieh and Wen-Ching Lin. Word-Based Montgomery Modular Multiplication Algorithm for Low-Latency Scalable Architectures. *IEEE Transactions on Computers*, 59(8):1145–1151, aug. 2010. doi:10.1109/TC.2010.72.

- [SMB⁺07] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede. Reconfigurable modular arithmetic logic unit supporting high-performance RSA and ECC over $GF(p)$. *International Journal of Electronics*, 94(5):501–514, May 2007.
- [SS11] D. Schinianakis and T. Stouraitis. A RNS Montgomery multiplication architecture. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 1167–1170, May 2011.
- [SS13] D. Schinianakis and T. Stouraitis. An RNS modular multiplication algorithm. In *Circuits, Electronics and Systems (ICECS), 2013 IEEE International Conference on*, 2013.
- [SS14] D. Schinianakis and T. Stouraitis. Multifunction residue architectures for cryptography. *accepted for publication in IEEE Transactions on Circuits and Systems I*, 2014.
- [SSS12] D. Schinianakis, A. Skavantzios, and T. Stouraitis. $GF(2^n)$ Montgomery multiplication using Polynomial Residue Arithmetic. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 3033–3036, 2012.
- [ST67] N. Szabo and R. Tanaka. Residue Arithmetic and its Applications to Computer Technology. *New York: McGraw-Hill*, 1967.
- [ST03] Akashi Satoh and Kohji Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52:449–460, April 2003.
- [STK00] Erkey Savaş, Alexandre Tenca, and Çetin Koç. A Scalable and Unified Multiplier Architecture for Finite Fields $GF(p)$ and $GF(2^m)$. In *Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 277–292. Springer Verlag, 2000.
- [Tay88] F. J. Taylor. Residue arithmetic: a tutorial with examples. *IEEE Computer*, 17:50–62, May 1988.
- [TjZbXHQj10] Yang Tong-jie, Dai Zi-bin, Yang Xiao-Hui, and Zhao Qian-jin. An improved RNS Montgomery modular multiplier. In *Computer Application and System Modeling (ICCA SM), 2010 International Conference on*, volume 10, pages 144–147, 2010.
- [TK03] A.F. Tenca and Ç.K. Koç. A scalable architecture for modular multiplication based on Montgomery’s algorithm. *IEEE Transactions on Computers*, 52(9):1215–1221, sep. 2003.
- [TT04] L. A. Tawalbeh and A. Tenca. An algorithm and hardware architecture for integrated modular division and multiplication in $GF(p)$. In *Proc. IEEE Int.*

Conf. Application-Specific Systems, Architectures, and Processors (ASAP'04), pages 247–257, September 2004.

- [US 00] US National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), 2000. URL: <http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>.
- [Vig08] D. Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In *Proc. Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 08)*, pages 130–145, 2008.
- [Wal99] C. D. Walter. Montgomery exponentiation needs no final subtractions. *Electronic Letters*, 35(21):1831–1832, October 1999.
- [Wol03] J. Wolkerstorfer. Dual-field arithmetic unit for $GF(p)$ and $GF(2^m)$. In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES'02) LNCS*, volume 2523, pages 95–114, January 2003.
- [Wol13] Wolfram Research. Wolfram Research, Mathematica Student Edition, 2013. URL: <http://www.wolfram.com/mathematica/>.
- [XB01] S. Xu and L. Batina. Efficient implementation of elliptic curve cryptosystems on an ARM7 with hardware accelerator. In *Proc. Information Security (ISC'01)*, pages 266–279, October 2001.
- [Xil05] Xilinx Inc. Xilinx Data Sheets, 2005. URL: <http://www.xilinx.com/support/documentation/index.htm>.
- [Xil12a] Xilinx Inc. Xilinx Data Sheets, 2012. URL: <http://www.xilinx.com/support/documentation/index.htm>.
- [Xil12b] Xilinx Inc. Xilinx Downloads, 2012. URL: <http://www.xilinx.com/support/download/index.htm>.
- [YJ00] S.M. Yen and M. Joye. Checking Before Output May Not be Enough against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, Sept. 2000.
- [YKLM03] S.M. Yen, S. Kim, S. Lim, and S.J. Moon. RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis. *IEEE Transactions on Computers*, 52(4):461–472, Apr. 2003.
- [YM91] H. M. Yassine and W.R. Moore. Improved mixed-radix conversion for residue number system architectures. *Circuits, Devices and Systems, IEE Proceedings G*, 138(1):120–124, Feb. 1991.

- [ZWBC02] T. Zhou, X. Wu, G. Bai, and H. Chen. New algorithm and fast VLSI implementation for modular inversion in Galois field $GF(p)$. In *Proc. IEEE Int. Conf. Commun. Circuits Syst. West Sino Expo.*, volume 2, pages 1491–1495, July 2002.



Curriculum Vitae

Dimitrios Schinianakis

born January 21st, 1983 in Athens, Greece

2005 – 2013 **University of Patras, Greece**

Doctorate Student

Department of Electrical & Computer Engineering

2000 – 2005 **University of Patras, Greece**

Diploma in Electrical and Computer Engineering

Specialization: Electronics & Computer Science

1997 – 2000 **2nd Public High School, Peristeri, Athens, Greece**

Publications

Journals

- [J1] H. E. Michail, G. Selimis, M. Galanis, D. Schinianakis, and C. E. Goutis, “Novel Hardware Implementation of the Cipher Message Authentication Code”, *Journal of Computer Systems, Networks, and Communications*, vol. 2008, Article ID 923079.
- [J2] Schinianakis, D.M., Fournaris A.P, Michail H., Kakarountas A.P and Stouraitis T., “An RNS Implementation of an F_p Elliptic Curve Point Multiplier”, *IEEE Transactions in Circuits and Systems I, Regular Papers*, Vol. 56(6), 2009, pp.1202-1213

- [J3] H. E. Michail, D. Schinianakis, and C. E. Goutis, "Cipher Block Based Authentication Module: a Hardware Design Perspective", *Journal of Circuits, Systems, and Computers*, Vol. 20, No. 2 (2011) 163-184
- [J4] Mohammad Esmaeildoust, Dimitrios Schinianakis, Hamid Javashi, Thanos Stouraitis, and Keivan Navi, "Efficient RNS Implementation of Elliptic Curve Point Multiplication Over $GF(p)$ ", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 21(8), 2013, pp.1545-1549
- [J5] Schinianakis, D.M. and Stouraitis T., "Multifunction Residue Architectures for Cryptography, *accepted for publication in IEEE Transactions in Circuits and Systems I, Regular Papers*

Refereed Conferences

- [C1] Schinianakis D.M., Fournaris A.P., Kakarountas A.P. and Stouraitis T., "An RNS architecture of an F_p elliptic curve point multiplier", *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS 2006)*, pp. 3369-3373.
- [C2] Aisopos F, Aisopos K., Schinianakis D., Michail H. and Kakarountas A.P., "A novel high-throughput implementation of a partially unrolled SHA-512", *Proc. of IEEE Mediterranean Electrotechnical Conference (MELECON 2006)*, pp. 61-65.
- [C3] Schinianakis D.M., Kakarountas A.P. and Stouraitis T., "A new approach to elliptic curve cryptography: an RNS architecture", *Proc. of IEEE Mediterranean Electrotechnical Conference (MELECON 2006)*, pp. 1241-1245.
- [C4] Michail H.E., Thanasoulis V.N., Schinianakis D.M., Panagiotakopoulos G.A., and Goutis C.E., "Application of novel technique in RIPEMD-160 aiming at high-throughput", *Proc. of IEEE International Symposium on Industrial Electronics, (ISIE 2008)*, pp. 1937-1940.
- [C5] Skavantzios A., Abdallah M., Stouraitis T., and Schinianakis D. "Design of a balanced 8-modulus RNS", *Proc. of IEEE International Conference on Electronic Circuits and Systems (ICECS 2009)*, pp. 61-64.
- [C6] Schinianakis D, Kakarountas A., and Stouraitis T., "Elliptic Curve Point Multiplication in $GF(2^n)$ using Polynomial Residue Arithmetic", *Proc. of IEEE International Conference on Electronic Circuits and Systems (ICECS 2009)*, pp. 980-983.
- [C7] Schinianakis D, and Stouraitis T., "A RNS Montgomery Multiplication Architecture", *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS 2011)*, pp. 1167-1170.
- [C8] Schinianakis D.M., Skavantzios A., Stouraitis T., " $GF(2^n)$ Montgomery Multiplication using Polynomial Residue Arithmetic", *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS 2012)*, pp. 3033 - 3036.

- [C9] Schinianakis D, and Stouraitis T., “Hardware-fault attack handling in RNS-based Montgomery multiplier”, *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS 2013)*, pp. 3042-3045
- [C10] Schinianakis D, and Stouraitis T., “An RNS modular multiplication algorithm”, *Proc. of IEEE International Conference on Electronic Circuits and Systems (ICECS 2013)*
- [C11] Schinianakis D, and Stouraitis T., “An RNS Barrett modular multiplication architecture”, *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS 2014)*