UNIVERSITÉ DE MONTPELLIER

Habilitation à Diriger les Recherches

Discipline : Informatique
École doctorale : Information Structure Systèmes (I2S)

Multiplication in Finite Fields and Elliptic Curves

par

Christophe Negre

Jury

Jean-Claude Bajard	Pr.	UPMC	Examinateur
Laurent-Stéphane Didier	Pr.	Univ. Toulon	Examinateur
Jean-Guillaume Dumas	Pr.	Univ. Grenoble	Rapporteur
Sylvain Duquesne	Pr.	Univ. Rennes 1	Rapporteur
Laurent Imbert	DR	CNRS	Examinateur
Francisco Rodríguez Henríquez	Ass. Pr.	CINVESTAV-IPN	Rapporteur

Contents

1 Curriculum Vitae **3**
- 1.1 Academic Degree . . . 3
- 1.2 Positions . . . 3
- 1.3 Research interests . . . 4
- 1.4 Supervisions of Graduate Students . . . 4
- 1.5 Participation to jury of PhD Thesis Defense . . . 4
- 1.6 Grant and Supports . . . 4

2 Introduction and research summary **5**

3 Binary field multiplier **9**
- 3.1 Review of field representation and multiplication . . . 9
  - 3.1.1 Quadratic multipliers . . . 9
- 3.2 Subquadratic multipliers based on TMVP . . . 13
  - 3.2.1 Parallel multiplier modulo a nearly all one polynomial . . . 15
  - 3.2.2 Block recombination of ONB-II Fan-Hasan multiplier . . . 17
- 3.3 Parallel multiplier based on Karatsuba formula . . . 20
  - 3.3.1 Bernstein's optimization of two recursions of Karatsuba formula . . . 21
  - 3.3.2 Generalization of Bernstein's reconstruction . . . 24
- 3.4 Conclusion . . . 26

4 Implementation of the GHASH function of the Galois counter mode **28**
- 4.1 Improvement of parallel implementation of GHASH . . . 29
- 4.2 Computing GHASH using a characteristic polynomial . . . 31
- 4.3 Conclusion . . . 34

5 Multiplication in $\mathbb{F}_p$ and $\mathbb{F}_{p^k}$ **35**
- 5.1 Modular Number System . . . 37
  - 5.1.1 Montgomery multiplication in AMNS . . . 39
  - 5.1.2 AMNS multiplication with Lagrange approach . . . 40
  - 5.1.3 Multiplication in $\mathbb{F}_{p^k}$ with improved DFT . . . 41
- 5.2 Trade-off approach in leak resistant arithmetic in residue number system . . . 43
- 5.3 Conclusion . . . 46

6 Parallel scalar multiplication over $E(\mathbb{F}_{2^n})$ and $E(\mathbb{F}_{3^n})$ **47**
6.1 Parallel Montgomery-ladder over binary elliptic curves 48
6.2 Parallel Montgomery-ladder over $E(\mathbb{F}_{3^n})$ 50
6.2.1 Thirding formula 51
6.2.2 Parallel scalar multiplication in $E(\mathbb{F}_{3^n})$ and implementation results . . . 52
6.3 Conclusion 54

7 Conclusion and future works **55**
7.1 Conclusion 55
7.2 Future works 56
7.2.1 Toward further recombined algorithms 56
7.2.2 Randomization 57

8 Bilbiography **59**
8.1 Personnal publications 59
8.2 Other references 61

Chapter 1

Curriculum Vitae

Christophe Negre,
Équipe DALI/LIRMM,
Université de Perpignan, 52, avenue Paul Alduy.
66860 Perpignan Cedex, France.
christophe.negre@univ-perp.fr
+33 (0)4 68 66 21 35
http://perso.univ-perp.fr/christophe.negre/

1.1 Academic Degree

- PhD in Computer Science, Université Montpellier 2, France, September 2004. *Opérateur arithmétique pour la cryptographie basée sur les courbes elliptiques.* Advisor J.-C. Bajard, Co-Advisor P. Elbaz Vincent.
- DEA de Mathématiques, Université Montpellier 2, France, 2001.
- Maîtrise de Mathématiques, Université Montpellier 2, France, 2000.
- Licence de Mathématiques at Université Montpellier 2, France, 1999.
- DEUG MIAS, Université Montpellier 2, France, 1998.

1.2 Positions

- 2001-2004. Teaching assistant, Department of Mathematics, Université Montpellier 2, Montpellier, France.
- 2004-2006. Temporary associate Professor, Department of Mathematics and Computer Science, Université de Perpignan, Perpignan, France.
- 2006-2015. Associate Professor, Department of Mathematics and Computer Science, Université de Perpignan, Perpignan, France. (Two years leave 2010-2012).
- 2010-2012. Associate Researcher at the University of Waterloo, Waterloo, Ontario, Canada.

1.3 Research interests

- Public key cryptography.
- Secure implementation of cryptographic protocol.
- Finite field arithmetic.
- Elliptic curve and hyperelliptic curve arithmetic.

1.4 Supervisions of Graduate Students

Master of Science students:

- 2012. J.-M. Robert. Implementation and evaluation of some optimization techniques for the arithmetic of elliptic curves.
- 2007. Z. Ma (M. Sc.). Design and implementation of a database for the *pole DERBI*.
- 2006. M. Ba (M. Sc.). Implementation of the Double Polynomial System approach for binary field arithmetic.

PhD student:

- J.-M. Robert (M. Sc. and PhD) jointly supervized with B. Goossens. Algorithms for Robust and Efficient Implementation of Cryptographic Protocols Based on Elliptic Curves. Below are related publications:
  - Parallel Approaches for Efficient Scalar Multiplication over Elliptic Curve. C. Negre and J.-M. Robert, SECRYPT 2015, 202-209.
  - Efficient Modular Exponentiation Based on Multiple Multiplications by a Common Operand. C. Negre, T. Plantard and J.-M. Robert, Arith22, 2015.
  - New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic. C. Negre and J.-M. Robert. IEEE Trans. Computers 64(10): 2875-2890 (2015).
  - Impact of Optimized Field Operations AB, AC and $AB+CD$ in Scalar Multiplication over Binary Elliptic Curve. C. Negre and J.-M. Robert. AFRICACRYPT 2013: 279-296.

1.5 Participation to jury of PhD Thesis Defense

- External examiner of the PhD thesis of Mark Hamilton. *Cryptographic Coprocessors for Embedded Systems*. University of Cork, Ireland, 2014.

1.6 Grant and Supports

- 2011-2015, Irisa-DALI/LIRMM, ANR Pavois (ANR 12 BS02 002 01), 348 K€, http://pavois.irisa.fr/. Protections Arithmétiques Vis à vis des attaques physiques pour la cryptOgraphIe basée sur les courbeS elliptiques.

Chapter 2

Introduction and research summary

Confidentiality of communication is generally achieved by ciphering plaintexts with a secret key. Classic cryptographic schemes consist either in a permutation of the plaintext letters, or in a substitution of plain letters by cipher letters or in a modular addition of a key as it is done in the Vigenère cipher [34]. Modern block ciphers combine these three operations through a substitution permutation network repeated a number of rounds to encrypt a plaintext block. This is for example the case in the two block-ciphers DES and AES used as standards [35] during the past three decades. Such secret-key approach necessitates that the two persons communicating confidentially have to agree beforehand on a secret key.

After the second world war, due to a growing flow of encrypted data, the need of fast and secure remote key exchange became crucial. In 1976 Diffie and Hellman proposed a key-exchange protocol based on the intractability of the discrete logarithm problem in a cyclic multiplicative group. Moreover Diffie and Hellman gave also a generic construction of public key encryption based on trapdoor function [36]. This idea was realized by Rivest, Adleman and Shamir in [37] with the design of the RSA public key encryption scheme. The Diffie-Hellman key exchange and RSA encryption were the two game-changing algorithms in modern cryptography. Furthermore, after more than 30 years of cryptanalysis, RSA and Diffie-Hellman cryptosystems still remain unbroken and intensively used in practice.

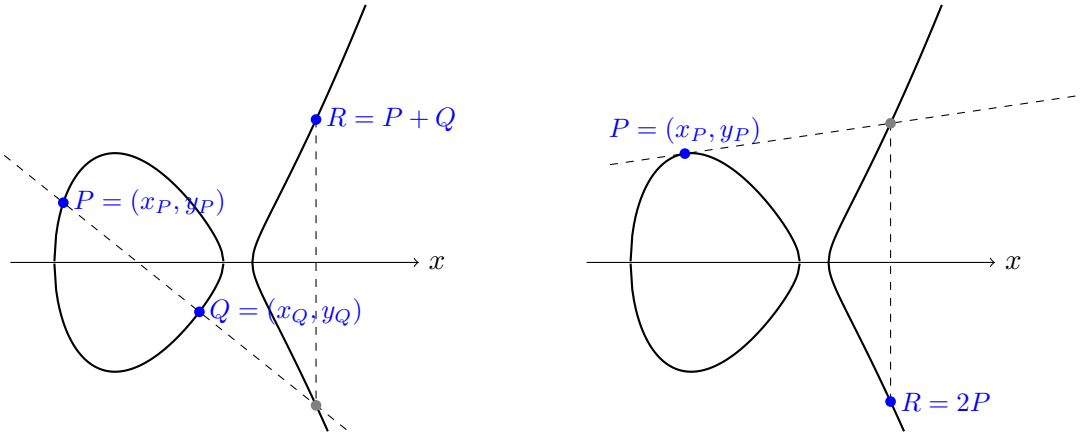
In the mid 1980's Miller [38] and Koblitz [39] independently proposed the use of the group of elliptic curve for implementing protocols based on discrete logarithm. This group is the set of points on a degree 3 smooth curve of the affine plane. The group law is given by the chord and tangent rules (cf. Figure 2.1): $P+Q$ is the symmetric of the point at the intersection of the line (PQ) and the elliptic curve, $2P$ is obtained as the symmetric of the point at the intersection of the tangent at P and the curve.

The discrete logarithm problem over an elliptic curve $(E(\mathbb{F}_q),+)$ defined over a finite field $\mathbb{F}_q$ is more difficult than in the group $(\mathbb{F}_q^\star,\times)$ initially proposed by Diffie and Hellman. Indeed the best attacks on $E(\mathbb{F}_q)$ are the generic attacks: the pollard ρ and kangaroo approaches [40] require $\sqrt{ord(P)}$ group operations while the discrete logarithm on $\mathbb{F}_q^\star$ can be solved in subexponential for large characteristic and quasi-polynomial time for small characteristic [41, 42].

The recent revelation of Edward Snowden showed that the Diffie-Hellman key exchange is one of the recommended approaches for key exchange. Indeed, the generated data are ephemeral. On the other hands the keys exchanged with the protocol based on RSA can all be recovered if the RSA secret key is leaked. This is not the case for the Diffie-Hellman scheme.

The core operations in cryptographic protocols are either an exponentiation in a finite field $\mathbb{F}_q$ with $q \in [2^{2024}, 2^{4048}]$ or an exponentiation modulo an integer N of size $[2^{2048}, 2^{4096}]$ or a

Figure 2.1: Chord and tangent rules



(a) Chord rule

(b) Tangent rule

scalar multiplication $k \cdot P$ in an elliptic curves. Such exponentiation (resp. scalar multiplication) are implemented as a sequence of multiplications in $\mathbb{F}_q$ or $\mathbb{Z}/N\mathbb{Z}$ (resp. additions in $E(\mathbb{F}_q)$).

Since the mid 90s new threats appeared on devices performing sensitive cryptographic computations. Indeed, Kocher *et al.* [43] showed that secret data can be recovered from the timings of several executions or by monitoring the power consumption [44] or electromagnetic emanation [45]. For example, the simple power analysis (SPA) uses the fact that the power trace of a multiplication and a squaring (resp. an addition and a doubling) are different. It is thus possible to recover from a single trace the exact sequence of operations performed by an embedded device. When this sequence of operations is correlated to the secret key, this key is leaked to the attacker. Consequently the cryptographic protocols must be implemented with an algorithm resistant to such attacks.

Summary of results. This manuscript presents our research results elaborated since 2004 in the DALI Team at the University of Perpignan in France and also during a stay of two years 2010-2012 at the ECE department of the University of Waterloo, ON, Canada.

We focused on improving multiplication in finite fields since it is the most used and the most costly operations in the considered cryptosystems. Our results first concerned the design of efficient parallel and sequential multipliers of binary field $\mathbb{F}_{2^n}$. We also tried to improve algorithms for multiplication in field or ring $\mathbb{Z}/N\mathbb{Z}$ and in prime field extension crafted for cryptographic protocols.

Our recent research effort was to include in these efficient multiplication algorithms some counter-measures preventing side channel attacks. Specifically, during the past three years, we have been part of the ANR project Pavois. Our task was to provide new approaches for cryptographic implementation which are resistant against side channel analyses, while remaining efficient.

We also provided new formulas for elliptic curve operation in $E(\mathbb{F}_{3^n})$ and studied parallel approaches for scalar multiplications in $E(\mathbb{F}_{2^n})$ and $E(\mathbb{F}_{3^n})$.

Our main results are:

- *Subquadratic binary field multipliers based on Toeplitz matrix vector product.* Fan and Hasan in [46] showed that a multiplication in a binary field can be expressed as a Toeplitz matrix vector product (TMVP). They could design a subquadratic space complexity mul-

tiplier using the subquadratic formula for TMVP of Winograd [47]. This construction is advantageous since it results in a multiplier with a reduced space and delay compared to subquadratic approach based on polynomial multiplication. We extended the approach of Fan and Hasan to the two following cases: polynomial multiplication modulo an irreducible nearly all one polynomial [1] and multiplication in Dickson polynomial basis [2]. We also introduced a block recombination approach in [3] and used it to improve TMVP approach for multiplication in type II optimal normal basis [4].

Part of these results are reviewed in Chapter 3, Section 3.2.

- *Subquadratic binary field multiplier based on polynomial multiplication.* In [48] Bernstein initiated an optimization of two recursions of Karatsuba formula for polynomial multiplication in $\mathbb{F}_2[x]$. In [5], we extended the idea of Bernstein to an arbitrary number of recursions. This lead to an improved parallel multiplier for $\mathbb{F}_{2^n}$.

  We present this approach in Chapter 3, Section 3.3.

- *Hardware architecture for GHASH computation.* Galois counter mode [49] is a ciphering mode for encryption and authentication. It is based on counter-mode for ciphering and on GHASH function for the generation of the authentication tag. The GHASH function consists in an evaluation of a polynomial in $\mathbb{F}_{2^n}[T]$ in a hash-key element $H \in \mathbb{F}_{2^n}$. We made several proposals for hardware architecture for GHASH computation. The first one uses a block recombination strategy [3] to reduce the space requirement of parallel implementation of the GHASH function. The second proposal uses the characteristic polynomial of H in order to reduce the critical path delay of one iteration in a GHASH computation.

  These results are reviewed Chapter 4.

- *Multiplication in prime field and prime field extension.* In [50] Bajard *et al.* introduced a novel system of representation of $\mathbb{F}_p$ called AMNS which models a multiplication in $\mathbb{F}_p$ as polynomial multiplication and lattice reduction. In [16] we introduced a Montgomery-like multiplication in the AMNS system and we also investigated an approach based on multipoint evaluation by using the polynomial form of a multiplication in AMNS. In another work [17] we took advantage of the AMNS to represent prime field element as polynomial $U = \sum_{i=0,\ldots,n-1} u_i\gamma^i$ where γ is a root of unity in $\mathbb{F}_p$. This lead us to a simplified multiplication by γ^i in $\mathbb{F}_p$ and then to an improved DFT approach for multiplication in $\mathbb{F}_{p^k}$.

  We present these results in Chapter 5, Section 5.1.

- *Regular exponentiation.* In a recent work [18] we proposed a modified word-level form of two modular Montgomery multiplications [51, 52] by a common operand AB, AC. These modified Montgomery multiplications share the common computations in AB and AC which reduces the word-level complexity. We used this approach to improve regular exponentiation algorithms like the Montgomery-ladder [53] or the left-to-right and right-to-left 2^w-ary exponentiation [54].

- *Randomized modular multiplication.* In [19], we considered the Montgomery multiplication in Residue Number System [55, 56]. We proposed in [19] a trade off approach of the leak resistant arithmetic [57] by randomly exchanging only one or a few moduli of the RNS bases. We could thus reduce the cost of this randomized modular multiplication.

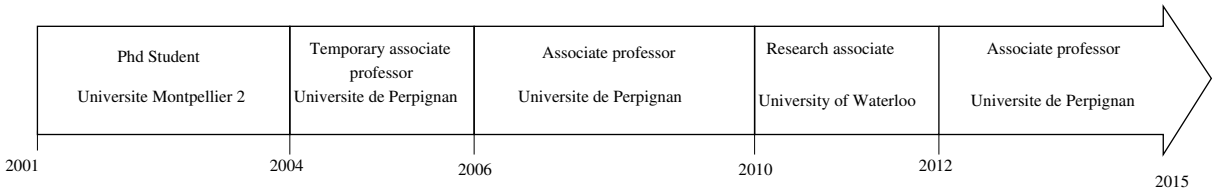
  This approach is reviewed in Chapter 5, Section 5.2.

- *Scalar multiplication over elliptic curves.* This concerns elliptic curves defined over fields $\mathbb{F}_{2^n}$ and $\mathbb{F}_{3^n}$. First, in [20, 6], we proposed some new formulas for point tripling and thirding over $E(\mathbb{F}_{3^m})$. In [6] we proposed new parallel approach for scalar multiplication: a parallelized version of the Montgomery-ladder in $E(\mathbb{F}_{2^n})$, and a parallel (third,triple)-and-add approach for scalar multiplication in $E(\mathbb{F}_{3^n})$.

  We review these results in Chapter 6.

Overview of Career. This HDR thesis present works done since 2004, which is the year the defense of our Phd thesis. These works where conducted while working at the Univeristé de Perpignan and at the University of Waterloo as detailed below. An overview is given in Fig. 2.2.

Figure 2.2: Career since 2001



- 2001-2004. PhD Student and teaching assistant at the Université Montpellier 2 in France. The subject of the thesis was the design of arithmetic operators for cryptographic protocols based on elliptic curves. This work was supervized by J.-C. Bajard and P. Elbaz-Vincent.
- 2004-2006. Temporary associate Professor in computer science at the Université de Perpignan in France, in the team DALI.
- 2006-2010. Associate Professor in the team DALI of the Université de Perpignan.
- 2010-2012. Researcher associate at the University of Waterloo in Canada (two year leave from the position at the Université de Perpignan). During this time we worked with the team of Anwar Hasan which is Professor in the electrical engineering department of the University of Waterloo.
- Associate Professor in the team DALI of the Université de Perpignan and the LIRMM (CNRS and Université de Montpellier). During this period the work done was part of the ANR project called Pavois concerning the protection of embedded cryptographic processors from side channel attacks. With B. Goossens we jointly supervized the preparation of the Phd thesis of Jean-Marc Robert which was founded by this ANR project.

Chapter 3

Binary field multiplier

In this chapter we focus on parallel multiplier for extended binary field. We first review quadratic approaches based on polynomial multiplication and matrix vector formulation of a multiplication in $\mathbb{F}_{2^n}$. We then recall the approach of Fan and Hasan [46] which sets a multiplication in $\mathbb{F}_{2^n}$ as a Toeplitz matrix vector product (TMVP) and uses the subquadratic approach of [47] to design a subquadratic space complexity parallel multiplier.

We then present two works which extend the approach of Fan and Hasan: the first one sets a multiplication modulo a nearly all one polynomial as a TMVP and the second one improves the normal basis multiplier based on TMVP of [58]. We also review a multiplier based on Karatsuba formula for polynomial multiplication: in this work we optimize the space and time complexity by reorganizing the computations in the recursive reconstruction of the Karatsuba approach.

3.1 Review of field representation and multiplication

The most usual approach to construct a binary field $\mathbb{F}_{2^n}$ consists in choosing a sparse irreducible polynomial $f(x)$ of degree n and set $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/(f(x))$. The elements of $\mathbb{F}_{2^n}$ are the polynomials of degree less than n and basic operations like addition and multiplication in $\mathbb{F}_{2^n}$ consists in multiplication and addition modulo $f(x)$:

$$W(x) = U(x) \times V(x) \mod f(x) \text{ and } W(x) = U(x) + V(x) \mod f(x).$$

The addition is a simple bitwise XOR of the coefficients of U and V while the multiplication consists in a polynomial multiplication followed by a reduction modulo $f(x)$. The simplicity of this arithmetic makes binary fields really attractive for hardware implementation since basic arithmetic operations in $\mathbb{F}_{2^n}$ are simpler than the ones in prime field $\mathbb{F}_p$ which involves costly and tedious carry propagations.

3.1.1 Quadratic multipliers

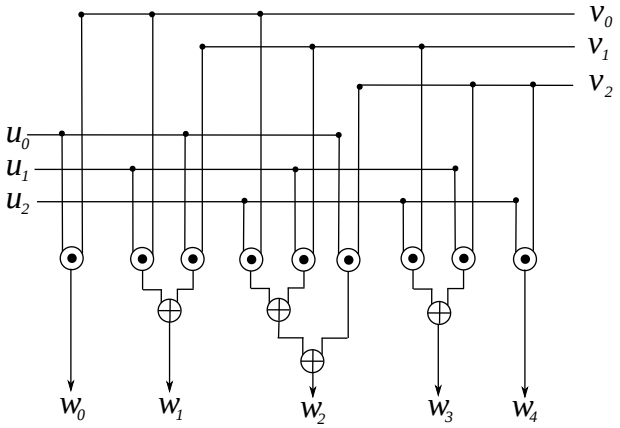
The basic approach to perform a polynomial multiplication is the schoolbook method which expands the product relatively to one operand, e.g. $U(x)$,

$$W(x) = U(x) \times V(x) = \sum_{i=0}^{n-1} u_i V \times x^i. \tag{3.1}$$

Such operation can either be done sequentially by accumulating $u_i V x^i$ in W stored in an $2n$-bit register and the result is output after n clock cycles. It can also be done in parallel fashion

with no reuse of circuit: the basic approach to perform this computation in parallel consists in n^2 AND gates computing all the products $u_i v_j$ for $i, j = 0, \ldots, n-1$ and then adding the n polynomials $u_i V \times x^i$ through a binary tree of adders requiring $(n-1)^2$ XORs. The delay of the computation is $\lceil \log_2(n) \rceil D_X + D_A$ where D_X and D_A denote the delay of an XOR and an AND gate, respectively. An example of such circuit which computes the multiplication of two degree 3 polynomials is shown in Figure 3.1.

Figure 3.1: Parallel circuit for degree 2 polynomial multiplication



The product $W(x) = U(x) \times V(x)$ have to be reduced modulo the irreducible polynomial $f(x)$. This polynomial $f(x)$ is generally chosen with a trinomial or a pentanomial form. Such irreducible polynomials exist for every cryptographic size, i.e., with $n \in [160, 512]$. The reduction modulo a trinomial or a pentanomial is quite simple: this can be performed with $O(n)$ XOR gates and a delay of $O(1)D_X$.

Another approach for the computation of binary field multiplication consists of computing simultaneously the multiplication and the reduction. This was used by Mastrovito [59] which rewrote the operation $U(x) \times V(x) \mod f(x)$ as a matrix vector product. Here, we review a more general setting of this approach where we represent the field elements through a general basis. Indeed, a finite field $\mathbb{F}_{2^n}$ has an underlying structure of vector space over $\mathbb{F}_2$. If we fix an $\mathbb{F}_2$-basis $\mathcal{B} = (e_0, \ldots, e_{n-1})$ for the representation of the field elements, the multiplication of $U = \sum_{i=0}^{n-1} u_i e_i$ and $V = \sum_{i=0}^{n-1} v_i e_i$ can be performed as follows

$$\begin{aligned} UV &= U \left(\sum_{i=0}^{n-1} v_i e_i \right) \\ &= \sum_{i=0,\ldots,n-1} U^{(i)} v_i, \end{aligned}$$

where $U^{(i)} = U e_i$. We can then rewrite the product as a matrix-vector product $W = M_U \cdot V$ where M_U is the matrix formed by the n columns $U^{(i)}$

$$M_U = \left[U^{(0)}, U^{(1)}, \cdots, U^{(n-1)} \right].$$

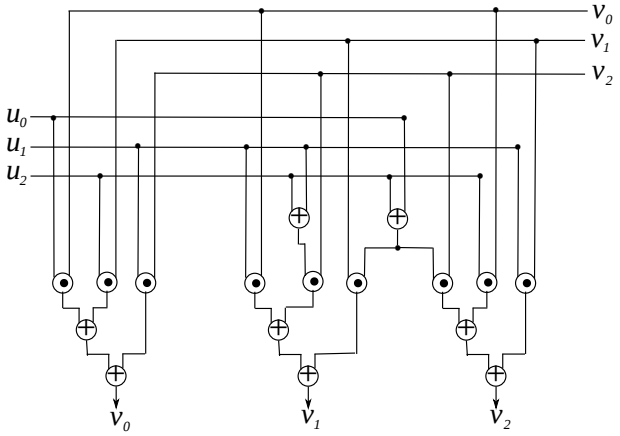
The computation of a product based on this approach consists in first computing the entries of M_U and then performing a matrix-vector product $M_U \cdot V$. The computation of the entries of the matrix M_U is related to the basis $\mathcal{B}$. The bases yielding the most efficient multipliers are the following:

- *Polynomial bases.* These bases are $\mathcal{B} = \{1, x, \ldots, x^{n-1}\}$ where, for efficiency reason, x is the root of an irreducible trinomial or pentanomial. This case corresponds to Mastrovito's multiplier [59]. For example, for $f(x) = x^2 + x + 1$ the matrix M_U is as follows

$$M_U = \begin{bmatrix} u_0 & u_2 & u_1 \\ u_1 & u_0 + u_2 & u_2 + u_1 \\ u_2 & u_1 & u_0 + u_2 \end{bmatrix}$$

and the corresponding parallel multiplier is described in Figure 3.2

Figure 3.2: Parallel multiplier for $\mathbb{F}_{2^3}$ based on matrix-vector formulation in polynomial basis



A variant of the polynomial basis is the shifted polynomial basis (SPB)

$$\{x^{-v}, x^{-v+1}, \ldots, x^{n-v}\}$$

introduced in [60]. This basis was proposed to reduce the cost of the computation of the coefficients of M_U. In [60] the authors showed that the delay of the computation of M_U is reduced when computing modulo a trinomial $x^n + x^k + 1$ where $k > 1$ and the delay and space complexities are reduced when computing modulo a pentanomial $x^n + x^{k+1} + x^k + x^{k-1} + 1$.

For example, for a multiplication modulo $f(x) = x^2 + x + 1$, the shifted polynomial basis $\mathcal{B} = \{x^{-2}, x^{-1}, 1, x, x^2\}$ provides the following matrix-vector product:

$$M_U = \begin{bmatrix} u_0 + u_{-2} & u_{-1} & u_{-2} & u_2 & u_1 \\ u_1 + u_{-1} & u_0 + u_{-2} & u_{-1} & u_{-2} & u_2 \\ u_2 & u_1 & u_0 & u_{-1} + u_2 & u_{-2} + u_1 \\ u_{-2} & u_2 & u_1 & u_0 & u_{-1} + u_2 \\ u_{-1} & u_{-2} & u_2 & u_1 & u_0 \end{bmatrix} \cdot \begin{bmatrix} v_{-2} \\ v_{-1} \\ v_0 \\ v_1 \\ v_2 \end{bmatrix}.$$

- *Low complexity normal basis.* A normal basis has the following form

$$\mathcal{N} = \{\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}\}$$

where α is an element in $\mathbb{F}_{2^n}$. Such bases are really interesting since they induce a squaring in $\mathbb{F}_{2^n}$ which is almost free of computations: the squaring is just a cyclic shift of the coordinates.

Among all normal bases, those which provide efficient multiplication in $\mathbb{F}_{2^n}$ are such that the n^2 products $\alpha^{2^i} \times \alpha^{2^j}, i, j = 0, \ldots, n-1$ have a sparse representation in $\mathcal{N}$ [61]. Specifically, the optimal normal bases (ONB) considered in [61] are the one which have the sparsest basis products $\alpha^{2^i} \times \alpha^{2^j}$. Gao and Lenstra in [62] showed that there are two types of ONB: type I and type II.

Theorem 1 (ONB-I). *Suppose $n+1$ is a prime and 2 is a primitive element in $\mathbb{Z}/(n+1)\mathbb{Z}$. Then the n non-unit $(n+1)$-th roots of unity form a type I ONB of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$.*

Let $\alpha \in \mathbb{F}_{2^n}$ be a primitive $(n+1)$-root of unity, the ONB-I $\mathcal{N} = \{\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}\}$ can be re-ordered as $\mathcal{N}' = \{\alpha, \alpha^2, \alpha^3 \ldots, \alpha^n\}$ after reducing the exponent 2^i modulo $n+1$. In [63, 64] the authors noticed that a multiplication in the re-ordered basis $\mathcal{N}'$ can be set in the following matrix vector products:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{n-2} \\ w_{n-1} \\ w_n \end{bmatrix} = \begin{bmatrix} 0 & v_n & v_{n-1} & \ldots & v_4 & v_3 & v_2 \\ v_1 & 0 & v_n & \ldots & v_5 & v_4 & v_3 \\ v_2 & v_1 & 0 & \ldots & v_6 & v_5 & v_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ v_{n-3} & v_{n-4} & v_{n-5} & \ldots & 0 & v_n & v_{n-1} \\ v_{n-2} & v_{n-3} & v_{n-4} & \ldots & v_1 & 0 & v_n \\ v_{n-1} & v_{n-2} & v_{n-3} & \ldots & v_2 & v_1 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} v_n & v_{n-1} & \ldots & v_2 & v_1 \\ v_n & v_{n-1} & \ldots & v_2 & v_1 \\ v_n & v_{n-1} & \ldots & v_2 & v_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ v_n & v_{n-1} & \ldots & v_2 & v_1 \\ v_n & v_{n-1} & \ldots & v_2 & v_1 \\ v_n & v_{n-1} & \ldots & v_2 & v_1 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}. \tag{3.2}$$

Theorem 2 (ONB-II). *Let $2n+1$ be a prime and assume that either*

i) 2 is primitive in $\mathbb{Z}/(2n+1)\mathbb{Z}$, or

ii) $2n+1 \equiv 3 \mod 4$ and 2 generates the quadratic residues in $\mathbb{Z}/(2n+1)\mathbb{Z}$.

Then $\alpha = \beta + \beta^{-1}$ generates a type II ONB of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, where β is a primitive $(2n+1)$-th root of unity in $\mathbb{F}_{2^{2n}}$.

The authors in [65] noticed that the normal basis $\mathcal{N} = \{\alpha = \beta + \beta^{-1}, \alpha^2 = \beta^2 + \beta^{-2}, \ldots, \alpha^{2^{n-1}} = \beta^{2^{n-1}} + \beta^{-2^{n-1}}\}$ can be re-ordered by reducing modulo $2n+1$ the exponent of β in α^{2^i}, this leads to the following permuted basis

$$\mathcal{N}' = \left\{\alpha_1 = \alpha + \alpha^{-1}, \alpha_2 = \alpha^2 + \alpha^{-2}, \alpha_3 = \alpha^3 + \alpha^{-3}, \ldots, \alpha_n = \alpha^n + \alpha^{-n}\right\}.$$

The authors in [65] could set the multiplication in $\mathcal{N}'$ as a sum of two matrix-vector products:

$$W = \left(\begin{bmatrix} v_n & v_n & v_{n-1} & \ldots & v_3 & v_2 \\ v_{n-1} & v_n & v_n & \ldots & v_4 & v_3 \\ v_{n-2} & v_{n-1} & v_n & \ldots & v_5 & v_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ v_2 & v_3 & v_4 & \ldots & v_n & v_n \\ v_1 & v_2 & v_3 & \ldots & v_{n-1} & v_n \end{bmatrix} + \begin{bmatrix} v_{n-1} & v_{n-2} & v_{n-3} & \ldots & v_1 & 0 \\ v_{n-2} & v_{n-3} & v_{n-4} & \ldots & 0 & v_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ v_2 & v_1 & 0 & \ldots & v_{n-4} & v_{n-3} \\ v_1 & 0 & v_1 & \ldots & v_{n-3} & v_{n-2} \\ 0 & v_1 & v_2 & \ldots & v_{n-2} & v_{n-1} \end{bmatrix} \right) \cdot \begin{bmatrix} u_n \\ u_{n-1} \\ u_{n-2} \\ \vdots \\ u_2 \\ u_1 \end{bmatrix}. \tag{3.3}$$

- *Dual basis multiplier.* Let $\mathcal{B} = \{e_0, \ldots, e_{n-1}\}$ be a basis of $\mathbb{F}_{2^n}$ and let $\varphi\colon \mathbb{F}_{2^n} \to \mathbb{F}_2$ be a non-zero linear form over $\mathbb{F}_2$. The weakly-dual basis $\mathcal{B}^* = \{e_0^*, \ldots, e_{n-1}^*\}$ of $\mathcal{B}$ relatively to φ is the unique basis satisfying

$$\varphi(e_i e_j^*) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

The conversion from a representation in $\mathcal{B}$ of an element $U \in \mathbb{F}_{2^n}$ to its representation in the dual basis $\mathcal{B}^*$ is done as follows

$$U = \sum_{i=0}^{n-1} \varphi(Ue_i) e_i^*,$$

which can be computed efficiently if the n terms $\varphi(Ue_i), i = 0, \ldots, n-1$ involve only a small number of additions.

For the special case of a polynomial basis $\mathcal{B} = \{1, x, \ldots, x^{n-1}\}$, the multiplication of two elements U and V can be expressed as matrix-vector product:

$$\begin{bmatrix} \varphi(U) & \varphi(Ux) & \cdots & \varphi(Ux^{n-1}) \\ \varphi(Ux) & \varphi(Ux^2) & \cdots & \varphi(Ux^n) \\ \vdots & & & \vdots \\ \varphi(Ux^{n-1}) & \varphi(Ux^n) & \cdots & \varphi(Ux^{2n-2}) \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} \varphi(W) \\ \varphi(Wx) \\ \vdots \\ \varphi(Wx^{n-1}) \end{bmatrix} \tag{3.4}$$

The computation of the matrix is efficient if the computation of $\varphi(Ux^i)$ is efficient for $i = 0, \ldots, 2n-2$. In order to have a multiplication with U, V and W given in the same basis, the above scheme (3.4) also requires efficient conversion from $\mathcal{B}^\star$ to $\mathcal{B}$. Several strategies were proposed in the literature [66, 67] to reach this goal.

3.2 Subquadratic multipliers based on TMVP

During the past decade the design of subquadratic space complexity parallel multiplier have drawn the attention of researchers. These multipliers reduce significantly the size of the circuit, specifically for cryptographic size. In counter parts, they have a critical path which is twice longer than the critical path of a quadratic multiplier, but it is still logarithmic.

Such multipliers are based on recursive subquadratic methods for polynomial multiplication and matrix vector product. We first review the methods based on Toeplitz-matrix vector product and present a few contributions on this matter. We will discuss subquadratic multiplier based on Karatsuba formulas for polynomial multiplication and a related contribution in Section 3.3.

Let us first remind that a Toeplitz matrix is an $n \times n$ matrix $T = [t_{i,j}; i, j = 0, 1, \cdots, n-1]$ such that $t_{i,j} = t_{i-1,j-1}$ for $i, j > 0$. Fan and Hasan showed in [46] that the matrix M_U of the multiplication in shifted polynomial basis $\mathcal{B} = \{x^{-v}, \ldots, x^{n-v}\}$ can be set in a Toeplitz form in the following cases:

- *Multiplication modulo a trinomial $x^n + x^k + 1$.* Fan and Hasan showed in [46] that when $v = k$ exchanging the first k rows with the last $n-k$ rows of the matrix M_U leads to a Toeplitz structure. In other words the following matrix T_U is Toeplitz

$$T_U = S \cdot M_U \text{ where } S = \begin{bmatrix} 0 & I_{k,k} \\ I_{n-k,n-k} & 0 \end{bmatrix}.$$

- *Multiplication modulo a pentanomial* $x^n + x^{k+1} + x^k + x^{k-1} + 1$. Fan and Hasan showed in [46] that, when $v = k$, the matrix M_U can be set in a Toeplitz form as follows

$$T_U = S \cdot M_U \text{ where } S = \begin{bmatrix} 0 & I_{n-k,n-k} + J_{n-k,n-k} \\ I_{k,k} + J_{k,k} & 0 \end{bmatrix},$$

  where $J_{k,k}$ is the $k \times k$ matrix with the single entry at $(0, k-1)$ being 1 and the others being 0.

When 2 divide n, Fan and Hasan proposed to use the two-way split formula of Winograd [47] shown in (3.5) to compute a Toeplitz matrix vector product (TMVP). This models a Toeplitz matrix-vector product of size n by three Toeplitz matrix-vector products of size $n/2$ [47, 46] as follows

$$\begin{bmatrix} T_1 & T_0 \\ T_2 & T_1 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} P_0 + P_2 \\ P_1 + P_2 \end{bmatrix} \quad \text{where} \quad \begin{cases} P_0{=}(T_1 + T_0) \cdot V_1, \\ P_1{=}(T_2 + T_1) \cdot V_0, \\ P_2{=}T_1 \cdot (V_0 + V_1). \end{cases} \tag{3.5}$$

If $n = 2^s$ and if the above formula is performed recursively, the computation can be separated into three independent operations:

- The component vector formation (CVF) which recursively splits V into two parts V_0, V_1 and computes $V_0, V_0 + V_1, V_1$.
- The component matrix formation (CMF) which recursively splits T into three Toeplitz matrices T_0, T_1, T_2 and generates $T_1 + T_0, T_2 + T_1, T_1$.
- If $\hat{V}$ and $\hat{T}$ are the $n^{\log_2(3)}$ bits output by the CVF and the CMF (the component representation of T and V), the component multiplication (CM) performs a bitwise AND $\hat{W} = \hat{T} \cdot \hat{V}$.
- The reconstruction (R) recursively reconstructs $W = T \cdot V$ as $W = [R(\hat{W}_0) + R(\hat{W}_1)$ $R(\hat{W}_1) + R(\hat{W}_2)]$ given the splitting into three parts $\hat{W} = [\hat{W}_0, \hat{W}_1, \hat{W}_2]$.

In Subfigure 3.3(a) we provide the block decomposition of a subquadratic two-way split multiplier for TMVP. In Subfigure 3.3(b) we provide the multiplier for the product of a 4×4 Toeplitz matrix and a vector of size 4. This multiplier is decomposed into the four blocks: CMF, CVF, CM and R.

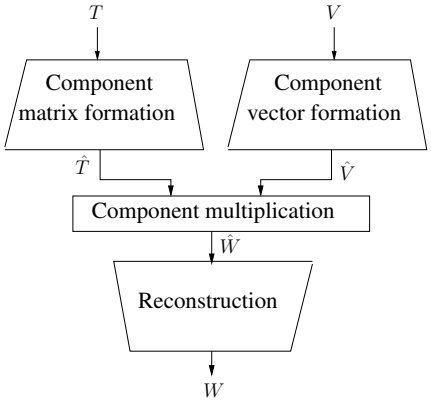
The complexity of a subquadratic space complexity TMVP multiplier is as follows for $n = 2^s$:

$$\begin{cases} \mathcal{S}_X(n) &= 5.5n^{\log_2(n)} - 6n + 0.5, \\ \mathcal{S}_A(n) &= n^{\log_2(3)}, \\ \mathcal{D}(n) &= 2\log_2(n)D_X + D_A. \end{cases} \tag{3.6}$$

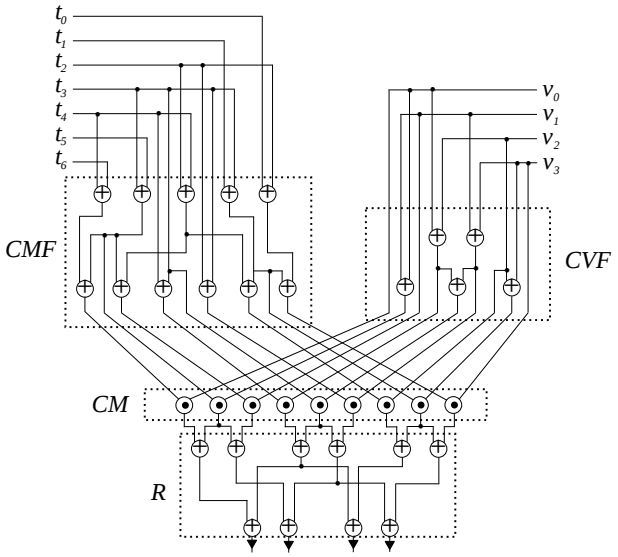
Note that $\mathcal{S}_X$ corresponds to the number of XOR gates, $\mathcal{S}_A$ to the number of AND gates and $\mathcal{D}$ represents the delay of the critical path of the circuit.

Remark 1. In a similar fashion, if $3|n$ one can use the three-way split of Winograd [47] which expresses a Toeplitz matrix-vector product of size n as six Toeplitz matrix-vector products of size $n/3$ each [46].

Figure 3.3: Block decomposition of the subquadratic space complexity TMVP multiplier



(a) Block decomposition of a TMVP multiplier



(b) TMVP multiplier for matrix and vector of size 4

3.2.1 Parallel multiplier modulo a nearly all one polynomial

We present in this subsection a work done with Anwar Hasan and Ashkan Namin in [1, 21] while Ashkan Namin was a postdoctoral fellow at the University of Waterloo in 2009-2010. Our goal was to design a subquadratic multiplier based on TMVP for multiplication modulo a nearly all one polynomials. Nearly all one polynomials (NAOPs) [68] have the following form

$$P = \sum_{i=0}^{k_2-1} x^i + \sum_{i=k_1}^{m} x^i \text{ with } k_2 < k_1.$$

In other words, for a NAOP, all the coefficients are equal to 1 unless they are in an interval $[k_2, k_1 - 1]$. If such a NAOP P is irreducible, we can define $\mathbb{F}_{2^m}$ as $\mathbb{F}_2[x]/(P)$. As noticed in [68], a multiplication in such a field can be done efficiently modulo $Q = (x+1) \times P$ since Q is a quadrinomial

$$Q = 1 + x^{k_2} + x^{k_1} + x^n \text{ with } n = m + 1.$$

In the sequel we will consider a generalization of the NAOPs, i.e., quadrinomials such that $Q = x^n + x^{k_1} + x^{k_2} + 1$ of degree $n = m + \gamma$ where γ is small relative to m and Q splits as

$$Q = R(x) \times P(x) \text{ where } \deg R = \gamma \text{ and } \deg P = m, \tag{3.7}$$

and P is irreducible. This enlarges the set of possible quadrinomials in $\mathbb{F}_{2^m}$ that can be used for a multiplication.

We now present our contribution [1, 21] on multiplication modulo $Q = 1 + x^{k_2} + x^{k_1} + x^n$ which is a quadrinomial with $0 < k_2 < k_1 < n$ in $\mathbb{F}_2[x]$. Our aim was to get a subquadratic complexity multiplier modulo such a quadrinomial. To reach this goal, we attempted to express the product of two elements modulo Q as a Toeplitz matrix vector product.

We used a double basis approach: the elements of $\mathbb{F}_2[x]/(Q)$ are represented in the basis $\mathcal{B} = \{e_0, e_1, \ldots, e_{n-1}\}$ given in (3.8) where $l_1 = n - k_1$ and $l_2 = n - k_2$ and a second basis $\mathcal{B}'$

given in (3.8). In the literature [69, 70] the basis $\mathcal{B}$ is called a triangular basis, and we will call $\mathcal{B}'$ a modified triangular basis.

$\mathcal{B}$	$\mathcal{B}'$	index
$e_i = x^i$	$e'_i = x^i$	for $i \in [0, l_1[$
$e_i = x^i + x^{i-l_1}$	$e'_i = x^i$	for $i \in [l_1, l_2[$,
$e_i = x^i + x^{i-l_1} + x^{i-l_2}$	$e'_i = e_i$	for $i \in [l_2, n[$

(3.8)

Below we give the construction of such a double basis multiplier: let $U = \sum_{i=0}^{n-1} u_i e_i$ be expressed relatively to $\mathcal{B}$ and $V = \sum_{j=0}^{n-1} v'_j e'_j$ be expressed relatively to $\mathcal{B}'$. The product $W = UV$ can be written as

$$\begin{aligned} W &= U\left(\sum_{j=0}^{n-1} v'_j e'_j\right) = \sum_{j=0}^{n-1} v'_j \left(U e'_j\right) = \sum_{j=0}^{n-1} v'_j U^{(j)} \\ &= \underbrace{\left[U^{(0)}, U^{(1)}, \cdots, U^{(n-1)}\right]}_{M_U} \cdot V = M_U \cdot V \end{aligned} \tag{3.9}$$

where $U^{(j)} = U e'_j$ are the column vectors consisting of the coefficients of $U^{(j)}$ in $\mathcal{B}$ and V is the column vector of the coefficients of V in $\mathcal{B}'$. In [1, 21] we showed that the $n \times n$ matrix M_U can be generated column by column as follows.

- **Generating the first l_2 columns of M_U.** First, note that $U^{(0)} = U$ and for $1 \leq i \leq l_2 - 1$ one can write

$$U^{(i)} = U e'_i = U x^i = U x^{i-1} x = U^{(i-1)} x \bmod Q.$$

  In [1, 21] we showed that the coordinates of $U^{(i)} = \sum_{j=0}^{n-1} u_j^{(i)} e_j$ in $\mathcal{B}$ are given in terms of the coordinates $u_j^{(i-1)}$ of $U^{(i-1)}$ as follows

$$U^{(i)} = \left(\sum_{j=0}^{n-2} u_j^{(i-1)} e_{j+1}\right) + u_{n-1}^{(i-1)} e_0 + u_{l_1-1}^{(i-1)} e_0 + u_{l_2-1}^{(i-1)} e_0.$$

  This previous expression enables us to compute $U^{(i)}$ as illustrated below:

$U^{(0)}$	$U^{(1)}$	$U^{(2)}$	$\cdots$
$\downarrow$	$\downarrow$	$\downarrow$	
u_0	$u_{n-1} + u_{l_1-1} + u_{l_2-1}$	$u_{n-2} + u_{l_1-2} + u_{l_2-2}$	$\cdots$
u_1	u_0	$u_{n-1} + u_{l_1-1} + u_{l_2-1}$	$\cdots$
u_2	u_1	u_0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	
u_{n-1}	u_{n-2}	u_{n-3}	$\cdots$

  This gives the column $U^{(i)}$ of M_U for $i < l_2$.

- **Generating the last k_2 columns of M_U.** We noticed in [1, 21] that $U^{(i)} = U(x^i + x^{i-l_1} + x^{i-l_2})$ for $i = l_2, l_2+1, \ldots, n-1$. Thus, if we factorize x out we obtain for $i = l_2, l_2+1, \ldots, n-1$

$$U^{(i)} = U(x^{i-1} + x^{i-1-l_1} + x^{i-1-l_2})x = U^{(i-1)} x \quad \Longrightarrow \quad U^{(i-1)} = U^{(i)} x^{-1}.$$

  In [1, 21], we could express $U^{(i-1)}$ in terms of $U^{(i)}$ as follows

$$U^{(i-1)} = \sum_{j=1}^{n-2} u_j^{(i)} e_{j-1} + u_0^{(i)} e_{n-1} + u_{l_2}^{(i)} e_{n-1} + u_{l_1}^{(i)} e_{n-1}.$$

We then use the fact that $U^{(n)} = U \times (x^n + x^{n-l_1} + x^{n-l_2}) = U$ to generate the columns $U^{(n-1)}, U^{(n-2)}, \ldots$ as follows:

$$\begin{array}{cccc}
\cdots & U^{(n-2)} & U^{(n-1)} & U^{(n)} = U \\
 & \downarrow & \downarrow & \downarrow \\
\cdots & u_2 & u_1 & u_0 \\
\cdots & u_3 & u_2 & u_1 \\
\cdots & u_4 & u_3 & u_2 \\
 & \vdots & \vdots & \vdots \\
\cdots & u_0 + u_{l_1} + u_{l_2} & u_{n-1} & u_{n-2} \\
\cdots & u_1 + u_{l_1+1} + u_{l_2+1} & u_0 + u_{l_1} + u_{l_2} & u_{n-1}
\end{array}$$

Finally, we could note that the matrix

$$T_U = [U^{(l_2)}, U^{(l_2+1)}, \ldots, U^{(n-1)}, U^{(0)}, U^{(1)}, \ldots, U^{(l_2-1)}] \tag{3.10}$$

is a Toeplitz matrix and that the first k_2 columns (resp. the last $l_2 = n - k_2$ columns) of T_U are the last k_2 columns (resp. the first $n - k_2$ columns) of $M_U = [U^{(0)}, U^{(1)}, \ldots, U^{(n-1)}]$. We then could use a subquadratic approach based on TMVP to compute a multiplication modulo Q.

3.2.2 Block recombination of ONB-II Fan-Hasan multiplier

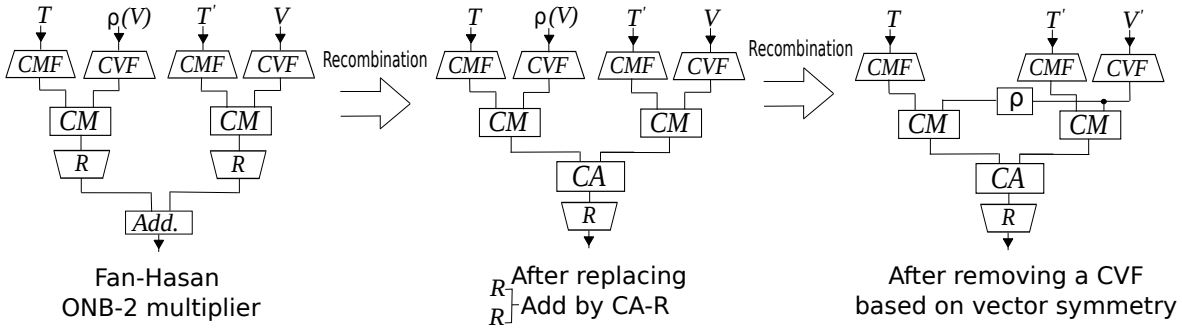
In this subsection, we present a work done with Jithra Adikari, Ayad Barsoum, Anwar Hasan and Ashkan Namin in 2010-2011. The goal was to improve the space requirement of the subquadratic space complexity multiplier of Fan and Hasan in [58] for ONB-II. We consider two elements $U = (u_1, u_2, \ldots, u_n)$ and $V = (v_1, v_2, \ldots, v_n)$ in $\mathbb{F}_{2^n}$ expressed in the permuted ONB-II $\mathcal{B}' = \{\beta^i + \beta^{-i}, i = 1, \ldots, n\}$. From (3.3) we know that the product of U and V can be formulated as a sum of a Hankel matrix-vector and a Toeplitz matrix-vector products. Writing the columns of the Hankel matrix in the reverse order, Fan and Hasan could express a multiplication in an ONB-II as the sum of two Toeplitz matrix-vector products:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{n-2} \\ w_{n-1} \\ w_n \end{bmatrix} = \begin{bmatrix} v_n & v_n & v_{n-1} & \ldots & v_3 & v_2 \\ v_{n-1} & v_n & v_n & \ldots & v_4 & v_3 \\ v_{n-2} & v_{n-1} & v_n & \ldots & v_5 & v_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ v_2 & v_3 & v_4 & \ldots & v_n & v_n \\ v_1 & v_2 & v_3 & \ldots & v_{n-1} & v_n \end{bmatrix} \cdot \begin{bmatrix} u_n \\ u_{n-1} \\ u_{n-2} \\ \vdots \\ u_2 \\ u_1 \end{bmatrix} + \begin{bmatrix} 0 & v_1 & v_2 & \ldots & v_{n-2} & v_{n-1} \\ v_1 & 0 & v_1 & \ldots & v_{n-3} & v_{n-2} \\ v_2 & v_1 & 0 & \ldots & v_{n-4} & v_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ v_{n-2} & v_{n-3} & v_{n-4} & \ldots & 0 & v_1 \\ v_{n-1} & v_{n-2} & v_{n-3} & \ldots & v_1 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}. \tag{3.11}$$

Fan and Hasan in [58] proposed to use this formulation to compute the product in an ONB-II through two parallel subquadratic space complexity TMVP multipliers combined with a final adder. This approach is shown in the left side of Figure 3.4. In this figure there is the vector V and the same vector in the reverse order $\rho(V)$. In the sequel we will denote $\rho(U)$ the operator which reverses the order of the entries of a vector U.

In [4] we evaluated the repartition of the space complexity of the different blocks of a

Figure 3.4: Block recombination of ONB-II multiplier



subquadratic space complexity TMVP multiplier. When $n = 2^s$, we found that

$$\begin{aligned} \text{CMF}: &\quad \tfrac{5}{2}n^{\log_2(3)} \text{ XOR gates,} \\ \text{CVF}: &\quad n^{\log_2(3)} - n \text{ XOR gates,} \\ \text{CM}: &\quad n^{\log_2(3)} \text{ AND gates,} \\ \text{R}: &\quad 2n^{\log_2(3)} - 2n \text{ XOR gates.} \end{aligned}$$

In [4] we proposed some optimizations of the ONB-II multiplier of Fan and Hasan. Our strategy was to recombine the computations (CMF,CVF,...) in order to replace *big* blocks by smaller ones. The recombinations proposed in [4] are the following:

- *Replacing a block R by a component addition (CA).* We first used the linearity of the reconstruction R in the subquadratic formula for TMVP, i.e., $R(\hat{W} + \hat{W}') = R(\hat{W}) + R(\hat{W}')$. In other words, we can perform the addition before the reconstruction. We save one reconstruction block and in counter part we have an addition which is more costly, i.e., $n^{\log_2(3)}$ XOR gates, since it is done in component representation. This recombination applied to the Fan-Hasan ONB-II multiplier results in the multiplier shown in the middle of Figure 3.4.

- *Removing a block CVF using a symmetry of the vectors.* We noticed the following fact: in (3.11) the two vectors are the reverse of each other. We proved in [4] that the CVF applied to $\rho(V)$ (i.e., V in the reverse order) produces the same coefficients as CVF(V) but in reverse order:
$$\text{CVF}(\rho(V)) = \rho(\text{CVF}(V))$$
This property was used in [4] to remove one of the CVF block of the Fan-Hasan ONB-II multiplier. The recombined multiplier is shown on the right side of Figure 3.4.

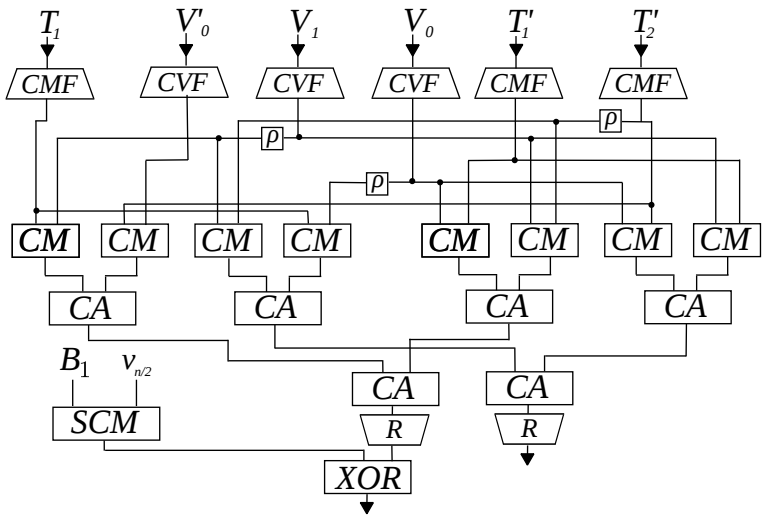
- *Removing CMF blocks using matrix symmetries.* The third recombination is based on the symmetry of the matrices. In order to extract this symmetry we decomposed each matrix of (3.11) in four sub-matrices and each vector of (3.11) into two sub-vectors. We then could rewrite (3.11) as follows:
$$W = \begin{bmatrix} T_1 \cdot \rho(U_1) + u_{n/2}V_1 + T_2' \cdot V_0' \\ (T_2')^t \cdot \rho(V_1) + T_1 \cdot \rho(V_0) \end{bmatrix} + \begin{bmatrix} T_1' \cdot V_0 + (T_2')^t \cdot V_1 \\ T_2' \cdot V_0 + T_1' \cdot V_1 \end{bmatrix}. \tag{3.12}$$
In the above equation we have an $n/2 \times n/2$ matrix T_2' which appears a multiple of time as sub-matrix of the two initial $n \times n$ matrices T, T'. Consequently we can share the related

computations of the CMF of T and T' corresponding to T'_2. Furthermore we have also T'^t_2 which appears in (3.12). We showed in [4] that the CMF of a transposed matrix T^t has the same entries as CMF(T) but in the reverse order:

$$\text{CMF}(T^t) = \rho(\text{CMF}(T)).$$

This reduces the computations of the CMF of T and T' to the CMF computation of the three matrices T_1, T'_1 and T'_2. In counter part we have an additional CVF computation since a new term V'_0 appeared in (3.12). The resulting multiplier architecture based on this recombination is shown in Figure 3.5 where SCM stands for a *multiplication by a scalar*.

Figure 3.5: Recombined ONB multiplier based on matrix symmetries



At the end we obtained in [4] a multiplier with the complexity reported in Table 3.1. We notice that compared to the multiplier of Fan and Hasan based on TMVP the total number of gates is reduced by 25% while having the same delay. We provide also the complexity of ONB-II multipliers based on polynomial multiplication: this approach was initiated by Shokrollahi, Shokrollahi and von zur Gathen in [71] and later improved by Bernstein and Lange in [72]. Our proposed multiplier have a total number of gate larger by 18% but it is at least two times faster.

Table 3.1: Space and time complexities of ONB II multipliers

Method	# AND	# XOR	Delay
Pol. with [72] and [48]	$n^{\log_2(3)}$	$5.5n^{\log_2(3)} - 7n + 2.5 + 2n\log_2(n)$	$D_A + (5\log_2(n) + 2)D_X$
Pol. with [72] and [73]	$n^{\log_2(3)}$	$6n^{\log_2(3)} - 8n + 3 + 2n\log_2(n)$	$D_A + (4\log_2(n) + 2)D_X$
Fan-Hasan [58] based on TMVP	$2n^{\log_2(3)}$	$11n^{\log_2(3)} - 12n + 1$	$D_A + (2\log_2(n) + 1)D_X$
Proposed recombined TMVP [4]	$2.66n^{\log_2(3)} + 0.5n$	$6.83n^{\log_2(3)} - 7.5n + 1.5$	$D_A + (2\log_2(n) + 1)D_X$

We present in Table 3.2 hardware implementation results (FPGA and ASIC) provided in [4]. Our implementations are for the field $\mathbb{F}_{2^{233}}$, the only binary field with an ONB-II in the NIST

recommendations for ECC applications. We implemented the multiplier based on the Bernstein-Lange approach [72], for this we used the two-way formulas given in [48] and [73] for binary polynomial multiplication. We also implemented the method based on TMVP given in [58] and the proposed block recombination of Fan and Hasan multiplier. The implementation results are given in Table 3.2.

Table 3.2: FPGA Place & Route, ASIC Synthesis Results for ONB-II Multiplier in $\mathbb{F}_{2^{233}}$

Design	FPGA				ASIC			
	Slice	LUT	Max. Freq. (MHz)	FPGA Eff.†	Area (μm^2)	Gate Count	Max. Freq. (MHz)	ASIC Eff. ‡
Pol. with [72] and [48]	6,624	20,867	130.575	6,257	151,857	140,608	396.83	2,822
Pol. with [72] and [73]	7,030	22,817	134.809	5,908	158,302	146,575	403.23	2,751
TMVP [58]	10,007	32,116	192.340	5,989	219,127	202,896	568.18	2,800
Proposed recomb. TMVP	7,332	23,418	198.067	8,415	178,083	164,892	543.47	3,296

According to Table 3.2, ONB-II multipliers based on [72]-[48] combined have the smallest area in both FPGA and ASIC implementations. The proposed recombined TMVP multiplier are better than the original approach of Fan and Hasan in terms of area and speed for FPGA implementations. In ASIC implementations, ONB-II multipliers based on TMVP ([58] and the proposed recombined variants) have the highest maximum operating frequencies. With regard to hardware efficiency, the proposed method always has the best results.

3.3 Parallel multiplier based on Karatsuba formula

In this section we consider polynomial approach for the design of subquadratic space complexity multiplier in $\mathbb{F}_{2^n}$. Specifically, we will focus on parallel multiplier derived from Karatsuba formula for polynomial multiplication. We consider two degree $n-1$ polynomials $U(x) = \sum_{i=0}^{n-1} u_i x^i$ and $V(x) = \sum_{i=0}^{n-1} v_i x^i$ in $\mathbb{F}_2[x]$ with, for the sake of simplicity, $n = 2^s$. The method of Karatsuba for polynomial multiplication consists in expressing the product $W = U \times V$ in terms of three multiplications of half-size polynomials. The detailed computations are given below:

- *Component polynomial formation (CPF).* The component polynomial formation is achieved by splitting U into two halves

$$U(X) = \underbrace{\sum_{i=0}^{n/2-1} u_i x^i}_{U_L} + x^{n/2} \underbrace{\sum_{i=0}^{n/2-1} u_{i+n/2} x^i}_{U_H}$$

  and then generating three polynomials of half the size of U: $U'_0 = U_L, U'_1 = U_L + U_H$ and $U'_2 = U_H$. The same is done for $V = V_L + V_H x^{n/2}$: we generate $V'_0 = V_L, V'_1 = V_L + V_H$ and $V'_2 = V_H$.

- *Recursive products.* We perform the pairwise products of the CPF of U and V

$$\begin{aligned} W'_0 &= U'_0 V'_0 = U_L V_L, \\ W'_1 &= U'_1 V'_1 = (U_L + U_H)(V_L + V_H), \\ W'_2 &= U'_2 V'_2 = U_H V_H. \end{aligned} \tag{3.13}$$

- *Reconstruction.* We then reconstruct $W = U \times V$ as

$$\begin{array}{rcl} W & = & W'_0(1+x^{n/2}) + W'_1 x^{n/2} + W'_2 x^{n/2}(1+x^{n/2}), \\ & = & W'_0 + (W'_0 + W'_1 + W'_2)x^{n/2} + W'_2 x^n. \end{array} \tag{3.14}$$

This reconstruction can be optimized as follows:

$$\begin{array}{ll} \textit{Step 1.} & R_0 = W'_0 + x^{n/2} W'_1, \\ \textit{Step 2.} & R_1 = R_0(1+x^{n/2}), \\ \textit{Step 3.} & C = R_1 + W'_2 x^{n/2}. \end{array} \tag{3.15}$$

In the sequel we will call the reconstruction formula (3.15) as the Bernstein's Reconstruction (BR_1) of one recursion of the Karatsuba formula.[1]

The three half-size products W'_0, W'_1 and W'_2 of (3.13) are computed by applying the same method recursively. If the recursive computations are performed in parallel we get a parallel multiplier with a subquadratic space complexity and a logarithmic delay. Specifically, the number of XOR gates ($\mathcal{S}_X(n)$), AND gates ($\mathcal{S}_A(n)$) and the delay $\mathcal{D}(n)$ of the Karatsuba approach involving the reconstruction of (3.15) are given in (3.16) for a recursion of depth $\log_2(n)$.

$$\left\{ \begin{array}{rcl} \mathcal{S}_X(n) & = & 5.5n^{\log_2(n)} - 7n + 1.5, \\ \mathcal{S}_A(n) & = & n^{\log_2(3)}, \\ \mathcal{D}(n) & = & 3\log_2(n) D_X + D_A. \end{array} \right. \tag{3.16}$$

The above complexity shows that parallel multipliers based on Karatsuba formula are not competitive with the two-way split TMVP approach: the space complexity is roughly the same but the delay is 50% larger. In [73] the authors propose an overlap-free variant of the Karatsuba formula which reduces the delay to $2\log_2(n)D_X + D_A$ but increases the space complexity up to $6n^{\log_2(n)} + O(n)$ XOR gates.

To illustrate the kind of multiplier obtained with Karatsuba formula we provide in Figure 3.6 the subquadratic space complexity multiplier for degree four polynomials U and V. This multiplier is decomposed into the three blocks CPF, CM and R.

3.3.1 Bernstein's optimization of two recursions of Karatsuba formula

In [48] Bernstein proposes an optimization on two recursions of Karatsuba formula. Here, we review this approach in a slightly different manner. We consider two degree $n-1$ polynomials $U(x) = \sum_{i=0}^{n-1} u_i x^i$ and $V(x) = \sum_{i=0}^{n-1} v_i x^i$ in $\mathbb{F}_2[x]$ with $n = 2^s$.

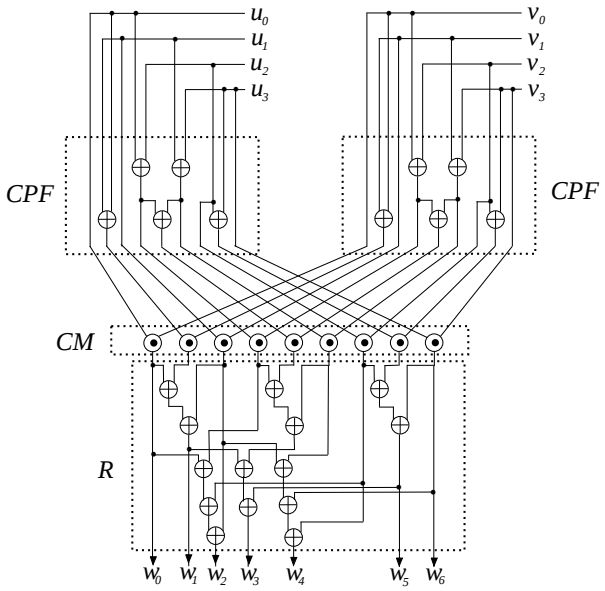
Component formation and recursive products. The component formation is recursively applied twice resulting in nine terms of size $n/4$. The polynomial U is split in four parts $U = U_{LL} + U_{LH}x^{n/4} + U_{HL}x^{2n/4} + U_{HH}x^{3n/4}$ and the nine terms of size $n/4$ generated by the two recursions of the CPF are as follows

$$\begin{array}{lll} U_0^{(2)} = U_{LL}, & U_1^{(2)} = (U_{LL} + U_{LH}), & U_2^{(2)} = U_{LH}, \\ U_3^{(2)} = (U_{LL} + U_{HL}), & U_4^{(2)} = (U_{LL} + U_{LH} + U_{HL} + U_{HH}) & , \\ U_5^{(2)} = (U_{LH} + U_{HH}), & U_6^{(2)} = U_{HL}, & U_7^{(2)} = (U_{HL} + U_{HH}), \\ U_8^{(2)} = U_{HH}. & & \end{array}$$

The same formula is applied to V which results in nine terms $V_i^{(2)}$, $i = 0, 1, \ldots, 8$ of size $n/4$. The nine recursive products are $W_i^{(2)} = U_i^{(2)} \times V_i^{(2)}, i = 0, 1 \ldots, 8$ and have degree $2n/4 - 2$.

[1] The reconstruction (3.15) was known for some times: it can be found in [48, 74, 75].

Figure 3.6: Karatsuba multiplier for degree four polynomials



Reconstruction. The first recursion in the reconstruction process produces $W_i^{(1)}, i = 0, 1, 2$ in terms of $W_i^{(2)}, i = 0, 1, \ldots, 8$. Specifically, $W_0^{(1)}$ is expressed in terms of $W_i^{(2)}, i = 0, 1, 2$, and $W_1^{(1)}$ is expressed in terms of $W_i^{(2)}, i = 3, 4, 5$, and $W_2^{(1)}$ is expressed in terms of $W_i^{(2)}, i = 6, 7, 8$, as follows:

$$\begin{array}{rcl} W_0^{(1)} & = & (1+x^{n/4})W_0^{(2)} + x^{n/4}W_1^{(2)} + x^{n/4}(1+x^{n/4})W_2^{(2)}, \\ W_1^{(1)} & = & (1+x^{n/4})W_3^{(2)} + x^{n/4}W_4^{(2)} + x^{n/4}(1+x^{n/4})W_5^{(2)}, \\ W_2^{(1)} & = & (1+x^{n/4})W_6^{(2)} + x^{n/4}W_7^{(2)} + x^{n/4}(1+x^{n/4})W_8^{(2)}. \end{array} \tag{3.17}$$

The second recursion of the reconstruction produces $C = W^{(0)}$ in terms of $W_0^{(1)}, W_1^{(1)}, W_2^{(1)}$ as

$$W = (1+x^{n/2})W_0^{(1)} + x^{n/2}W_1^{(1)} + x^{n/2}(1+x^{n/2})W_2^{(1)}. \tag{3.18}$$

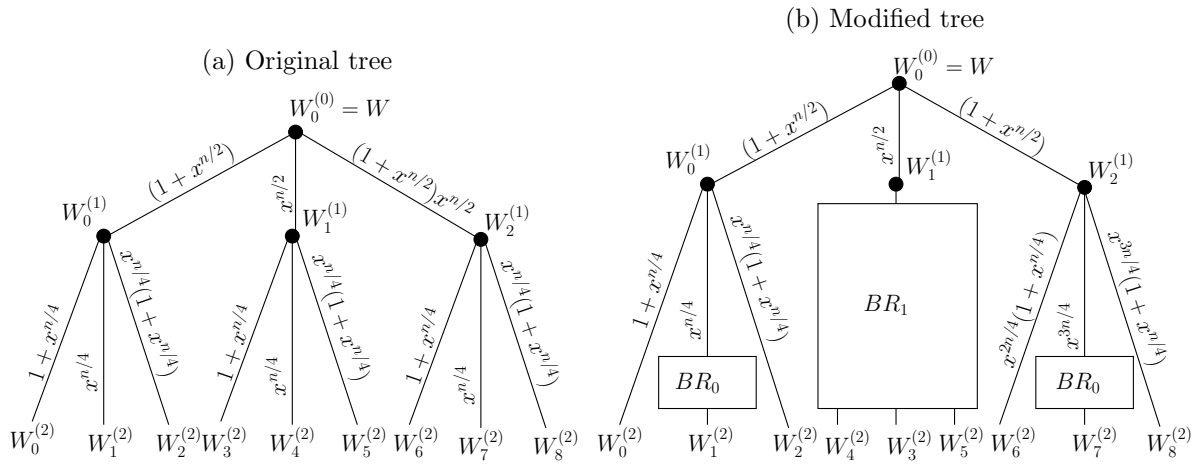
Note that the superscript (i) of $U^{(i)}, V^{(i)}$ and $W^{(i)}$ indicates the depth of the data in the recursion.

The operations of (3.17) and (3.18) can be organized in a tree structure: starting from the root $W = W_0^{(0)}$ which is linked to the three children $W_0^{(1)}, W_1^{(1)}$ and $W_2^{(1)}$. The links are labeled by the respective factors $(1+x^{n/2}), x^{n/2}$ and $x^{n/2}(1+x^{n/2})$ of $W_i^{(1)}, i = 0, 1, 2$, appearing in (3.18). This process is repeated for each $W_i^{(1)}, i = 0, 1, 2$, the resulting reconstruction tree is shown in Subfigure 3.7(a).

The optimization of Bernstein can be formulated by the following changes on the reconstruction tree:

- The sub-tree below $W_1^{(1)}$ is replaced by a block representing the reconstruction formula BR_1 given in (3.15) in terms of $W_3^{(2)}, W_4^{(2)}, W_5^{(2)}$.
- The middle terms $W_1^{(2)}$ and $W_7^{(2)}$ are also replaced by a block BR_0 representing a reconstruction of depth 0: by convention $BR_0(U) = U$ for any U.

Figure 3.7: Original and modified trees of two recursions of the Karatsuba formula



- Finally, the factor $x^{n/2}$, which appears in the label of the link between $W_0^{(0)}$ and $W_2^{(1)}$, is moved down to the three links below $W_2^{(1)}$.

The resulting modified tree is shown in Subfigure 3.7(b). This tree provides the reconstruction of Bernstein for two recursions of the Karatsuba formula. The operations labeled on the different links are performed in the following order: by starting from the bottom of the modified tree and climbing up to the root. At each level we accumulate the middle terms and multiply by the level factor. The resulting sequence of operations of Bernstein's reconstruction is given in Algorithm 1.

Algorithm 1 BR_2

Require: $W_0^{(2)}, W_1^{(2)}, \ldots, W_8^{(2)}$, such that $W_i^{(2)} = U_i^{(2)} \times V_i^{(2)}$ where $U_i^{(2)}$ and $V_i^{(2)}$ are generated by two recursions of the CPF applied to two degree $n = 4n'$ polynomials A and B.
Ensure: $W = U(x) \times V(x)$

// Initialization: accumulation of the leaf-coefficients $W_0^{(2)}, W_2^{(2)} W_6^{(2)}, W_8^{(2)}$
$W \leftarrow W_0^{(2)} + W_2^{(2)} x^{n/4} + W_6^{(2)} x^{2n/4} + W_8^{(2)} x^{3n/4}$
// Multiplication by the common factor $(1 + x^{n/4})$
$W \leftarrow W \times (1 + x^{n/4})$
// Recursive reconstruction of the middle terms $W_1^{(2)}$ and $W_1^{(2)}$ of depth 2
$Z_0 \leftarrow BR_0(W_1^{(2)}),\ Z_1 \leftarrow BR_0(W_7^{(2)})$
// Accumulation of the reconstructed middle terms of depth 2
$W \leftarrow W + x^{n/4} Z_0 + x^{3n/4} Z_1$
// Multiplication by the common factor $(1 + x^{n/2})$
$W \leftarrow W \times (1 + x^{n/2})$
// Reconstruction of the middle term of depth 1
$Z_0 \leftarrow BR_1(W_3^{(2)}, W_4^{(2)}, W_5^{(2)})$
// Accumulation of the middle term of depth 1 multiplied by its factor $x^{n/2}$
$W \leftarrow W + x^{n/2} Z_0$

3.3.2 Generalization of Bernstein's reconstruction

In [5] we proposed a generalization of the optimization of Bernstein on two recursions of Karatsuba reconstruction. Let us review this approach considering s recursions of Karatsuba formula:

Component polynomial formations and recursive products. The component polynomial formation (CPF) consists in recursively splitting into two halves and then generating three half-size polynomials: the two halves and their sum. The application of the recursive CPF of depth s to the polynomial U of size n results in 3^s polynomials $U_i^{(s)}, i = 0, 1, \ldots, 3^s - 1$ of size $n/2^s$. Similarly, if we also apply a recursive CPF of depth s to the polynomial V we obtain 3^s terms $V_i^{(s)}$, $i = 0, 1, \ldots, 3^s - 1$. Then, there are 3^s products $W_i^{(s)} = U_i^{(s)} \times V_i^{(s)}, i = 0, \ldots, 3^s - 1$, and the resulting polynomials $W_i^{(s)}$ are of degree $2n/2^s - 2$.

Reconstruction tree. The reconstruction consists in applying the formula (3.14) to each group of three consecutive $W_i^{(s)}$

$$W_i^{(s-1)} = W_{3i}^{(s)}(1 + x^{n/2^s}) + W_{3i+1}^{(s)} x^{n/2^s} + W_{3i+2}^{(s)} x^{n/2^s}(1 + x^{n/2^s}), \tag{3.19}$$

where $0 \leq i \leq 3^{s-1} - 1$. Then we repeat this process for the depths $s - 1, s - 2, \ldots, 1, 0$ until we obtain $W = W_0^{(0)}$. This recursive reconstruction can be arranged through a reconstruction tree of depth s which has the following properties:

- The root node is labeled as $W_0^{(0)}$ and corresponds to the reconstructed product $W = U \times V$.
- The intermediate nodes of depth h are labeled as $W_i^{(h)}$ where $0 \leq i < 3^h$. We measure the depth h relatively to the root $W_0^{(0)}$ of the reconstruction tree.
- Each node $W_i^{(h)}$ of depth $h < s$ has three children $W_{3i}^{(h+1)}, W_{3i+1}^{(h+1)}$ and $W_{3i+2}^{(h+1)}$ of depth $h+1$, and three links which are labeled by $(1 + x^{n/2^{h+1}}), x^{n/2^{h+1}}$ and $x^{n/2^{h+1}}(1 + x^{n/2^{h+1}})$ respectively. This corresponds to one application of the reconstruction formula (3.14). In the sequel, the link joining $W_i^{(h)}$ to $W_{3i}^{(h+1)}$ will be called the left (L) link, the link joining $W_i^{(h)}$ to $W_{3i+2}^{(h+1)}$ will be called the right (R) link and the link $W_i^{(h)}$ to $W_{3i+1}^{(h+1)}$ will be called the middle (M) link.

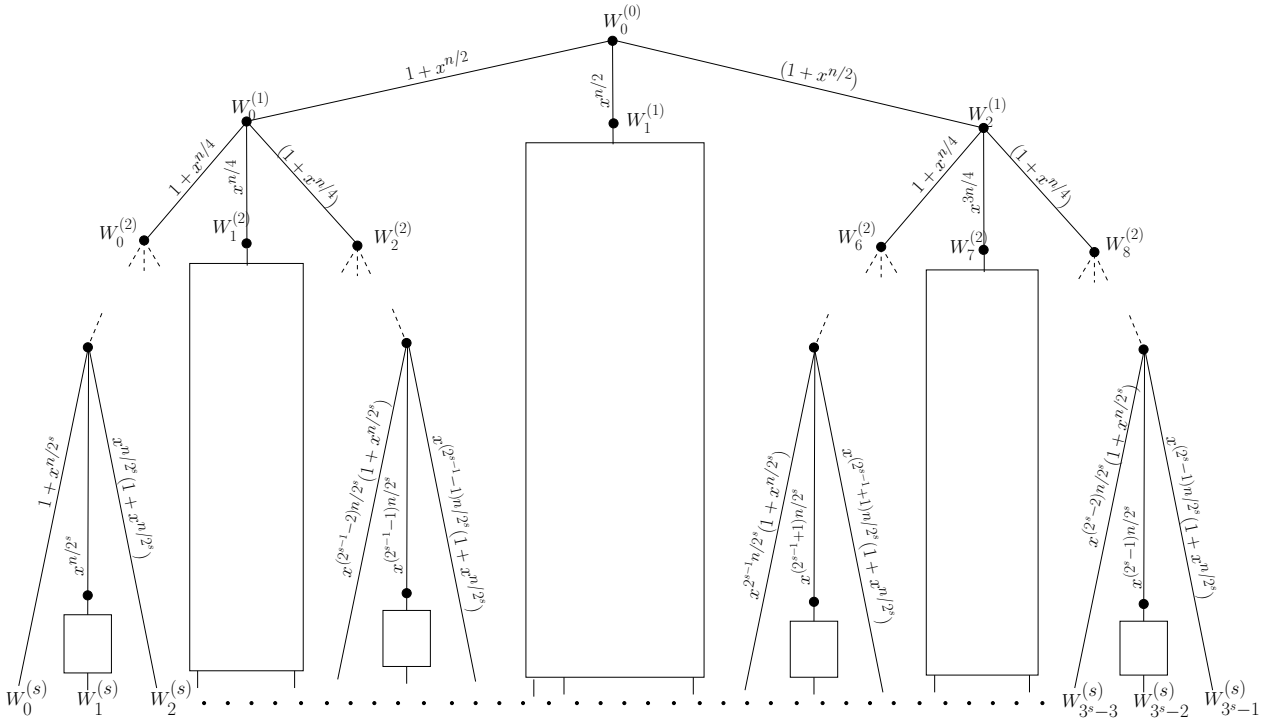
In [5], in order to generalize Bernstein reconstruction we extended the modifications done on the reconstruction three of depth 2 (cf. Subsection 3.3.1). We first defined the L/R-then-M nodes: such a node N is connected from the root to N with only left or right links and ends by a middle link. We replaced the sub-tree below each L/R-then-M node by a box representing a recursive reconstruction. We also moved the factors $x^{n/2^h}$ of the links down to the lowest possible links. This modified reconstruction tree is depicted in Figure 3.8.

The GBR_s algorithm. We used this modification of the reconstruction tree to derive a generalized Bernstein's reconstruction (GBR_s) of depth s. The values of the modified reconstruction tree are accumulated depth by depth, starting from the leaves and climbing up to the root. In order to go over the L/R-then-M nodes of depth h we needed the function $\sigma\colon \mathbb{N} \to \mathbb{N}$ defined as follows: for an integer i with binary representation $i = \sum_{k=0}^{\ell} i_k 2^k$ with $i_k \in \{0, 1\}$ the image of i by σ is

$$\sigma(i) = \sum_{k=0}^{\ell} 2 i_k 3^k.$$

We then proposed in [5] to reconstruct the product W based on the modified reconstruction tree as follows:

Figure 3.8: Modified reconstruction tree of depth s



- *Step 1: Initialization.* We accumulate, in an intermediate value Y, the values of the L/R leaves $W_{\sigma(i)}^{(s)}$, $i = 0, \ldots, 2^s - 1$, multiplied by their respective factors $x^{\frac{in}{2^s}}$.
- A loop on the depth h starting from s and going down to 1, where we perform the three following operations:
  - *Step 2: Multiplication by the depth factor:* we multiply Y by $(1 + x^{2^h})$.
  - *Step 3: Reconstruction of the L/R-then-M terms of depth h.* We reconstruct the L/R-then-M terms $W_{3\sigma(i)+1}^{(h)}, i = 0, \ldots, 2^{h-1} - 1$, of depth h by applying GBR_{s-h} to the leaves of each sub-tree with root $W_{3\sigma(i)+1}^{(h)}$.
  - *Step 4: Accumulation of the reconstructed L/R-then-M terms.* We accumulate into Y the values of the reconstructed L/R-then-M nodes $W_{3\sigma(i)+1}^{(h)}$ multiplied by their respective factors $x^{\frac{in}{2^{h-1}}+\frac{n}{2^h}}$.

The above method is specified in Algorithm 2. We proved its validity in [5].

In Table 3.3 we review the complexities of the best known approaches to multiply two polynomials of size $n = 2^s$. We report the complexities of the optimized Karatsuba formulas with BR1 and BR2. We also report the complexity results of Fan *et al.* [73] which has the best delay among such subquadratic multipliers. We also report complexity results for TMVP multiplier, since the best known subquadratic multipliers for extended binary fields use Toeplitz-matrix vector product (TMVP) approach [46].

The results of Table 3.3 show that the generalization of Bernstein's approach provides multipliers with improved space complexities. This was the initial goal of Bernstein. But surprisingly

Algorithm 2 GBR_s

Require: $W_0^{(s)}, \ldots, W_{3^s-1}^{(s)}$
Ensure: $Y = \mathrm{GBR}_s(W_0^{(s)}, \ldots, W_{3^s-1}^{(s)})$
$Y \leftarrow (\sum_{i=0}^{2^s-1} W_{\sigma(i)}^{(s)} x^{\frac{in}{2^s}})$ *//Step 1: Initialization*
for $h = s$ **to** 1
  $Y \leftarrow Y \times (1 + x^{\frac{n}{2^h}})$ *//Step 2: Multiplication of depth h*
  for $i = 0$ **to** $2^{h-1} - 1$
    $j \leftarrow 3\sigma(i) + 1$
    $Z_i \leftarrow \mathrm{GBR}_{s-h}(W_{3^{s-h}j}^{(s)}, \ldots, W_{3^{s-h}j+3^{s-h}-1}^{(s)})$ *//Step 3 : Reconstruction of the L/R-then-M terms*
  end for
  $Y \leftarrow Y + x^{\frac{n}{2^h}} \left(\sum_{i=0}^{2^{h-1}-1} Z_i x^{\frac{in}{2^{h-1}}}\right)$ *//Step 4 : Accumulation of the L/R-then-M terms*
end for
return(U)

Table 3.3: Space and time complexities of Karatsuba multipliers

Operat.	Method	Form of n	# AND	# XOR	Delay
TMVP	Fan-Hasan [46]	2^k	$n^{\log_2(3)}$	$5.5n^{\log_2(3)} - 6n + 0.5$	$D_A + 2\log_2(n)D_X$
Poly. Mult.	Original Karatsuba [76]	2^k	$n^{\log_2(3)}$	$6n^{\log_2(3)} - 8n + 2$	$D_A + 3\log_2(n)D_X$
	Fan *et al.* [73]	2^k	$n^{\log_2(3)}$	$6n^{\log_2(3)} - 8n + 2$	$D_A + 2\log_2(n)D_X$
	Karatsuba with BR1 [48, 75, 74]	2^k	$n^{\log_2(3)}$	$5.5n^{\log_2(3)} - 7n + 1.5$	$D_A + 3\log_2(n)D_X$
	Karatsuba with BR2 [48]	2^{2k}	$n^{\log_2(3)}$	$5.43n^{\log_2(3)} - 6.8n + 1.375$	$D_A + 2.5\log_2(n)D_X$
	Proposed with $s = 3$	2^{3k}	$n^{\log_2(3)}$	$5.37n^{\log_2(3)} - 6.68n + 1.30$	$D_A + 2.33\log_2(n)D_X$
	Proposed with $s = 4$	2^{4k}	$n^{\log_2(3)}$	$5.34n^{\log_2(3)} - 6.61n + 1.27$	$D_A + 2.25\log_2(n)D_X$
	Proposed with $s = \log_2(n)$	2^k	$n^{\log_2(3)}$	$5.25n^{\log_2(3)} - 6n + 0.75 - 0.5\log_2(n)$	$D_A + (2\log_2(n) + 1)D_X$

this also reduces the delay of the resulting multiplier. The results in Table 3.3 even show that, as s increases, the leading terms of the complexities (space and time) slowly approach the leading term of the complexity of the multiplier obtained with a GBR applied to a recursive reconstruction of depth $s = \log_2(n)$. For example the leading term of the total number of gates evolves as follows when s increases:

$$\begin{array}{ccccccccc} \text{BR1} & & \text{BR2} & & \text{GBR with } s=3 & & \text{GBR with } s=4 & & \text{GBR with } s=\log_2(n) \\ 6.5n^{\log_2(3)} & \rightarrow & 6.43n^{\log_2(3)} & \rightarrow & 6.37n^{\log_2(3)} & \rightarrow & 6.34n^{\log_2(3)} & \cdots \rightarrow & 6.25n^{\log_2(3)} \\ +O(n) & & +O(n) & & +O(n) & & +O(n) & & +O(n) \end{array}$$

We also notice that the improvements provided by the generalized Bernstein reconstruction render the polynomial approach better than the two-way split TMVP approach considering both space and time complexities.

3.4 Conclusion

In this chapter we first briefly introduced the best approach for the design of subquadratic multiplier in $\mathbb{F}_{2^n}$. These multipliers are based either on TMVP approach or on Karatsuba formula for polynomial multiplication. We presented two results concerning multipliers based on TMVP: the first one extend the approach of Fan-Hasan [46] to a larger class of fields, while

the second reduces significantly the space requirement of the multiplier of Fan-Hasan based on ONB-II. In this later work we introduced a technique based on block recombination. This technique consists in re-expressing the full circuit given by the TMVP approach by removing some redundant computations.

Finally we looked at multipliers based on several recursions of Karatsuba formula and then we tried to improve the reconstruction of the results by extending the approach of Bernstein. This approach lead us to a result which makes multipliers based on polynomial multiplier better than the one based on TMVP.

Chapter 4

Implementation of the GHASH function of the Galois counter mode

The Galois counter mode (GCM) [49] is a ciphering mode for symmetric key encryption associated to a block-cipher $E_K\colon \{0,1\}^n \to \{0,1\}^n$. This mode encrypts $\ell - 1$ plaintext blocks $M_1, \ldots, M_{\ell-1}$ by initializing a counter $counter_0 = IV$ with a random initial vector IV. Each block M_i is then encrypted as

$$C_i = M_i \oplus E_K(counter_0 + i) \text{ for } i = 1, \ldots, \ell - 1. \tag{4.1}$$

The authentication tag depends on all M_i and the key K and is generated from the hash-key $H = E_K(0)$ and the ℓ blocks $X_1 = M_1, \ldots, X_{\ell-1} = M_{\ell-1}, X_\ell = \text{len}(C)$ as follows:

$$GHASH(X_1, \ldots, X_\ell, H) \oplus E_K(IV) = \underbrace{\left(X_1 H^\ell \oplus X_2 H^{\ell-1} \oplus \cdots \oplus X_{\ell-1} H^2 \oplus X_\ell H\right)}_{(*)} \oplus E_K(IV)$$

where the blocks $X_i, i = 1 \ldots, \ell$ and H in $(*)$ are considered as elements of $\mathbb{F}_{2^n}$. Figure 4.1 describes the overall processing of GCM.

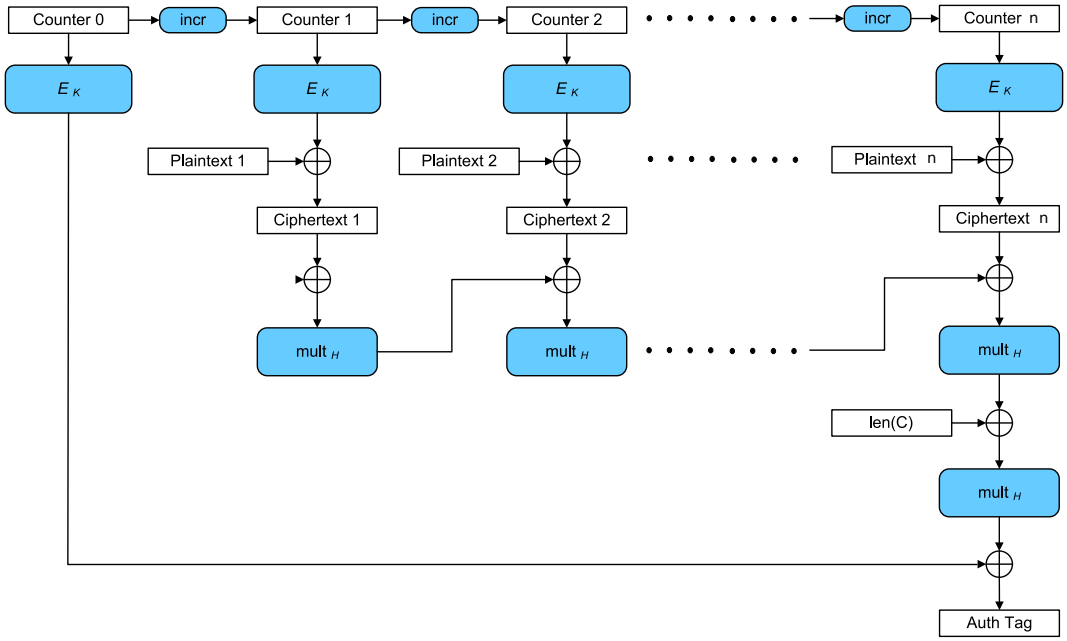
In the remaining of this section we will focus on efficient computation of the GHASH function. The basic approach for computing the GHASH function consists in a sequence of ℓ multiply-and-add operations (cf. Algorithm 3).

Algorithm 3 GHASH function

Require: $X_1, X_2, \ldots, X_\ell$, H in $\mathbb{F}_{2^n}$
Ensure: $GHASH_H(X)$
  $Y \leftarrow 0$
  for $i = 1$ **to** ℓ **do**
    $Y \leftarrow (Y + X_i) \times H$
  end for
  return Y

Note that Algorithm 3 involves ℓ multiplications and ℓ additions in the field $\mathbb{F}_{2^n}$. For hardware implementation, where we have only one multiplier and one adder, these operations are performed in sequence and the total computation time for GHASH is approximately ℓ times the combined delay of a multiplication and an addition. In binary fields, the delay of an

Figure 4.1: Galois counter mode of operation



addition is exactly that of a bitwise XOR operation. In Table 4.1, we summarize various bit-parallel multiplication methods and their space and time complexities in terms of gate counts and gate delays.

Multiplier	#AND	#XOR	Gate delay (D_M)
Karatsuba with GBR [5]	$O(n^{1.58})$	$O(n^{1.58})$	$(2\log_2(n)+\delta)D_X + D_A$
Fan-Hasan [46]	$O(n^{1.58})$	$O(n^{1.58})$	$(2\log_2(n)+\delta)D_X + D_A$
Mastrovito [59]	$O(n^2)$	$O(n^2)$	$(\log_2(n)+\delta)D_X + D_A$

Table 4.1: Complexities of various multipliers over $\mathbb{F}_{2^n}$ where D_X (resp. D_A) is the delay of a two-input XOR (resp. AND) gate (δ is a small constant).

4.1 Improvement of parallel implementation of GHASH

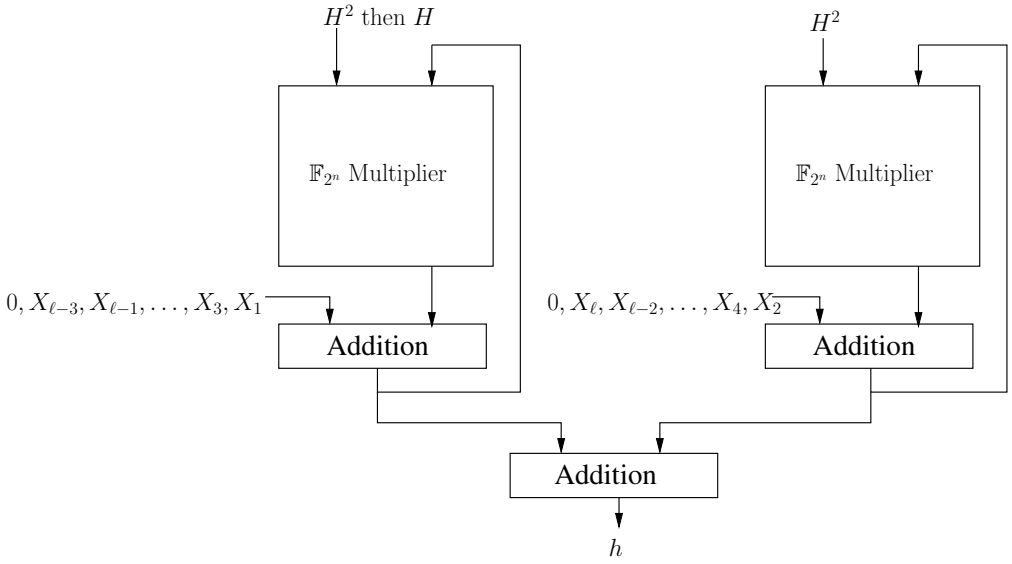
The main attractive property of GCM is that it is highly parallelizable. Indeed, in counter mode the encryption of the different blocks (4.1) are independent and can be thus processed in parallel. It was also noticed in [49, 77] that the GHASH function can be parallelized. For example, if ℓ, the number of blocks X_i, is even we can process even and odd blocks X_{2i} and X_{2i+1} in two parallel GHASH computations as follows:

$$\begin{aligned}\text{GHASH}(X_1, X_2, \ldots, X_{\ell-1}, X_\ell, H) &= \text{GHASH}(X_1, X_3, \ldots, X_{\ell-3}, X_{\ell-1}, H^2) \times H \\ &\quad +\text{GHASH}(X_2, X_4, \ldots, X_{\ell-2}, X_\ell, H^2).\end{aligned}$$

Such approach leads to the architecture shown in Figure 4.2 for a two-level parallelization of GHASH: we have two parallel multipliers and two adders. With $\ell/2$ clock cycles this architecture computes $\text{GHASH}(X_1, X_3, \ldots, X_{\ell-3}, X_{\ell-1}, H^2)$ through one multiplier and one adder and

concurrently computes GHASH$(X_2, X_4, \ldots, X_{\ell-2}, X_\ell, H^2)$ with the second multiplier and the second adder. At the very end an addition is performed before outputting the result.

Figure 4.2: Even-odd GHASH architecture [77, 78]



In a joined work with Anwar Hasan, Nicolas Méloni and Ashkan Namin [3] we showed that one of the block recombination discussed in Subsection 3.2.2 can be advantageously used for parallel implementation of the GHASH function. Before proceeding to the block recombination of the parallel GHASH architecture, we need to rewrite the GHASH computation as a sequence of two-multiplication-and-add as described in Algorithm 4.

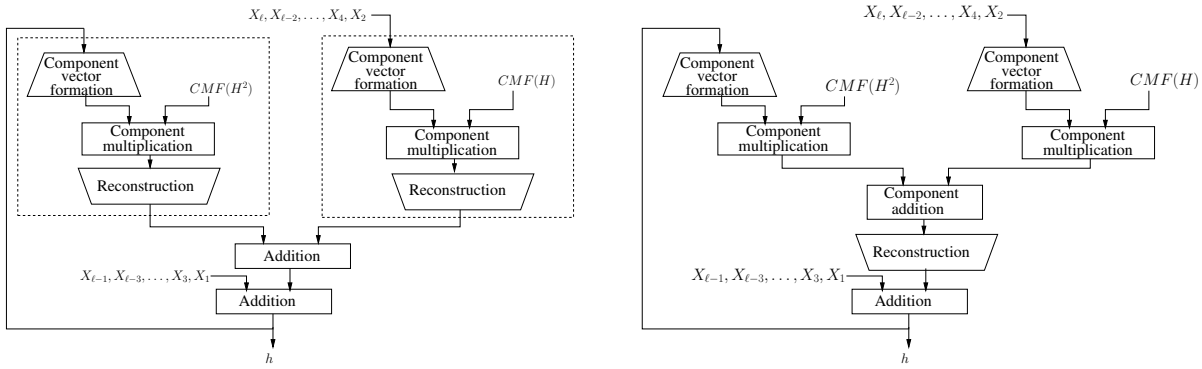
Algorithm 4 GHASH by two multiplications and add

Require: $C = (X_1, \ldots, X_\ell)$ where $X_i \in \mathbb{F}_{2^n}$ and $H \in \mathbb{F}_{2^n}$
Ensure: $h = \text{GHASH}(C)$
  $H' \leftarrow H^2$
  $h \leftarrow 0$
  for $i = \ell/2 - 1$ **to** 0 **do**
    $h \leftarrow h \times H' + X_{2i+2}H + X_{2i+1}$
  end for
  $h \leftarrow h \times H$
  return h

A direct architecture based on Algorithm 4 is shown in Subfigure 4.3a: it consists of two multipliers performing in parallel $h \times H^2$ and $X_{2i+2}H$, these two products are then added to X_{2i+1} to produce the intermediate value of h.

In [3] we assumed that the multipliers of Subfigure 4.3a are subquadratic space complexity multipliers based on TMVP with $CMF(H)$ and $CMF(H^2)$ precomputed. In [3] we proposed to recombine the multiplier by performing the addition $(h \times H^2) + (X_{2i+2}H)$ in component formation and then apply only one reconstruction to get $h \times H^2 + X_{2i+2}H$. The corresponding recombined hardware architecture using this approach is shown in the right side of Subfigure 4.3b.

Figure 4.3: Two-multiply-and-add GHASH architecture



(a) Based on Algorithm 4

(b) Recombined

Comparison of the two approaches. The gain in terms of space complexity is as follows: we replace one reconstruction block by a component addition. The proposed architecture get a slight penalty in the delay of computation: one additional XOR appears on the critical path of the architecture compared to Figure 4.2. The complexities of the recombined and non-recombined architectures based on TMVP are reported in Table 4.2.

Table 4.2: Complexity comparison of GHASH architectures based on TMVP multiplier

Archi.	Space complexity		Time complexity
	#XOR	#AND	
Figure 4.3(a)	$6n^{\log_2(3)} - 4n$	$2n^{\log_2(3)}$	$(2\log_2(n) + 2)D_X + D_A$
Figure 4.3(b)	$5n^{\log_2(3)} - 3n$	$2n^{\log_2(3)}$	$(2\log_2(n) + 2)D_X + D_A$

This approach can be generalized to a larger level r of parallelization. Without going into details the gain in space is roughly $r-1$ times the gain for $r = 2$. The penalty on delay is more important since it adds $\lceil \log_2(r) \rceil D_X$ to the critical path delay.

4.2 Computing GHASH using a characteristic polynomial

In the previous section we reviewed a technique which leads to a saving in the space complexity of the parallel implementation of a GHASH computation. In this section we present the strategy of [7, 22] which reduces the critical path delay of one iteration in the GHASH computation. This work was done in 2009-2010 in collaboration with Nicolas Méloni and Anwar Hasan during the time when Nicolas Méloni was a post-doctoral fellow at the University of Waterloo.

Let $H \in \mathbb{F}_{2^n}$ be the hash-key used in GHASH computation. There exists a degree n polynomial $\chi_H(T) = \sum_{i=0}^{n} c_i T^i \in \mathbb{F}_2[T]$, i.e., with coefficients $c_i \in \mathbb{F}_2$, such that $\chi_H(H) = 0$ and it is called the characteristic polynomial of H over $\mathbb{F}_2$. Furthermore, it can be shown for $H \neq 0$ that $c_n = c_0 = 1$. Let us see how to use such a polynomial to speed-up the computation for long GHASH computation.

Using the fact that $\chi_H(H) = 0$ and since $c_n = 1$ we deduce that $H^n = \sum_{i=0}^{n-1} c_i H^i$. Using this expression of H^n, we can reduce a polynomial in H of degree n to a polynomial of degree

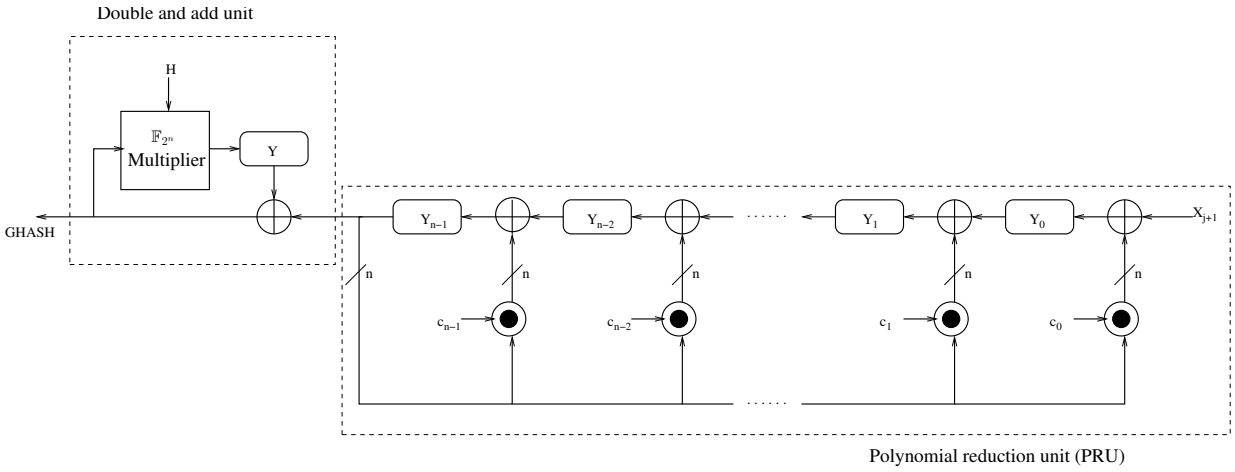


Figure 4.4: Implementation of the GHASH function using a PRU

$n-1$. Indeed, let $P = X_1H^n + X_2H^{n-1} + \cdots + X_nH$ with $X_i \in \mathbb{F}_{2^n}$, then

$$P \mod \chi_H = (X_2 + c_{n-1}X_1)H^{n-1} + (X_3 + c_{n-2}X_1)H^{n-2} + \cdots + (X_n + c_1X_1)H + c_0X_1.$$

It is important to note that all the c_is are in $\mathbb{F}_2$, which means that computing $(X_{n-i+1} + c_iX_i)$ is just an addition over $\mathbb{F}_{2^n}$ when $c_i = 1$. This reduction modulo χ_H of a polynomial in H of degree n can be performed with the polynomial reduction unit (PRU) of the architecture shown in Figure 4.4.

If we now consider a polynomial $P = X_1H^\ell + X_2H^{\ell-1} + \cdots + X_\ell H$ in H of degree ℓ, this polynomial can be decomposed as follows

$$P = ((\cdots((X_1H^n + X_2H^{n-1} + \cdots + X_{n+1})H + X_{n+2})H + \cdots + X_{\ell-1})H + X_\ell)H.$$

We can reduce P modulo χ_H by sequentially performing $\ell - n$ reductions of a degree n polynomial as follows

$$\begin{aligned} P = ((\cdots((X_1H^n + X_2H^{n-1} + \cdots + X_{n+1} \mod \chi_H)H \\ +X_{n+2} \mod \chi_H)H + \cdots + X_{\ell-1} \mod \chi_H)H \\ +X_\ell \mod \chi_H)H \mod \chi_H. \end{aligned}$$

Each reduction modulo χ_H can be done through the PRU of Figure 4.4. The PRU is originally loaded with $X_1, \ldots, X_n$ and then it is sequentially updated and fed with $X_{n+1}, \ldots, X_\ell$. The output of the polynomial reduction unit (PRU) is a polynomial of degree $n-1$ in H. The output of the PRU can then be sent to a multiply-and-add unit in order to compute the desired GHASH value, which is an element of $\mathbb{F}_{2^n}$. This is the approach used in the architecture shown in Figure 4.4. Specifically, in this architecture, the PRU is updated $\ell - n$ times through the feedback connection. Then, during the next n clock cycles, the feedback connections are not active; rather the contents of the PRU registers (i.e., Y_i, $i = n-1, n-2, \cdots, 0$) sequentially enter the multiply-and-add unit of the architecture.

In Algorithm 5 below, we give the operations performed in the circuit shown in Figure 4.4.

Complexity. The architecture in Figure 4.4 has a space complexity

$$\mathcal{S} = (n^2 + n)S_X + n^2S_A + S_M.$$

where S_X and S_A represents the space requirement of an XOR and an AND gates and S_M the complexity of a multiplier. Since the critical path of the PRU is equal to $D_X + D_A$ and the

Algorithm 5 GHASH$_H(X)$ using χ_H

Require: $X = X_1X_2 \dots X_\ell$ and $\chi_H(T) = \sum_{i=0}^{n} c_i T^i$
Ensure: GHASH$_H(X) = X_1H^\ell + X_2H^{\ell-1} + \cdots + X_\ell H$
  $Y_{\ell-1} \dots Y_0 \leftarrow X_1 \dots X_n$
  $temp \leftarrow 0$
  for $j = n$ **to** $\ell - 1$ **do**
    $C \leftarrow Y_{n-1}$
    $Y_i \leftarrow Y_{i-1} + c_iC, 1 \le i \le n-1$
    $Y_0 \leftarrow X_{j+1} + c_0C$
  end for
  for $i = n-1$ **down to** 1 **do**
    $temp \leftarrow (temp + Y_i) \times H$
  end for
  return $(temp + Y_0)$

critical path of the multiplier is D_M, the total time delay of the architecture shown in Figure 4.4 is equal to

$$\mathcal{D} = (\ell - n)(D_X + D_A) + (n-1)(D_M + D_X).$$

For long messages, the value of ℓ is expected to be much longer than that of n, e.g., if the size of X is 1MBytes, then $n \cong 2^{16}$ and the reduction modulo χ_H dominates the computation time. The overall computation time is better than the one of multiply-and-add approach (Algorithm 3).

The characteristic polynomial χ_H can be precomputed and used during a full GCM session. Consequently, the corresponding complexity can be treated separately in the GHASH computation.

Computation of the characteristic polynomial χ_H. We focus on the computation of the characteristic polynomial $\chi_H[T] \in \mathbb{F}_2[T]$ of the hash-key H. We first recall the method of Gordon [79] that determines the characteristic polynomial of an element of a finite field and then we propose a method which takes advantage of the tower structure of $\mathbb{F}_{2^n}$ when $n = 2^s$ that can be faster than the method of Gordon depending on the representation of the finite field.

Gordon's method works as follows: let $H \in \mathbb{F}_{2^n}$, the characteristic polynomial of H is given by

$$\chi_H(T) = \prod_{i=0}^{n-1} (T + H^{2^i}).$$

Gordon evaluates the above product by initializing $\chi_H \leftarrow 1$ and then sequentially performing $\chi_H \leftarrow \chi_H \times (T + H^{2^i})$ for $i = 0, \dots, n-1$. This method requires $n(n+1)/2$ multiplications and n squarings in $\mathbb{F}_{2^n}$.

Gordon's method is quite general and can be applied to any binary extension field $\mathbb{F}_{2^n}$. We now propose a method that takes into account that the field used in practice in GCM, i.e., $\mathbb{F}_{2^{128}}$, has a very special structure since the degree $128 = 2^7$.

In [7, 22] we used the following lemma which states the basic property for the proposed computation of the polynomial χ_H. Let $P = \sum_{i=0}^{d} p_i T^i$ be a polynomial in $\mathbb{F}_{2^n}[T]$ and k be an integer. We denote

$$\sigma_k(P) = \sum_{i=0}^{d} p_i^{2^k} T^i.$$

Lemma 1. *Let $\mathbb{F}_{2^n}$ be a binary field, and let $\mathbb{F}_{2^{2n}}$ be a degree 2 field extension of $\mathbb{F}_{2^n}$. The following assertions hold*

(i) If $P, Q \in \mathbb{F}_{2^n}[T]$ and k an is integer then

$$\sigma_k(PQ) = \sigma_k(P)\sigma_k(Q).$$

(ii) If $P \in \mathbb{F}_{2^{2n}}[T]$ satisfies $P(H) = 0$ for a given element $H \in \mathbb{F}_{2^{2n}}$, then the polynomial $Q = P \times \sigma_n(P)$ satisfies

$$Q(H) = 0 \text{ and } Q \in \mathbb{F}_{2^n}[T].$$

This lemma leads to the computation of a characteristic polynomial for $H \in \mathbb{F}_{2^n}$ where $n = 2^s$. Starting from $P = T + H$ and applying iteratively the point (ii) of Lemma 1 we generate $\chi_H^{(k)}(T)$ the characteristic polynomial of H over $\mathbb{F}_{2^{2^k}}$ for $k = s-1, s-2, \ldots, 0$. This method is detailed in Algorithm 6.

Algorithm 6 Computing the characteristic polynomial of H

Require: $H \in \mathbb{F}_{2^{2^s}}$
Ensure: P_H (the characteristic polynomial of H)
  $P \leftarrow T + H$
  for $k = s - 1$ **downto** 0 **do**
    $P \leftarrow P \times \sigma_{2^k}(P)$
  end for
  return (P)

In [22] we evaluated the complexity of Algorithm 6. We found that the overall cost in computing the characteristic polynomial of $H \in \mathbb{F}_{2^{2^s}}$ over $\mathbb{F}_2$ is $3sM_{2^s} + 2^{s+1}S_{2^s}$ where M_{2^s} and S_{2^s} represent a multiplication and a squaring in $\mathbb{F}_{2^{2^s}}$, respectively. In the case of GCM, $s = 7$ and thus the number of field operations is 21 multiplications and 256 squarings over $\mathbb{F}_{2^{128}}$ which is more efficient than the method of Gordon.

4.3 Conclusion

In this chapter we reviewed the Galois counter mode which encrypts a message through counter-mode of a given block-cipher and also generates an authentication tag. This authentication tag is computed through a sequence of multiplications and additions in $\mathbb{F}_{2^n}$. We showed that we could use a block recombination strategy presented in the Chapter 3 to reduce the space requirement of the parallel architecture computing the authentication tag. We also presented a strategy using the characteristic polynomial of the hash-key which speeds up the computation of the hash-tag. For this later strategy the simulations for FPGA done by Jithra Adikari showed that unfortunately the proposed approach was not competitive in practice. This was due to the large fan-out of the circuit. This approach might be further analyzed and modified in order to correct this drawback.

Chapter 5

Multiplication in $\mathbb{F}_p$ and $\mathbb{F}_{p^k}$

We consider in this section efficient multiplication in prime field $\mathbb{F}_p$ and prime field extension $\mathbb{F}_{p^k}$. Multiplication in $\mathbb{F}_p$ consists in integer multiplication followed by a reduction modulo a prime integer p. We will present in Section 5.1 some contributions related to the modular number system (MNS) [50] for the representation of the elements of $\mathbb{F}_p$. Then in Section 5.2 we will review some results concerning randomized arithmetic in residue number system (RNS) [80]. We first review classical methods for integer multiplication and reduction modulo p.

Integer multiplication. The following approaches are commonly used to compute efficiently an integer multiplication:

- *Schoolbook multiplication.* This method is the one taught in elementary school. Suppose we have $u = \sum_{i=0}^{n-1} u_i\beta^i$ and $v = \sum_{i=0}^{n-1} v_i\beta^i$ expressed in base β (for us $\beta = 2^t$ with $t = 1, 8, 16, 32$ or 64). The multiplication $w = u \times v$ is done by expanding the product relatively to $u = \sum_{i=0}^{n-1} u_i\beta^i$ and this results in n multiplications $u_i \times v$ which are accumulated to get w.

- *Karatsuba multiplication.* The principle is the same as the one reviewed in Section 3.3 for binary polynomial. Indeed, we split into two parts $u = u_0 + \beta^{\lceil n/2 \rceil} u_1$ and $v = v_0 + \beta^{\lceil n/2 \rceil} v_1$ and we re-express $w = u \times v$ in terms of three products of half-size integers:

$$w = u_0v_0 + ((u_0 + u_1)(v_0 + v_1) - u_0v_0 - u_1v_1)\beta^{\lceil n/2 \rceil} + u_1v_1\beta^{2\lceil n/2 \rceil}.$$

  This approach is generally performed recursively until we reach an integer size where schoolbook multiplication is efficient.

- *Lagrange and DFT approaches.* One can transpose the expression of an integer $u = \sum_{i=0}^{n-1} u_i\beta^i$ in base β to a polynomial $\mathbf{u}(x) = \sum_{i=0}^{n-1} u_i x^i$. We get back to the integer u by evaluating $\mathbf{u}(x)$ in β. We can then transform the integer multiplication $u \times v$ into a product of polynomials $\mathbf{u}(x) \times \mathbf{v}(x)$ where $\mathbf{u}(x) = \sum_{i=0}^{n-1} u_i x^i$ and $\mathbf{v}(x) = \sum_{i=0}^{n-1} v_i x^i$. Furthermore if the polynomials are considered as elements of $\mathbb{Z}/m\mathbb{Z}[x]$ one can take advantage of the Chinese remainder theorem (CRT) which asserts that the following application is an isomorphism:

$$\begin{array}{ccc} \mathbb{Z}/m\mathbb{Z}[x]/(\prod_{i=1}^{d}(x-\alpha_i)) & \longrightarrow & \mathbb{Z}/m\mathbb{Z}[x]/(x-\alpha_1) \times \cdots \times \mathbb{Z}/m\mathbb{Z}[x]/(x-\alpha_d) \\ \mathbf{u}(x) & \longmapsto & (\mathbf{u}(x) \bmod (x-\alpha_1), \ldots, \mathbf{u}(x) \bmod (x-\alpha_d)). \end{array} \tag{5.1}$$

  This isomorphism (5.1) re-expresses a polynomial $\mathbf{u}(x)$ by its evaluation at α_i for $i = 1, \ldots, d$ often called the Lagrange representation of $\mathbf{u}(x)$. Indeed, the computation of

$\mathbf{u} \mod (x - \alpha_i)$ is equal to $\mathbf{u}(\alpha_i)$. If the multiplication $\mathbf{w}(x) = \mathbf{u}(x) \times \mathbf{v}(x)$ is done in Lagrange representation it consists in d pairwise multiplication modulo m. If the product $\mathbf{w} = \mathbf{u}(x) \times \mathbf{v}(x)$ has degree $< d$ and coefficients $w_i < m$ then the Lagrange interpolation applied to $\mathbf{w}(\alpha_i), i = 1, \ldots, d$ correctly reconstruct $\mathbf{w}(x)$ and leads to the correct product $\mathbf{w}(x) = \mathbf{u}(x) \times \mathbf{v}(x)$ and then to $w = u \times v$.

A particular case, widely used in practice, is when α_is are the n-th of unity. In this case, the multi-point evaluation is referred to as the discrete Fourier transform (DFT). This particular case is interesting since the interpolation can also be expressed as a multi-point evaluation, and performed efficiently with the fast Fourier transform (FFT) algorithm when $n = 2^s$. This approach is used for example in the Schönhage-Strassen [81] algorithm for integer multiplication.

Integer reduction. We have to reduce modulo p the product $w = u \times v$ which is in $[0, p^2[$. This reduction modulo p consists of the computation of r the remainder of the division of w by p:

$$r = w - \lfloor w/p \rfloor \times p.$$

In practice two cases arise: if p has a sparse form and if p has not a sparse form. These two cases are discussed below:

- *p has a sparse form.* Such primes p are often used in elliptic curve cryptography: in particular, NIST standard [35] recommends the primes listed in Table 5.1. One can perform the reduction with a few additions or subtractions: for example a reduction of $w \in [0, 2^{1042}]$ modulo the NIST prime p_{521} consists in splitting w into two parts: $w = w_0 + 2^{521} w_1$ with $w_0 < 2^{521}$ and since $2^{521} \equiv 1 \mod p_{521}$ we get $w_0 + w_1 \equiv w \mod p$.

Table 5.1: NIST recommended primes for elliptic curve cryptography

p_{192}	$2^{192} - 2^{64} - 1$
p_{224}	$2^{224} - 2^{64} - 1$
p_{256}	$2^{256} - 2^{64} - 1$
p_{384}	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
p_{521}	$2^{521} - 1$

- *p has not a sparse form.* If p is not sparse the two main approaches are the following: the approach of Barrett [82] which clears the upper half part of w and the method of Montgomery [51] which clears the lower half part of w. The approach of Barrett approximates the quotient $\lfloor w/p \rfloor$ as follows:

$$\widetilde{q} = \left\lceil \frac{\lceil w/\beta^{n-1} \rceil \times \lceil \beta^{2n}/p \rceil}{\beta^{n+1}} \right\rceil$$

The above computations of $\widetilde{q}$ consists in right shiftings for the division by β^{n-1} and β^{n+1} and a multiplication by a constant $\mu = \lceil \beta^{2n}/p \rceil$. This results in an approximated $\widetilde{r}$

$$\widetilde{r} = w - \widetilde{q}p$$

which from [82] satisfies $\widetilde{r} \leq 3p$.

The approach of Montgomery clears the least significant bits of w by first computing

$$q = w \times (-p^{-1}) \mod \beta^n$$

which then implies that $w + qp \equiv 0 \mod \beta^n$ which can be divided by β^n. This method is detailed in Algorithm 7.

Algorithm 7 Montgomery multiplication

Require: A modulus p, two integers $u, v \in [0, p-1]$, $M = \beta^n$ such that $M > p$ and $\gcd(M, p) = 1$ and the precomputed integer $p' = -p^{-1} \mod M$
Ensure: $r = (u \times v \times M^{-1}) \mod p$ and $r \in [0, p-1]$

```
w ← u × v
q ← w × p' mod M
r ← (w + p × q)/M
if r ≥ p then
  r ← r − p
end if
return r
```

5.1 Modular Number System

Efficient arithmetic modulo a prime integer p is generally related to the system of representation. Here we are interested in integer multiplication modulo a prime integer p of cryptographic size $p \in [2^{160}, 2^{4048}]$ like the one used in elliptic curve cryptography or digital signature algorithm (DSA) [35]. The authors in [50] introduced a variant of the base β for integer representation: the *Modular Number System.*

Definition 1 (MNS [50])**.** *A Modular Number System (MNS) $\mathcal{B}$, is a quadruple (p, n, γ, ρ), such that for all positive integers $0 \leq u < p$ there exists a polynomial $\mathbf{u}(x) = \sum_{i=0}^{n-1} u_i x^i$ such that*

$$\begin{aligned} &\mathbf{u}(\gamma) = u \bmod p, \\ &\deg(\mathbf{u}(x)) < n, \\ &\|\mathbf{u}\|_\infty < \rho. \end{aligned} \tag{5.2}$$

The polynomial $\mathbf{u}(x)$ is a representation of u in $\mathcal{B}$.

In practice an MNS has a base $\gamma \cong p$ and a coefficient bound $\rho \cong p^{1/n}$. In other words, compared to the usual β representation, MNS enlarge the possible values for the base element γ and makes it possible to find γ yielding a more efficient arithmetic modulo p.

Example 1. In Table 5.2, we list the representation of the integer modulo $p = 17$ and this proves that the quadruple $(17, 3, 7, 2)$ is an MNS for $p = 17$.

Table 5.2: The elements of $\mathbb{Z}_{17}$ in $\mathcal{B} = MNS(17, 3, 7, 2)$

0	1	2	3	4	5	6	7	8	9
0	1	$-x^2$	$1 - x^2$	$-1 + x + x^2$	$x + x^2$	$-1 + x$	x	$1 + x$	$-x - 1$

10	11	12	13	14	15	16
$-x$	$-x + 1$	$-x - x^2$	$1 - x - x^2$	$-1 + x^2$	x^2	-1

We can check that if we evaluate $(-1 + x + x^2)$ in γ, we have $-1 + \gamma + \gamma^2 = -1 + 7 + 49 = 55 \equiv 4 \bmod 17$. We have also $\deg(-1 + x + x^2) = 2 < 3$ and $\| - 1 + x + x^2\|_\infty = 1 < 2$.

The second definition of this section corresponds to a sub-family of the Modular Number System. The authors in [50] used the possibility to choose freely the base γ to focus on MNS yielding an efficient modular multiplication. The authors in [50] said that these systems are adapted to the modular arithmetic: they are *Adapted Modular Number System.*

Definition 2 (AMNS [50])**.** *A Modular Number System $\mathcal{B} = (p, n, \gamma, \rho)$ is said to be Adapted (AMNS) if there exists a small integer λ such that*

$$\gamma^n = \lambda \bmod p. \tag{5.3}$$

This means that the polynomial $E = x^n - \lambda$ has γ as root in $\mathbb{Z}/p\mathbb{Z}$:

$$E(\gamma) \equiv 0 \mod p. \tag{5.4}$$

We also denote $(p, n, \gamma, \rho)_E$ the Modular Number System (p, n, γ, ρ) which is adapted to the polynomial E.

The difficulty in the construction of an AMNS is to find an n-th root modulo p of a fixed *small* element $\lambda \in \mathbb{Z}/p\mathbb{Z}$. Since p is a prime, when such root exists the problem can be easily solved [83]. Since in this chapter we focus on arithmetic modulo a prime p, we assume that we can efficiently compute such γ.

As proposed by the authors in [50] the multiplication of two elements $\mathbf{u}(x)$ and $\mathbf{v}(x)$ in an AMNS is done in three steps: polynomial multiplication, polynomial reduction modulo $E(x)$ and coefficient reduction (cf. Algorithm 8). For the reduction of the coefficients of $\mathbf{w}'$ which lie in $]-n\rho^2\lambda, n\rho^2\lambda[$ we have to modify $\mathbf{w}'$ in order to get $\mathbf{r} = \text{CoeffRed}(\mathbf{w}')$ with coefficients in the range $]-\rho, \rho[$, i.e., $\mathbf{r}$ expressed in the AMNS $(p, n, \gamma, \rho)_E$.

Algorithm 8 Modular multiplication in AMNS

Require: $\mathbf{u}(x)$ and $\mathbf{v}(x)$ the AMNS representation of two integers modulo p
Ensure: $\mathbf{r}(x)$ such that $\mathbf{r}(\gamma) = \mathbf{u}(\gamma)\mathbf{v}(\gamma) \mod p$ and $\deg \mathbf{r}(x) < n$ and $||\mathbf{r}||_\infty < \rho$
1: $\mathbf{w}(x) \leftarrow \mathbf{u}(x) \times \mathbf{v}(x)$ // Polynomial multiplication
2: $\mathbf{w}'(x) \leftarrow \mathbf{w}(x) \mod E(x)$ // Polynomial reduction
3: $\mathbf{r} \leftarrow \text{CoeffRed}(\mathbf{w}')$ // Coefficient reduction

The first step can be done with usual methods for polynomial multiplication: school-book, Karatsuba, or DFT/FFT methods. The second step is quite easy: thanks to the form of E we have only to add the lower part of $\mathbf{w}$ with λ times the higher part of $\mathbf{w}$ to get $\mathbf{w}'$. The reduction of the coefficients, is for now the most complicated part. In [50] Bajard, Imbert and Plantard focused on AMNS $(p, n, \gamma, \rho)_E$ such that $2^k \cong \rho$ has a sparse AMNS representation $2^k = \sum_{i=0}^{n-1} \xi_i \gamma^i$ with all ξ_i small relatively to ρ. They could reduce the coefficient of $\mathbf{w}$ by splitting it as $\mathbf{w} = \mathbf{w}_0 + 2^k \mathbf{w}_1$ with $\|\mathbf{w}_0\|_\infty < 2^k$ and then replacing 2^k by $\sum_{i=0}^{n-1} \xi_i x^i$ as follows

$$\mathbf{w} = \mathbf{w}_0 + (\sum_{i=0}^{n-1} \xi_i x^i)\mathbf{w}_1.$$

This operation is repeated until one gets $\|\mathbf{w}\|_\infty < \rho$. But AMNS $(p, n, \gamma, \rho)_E$ providing 2^k with a sparse AMNS representation are not the general case. To deal with general AMNS Bajard *et al.* provide in [84] a method for coefficient reduction based on table look-up. For practical application some alternative method needed to be developed avoiding look-up table approach.

5.1.1 Montgomery multiplication in AMNS

In a joined work with Thomas Plantard [16] we instigated several strategies for arithmetic in AMNS. We first introduced a variant of the Montgomery reduction [51] for coefficient reduction in AMNS. Let us fix an AMNS $\mathcal{B} = (p, n, \gamma, \rho)_E$ and $M(x)$ a polynomial of degree $n-1$ such that

$$M(\gamma) = 0 \mod p, \quad \text{and } \gcd(M, E) = 1.$$

As we will see later, $M(x)$ must be chosen with small coefficients.

We first compute a product $\mathbf{w}(x) = \mathbf{u}(x) \times \mathbf{v}(x) \mod E$ where $\mathbf{u}$ and $\mathbf{v}$ are expressed in the AMNS $\mathcal{B} = (p, n, \gamma, \rho)_E$. We adapt the Montgomery reduction [51] to our context as follows: we fix a modulus m which in practice has a size close to ρ and such that $M' = -M^{-1} \mod (E, m)$ exists. We then compute $\mathbf{q} = \mathbf{w} \times M' \mod (E, m)$ and then if we set $\mathbf{r}' = \mathbf{w} + \mathbf{q} \times M \mod E$ we can check that

$$\mathbf{r}'(\gamma) = \mathbf{u}(\gamma) \times \mathbf{v}(\gamma) \mod p \quad \text{and} \quad \mathbf{r}' \mod m = 0.$$

This means that $\mathbf{r}'$ can be divided by m yielding the required reduction of the coefficients $\mathbf{r} = \mathbf{r}'/m$. This leads to Algorithm 9.

Algorithm 9 AMNS Multiplication

Require: $\mathbf{u}(x), \mathbf{v}(x) \in \mathcal{B} = AMNS(p, n, \gamma, \rho)_E$ with $E = x^n - \lambda$ with precomputed M such that $M(\gamma) \equiv 0 \pmod p$ an integer m and $M' = -M^{-1} \bmod (E, m)$
Ensure: $\mathbf{r}$ such that $\mathbf{r}(\gamma) = \mathbf{u}(\gamma)\mathbf{v}(\gamma)m^{-1} \bmod p$
$\mathbf{w} \leftarrow \mathbf{u} \times \mathbf{v} \bmod E$
$\mathbf{q} \leftarrow \mathbf{w} \times M' \bmod (E, m)$
$\mathbf{r} \leftarrow (\mathbf{w} + \mathbf{q} \times M \bmod E)/m$ // can be performed modulo m' larger than $\|\mathbf{r}\|_\infty$

The resulting polynomial $\mathbf{r}$ of Algorithm 9 satisfies $\mathbf{r}(\gamma) = \mathbf{u}(\gamma)\mathbf{v}(\gamma)m^{-1} \bmod p$. But we do not know whether it is correctly expressed in the AMNS, i.e., if the coefficients of $\mathbf{r}$ are smaller than ρ. This is the purpose of the following theorem which provides the parameters ρ and n such that

$$||\mathbf{u}||_\infty < \rho \text{ and } ||\mathbf{v}||_\infty < \rho \Longrightarrow ||\mathbf{r}||_\infty < \rho.$$

Theorem 3. *Let $\mathcal{B} = AMNS(p, n, \gamma, \rho)_E$ be an Adapted Modular Number System, M be a polynomial of $\mathcal{B}$ be such that $M(\gamma) \equiv 0 \pmod p$ and $\sigma = \|M\|_\infty$, and $\mathbf{u}(x), \mathbf{v}(x)$ be two elements of $\mathcal{B}$. If we have ρ and m such that*

$$\rho > 2|\lambda| n\sigma \quad \text{and} \quad m > 2|\lambda| n\rho \tag{5.5}$$

then the polynomial $\mathbf{r}$, output by Algorithm 9 with input $\mathcal{B}$,M,m,$\mathbf{u}$ and $\mathbf{v}$, is in the Adapted Modular Number System $\mathcal{B}$.

An important remark on Theorem 3 is that the size of ρ depends on $\sigma = \|M\|_\infty$. Specifically, if σ is small then ρ can also be taken small, i.e., ideally, close to $p^{1/n}$.

To construct such polynomial M with small σ we used a technique provided by lattice theory. We considered the lattice $\mathcal{L}(\vec{B})$ generated by the following row vectors

$$\vec{B} = \begin{pmatrix} p & 0 & 0 & 0 & \dots & 0 \\ -\gamma & 1 & 0 & 0 & \dots & 0 \\ -\gamma^2 & 0 & 1 & 0 & \dots & 0 \\ \vdots & & & \ddots & & \vdots \\ -\gamma^{n-2} & 0 & 0 & \dots & 1 & 0 \\ -\gamma^{n-1} & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \begin{matrix} \leftarrow p \\ \leftarrow x - \gamma \\ \leftarrow x^2 - \gamma^2 \\ \vdots \\ \leftarrow x^{n-2} - \gamma^{n-2} \\ \leftarrow x^{n-1} - \gamma^{n-1} \end{matrix}.$$

Such lattice $\mathcal{L}(\vec{B})$ forms a sub-lattice of $\mathcal{L}_n = \{\mathbf{q} \in \mathbb{Z}[x] \text{ with } \deg \mathbf{q} \leq n-1\} \cong \mathbb{Z}^n$. We are looking for a polynomial $M(x)$ such that $M(\gamma) = 0 \mod p$ and $\sigma = \|M\|_\infty$ is small, i.e., it is a short vector of $\mathcal{L}(\vec{B})$. In 1896 [85], Minkowski gave a bound on the norm of the shortest vector of a lattice $\mathcal{L}$ for any fixed norm. In particular, in the case of the norm $\| \cdot \|_\infty$ the shortest vector $\mathbf{v}$ satisfies $\|\mathbf{v}\|_\infty \leq |\det \mathcal{L}|^{1/d}$ if $d = \dim \mathcal{L}$. In our case this means that

$$\|\mathbf{v}\|_\infty \leq \left|\det \mathcal{L}(\vec{B})\right|^{1/n} = p^{1/n}.$$

In practice we can use the LLL algorithm [86] to compute a short polynomial $M(x)$, which has $\sigma = \|M\|_\infty$ close to $p^{1/n}$, even if σ is not optimal.

5.1.2 AMNS multiplication with Lagrange approach

We review in this section the modified version of Algorithm 9 based on Lagrange representation of the polynomials. These results are also part of [16]. Our method performs the polynomial multiplication and the reduction modulo E at the same time. In order to use Lagrange representation for polynomial multiplication in Algorithm 9, we select two integers m and m' such that the polynomial $E = (x^n - \lambda)$ splits entirely in $\mathbb{Z}/m\mathbb{Z}[x]$ and $\mathbb{Z}/m'\mathbb{Z}[x]$

$$E = \prod_{i=1}^{n}(x - \alpha_i) \mod m, \quad E = \prod_{i=1}^{n}(x - \alpha'_i) \mod m'.$$

We can then represent the polynomials $\mathbf{u}$ and $\mathbf{v}$ in Algorithm 9 in Lagrange representation relatively to the roots of E modulo m and E modulo m'.

Notation 1. *We will use in the sequel the following notation : for a polynomial* $\mathbf{u}$ *of degree* $n-1$ *we will denote* $\bar{\mathbf{u}}$ *the Lagrange representation in* α_i *modulo* m *and* $\bar{\bar{\mathbf{u}}}$ *the Lagrange representation in* α'_i *modulo* m'.

We modify Algorithm 9 in order to perform the polynomial multiplication modulo E in Lagrange representation. This approach is shown in Algorithm 10.

Algorithm 10 Lagrange-AMNS Multiplication

Require: $\bar{\mathbf{u}}, \bar{\bar{\mathbf{u}}}, \bar{\mathbf{v}}, \bar{\bar{\mathbf{v}}}$ the Lagrange representation modulo m and m' of $\mathbf{u}(x)$ and $\mathbf{v}(x)$, $\bar{\bar{M}}$ the Lagrange representation of the shortest polynomial $M(x)$, $\bar{M'}$ the Lagrange representation of $M' = -M^{-1} \mod (E, m)$.

Ensure: $\bar{\mathbf{r}}, \bar{\bar{\mathbf{r}}}$ such that $\mathbf{r} \in \mathcal{B}$ and $\mathbf{r}(\gamma) = \mathbf{u}(\gamma)\mathbf{v}(\gamma)m^{-1} \bmod p$

1: $\bar{\mathbf{q}} \leftarrow \bar{\mathbf{u}} \times \bar{\mathbf{v}} \times \bar{M'}$
2: $\bar{\bar{\mathbf{q}}} \leftarrow ChangeLR_{m \to m'}(\bar{\mathbf{q}}))$
3: $\bar{\bar{\mathbf{r}}} \leftarrow (\bar{\bar{\mathbf{u}}} \times \bar{\bar{\mathbf{v}}}) + \bar{\bar{\mathbf{q}}} \times \bar{\bar{M}}) \times m^{-1}$
4: $\bar{\mathbf{r}} \leftarrow ChangeLR_{m' \to m}(\bar{\bar{\mathbf{r}}})$

We have to deal with some troubleshooting provided by this strategy. Indeed, at the end of the first step we only know $\bar{\mathbf{q}}$, but we do not know $\bar{\bar{\mathbf{q}}}$ which is required in the modified step 3 of the AMNS multiplication. So we must perform a *change of Lagrange representation* to compute $\bar{\bar{\mathbf{q}}}$ from $\bar{\mathbf{q}}$. Similarly, to get a complete multiplication algorithm, we need to know $\bar{\mathbf{r}}$ at the end of the AMNS multiplication to get the Lagrange representation of $\mathbf{r}$ modulo m and m'.

In [16] we noticed that due to the specific form of $E = x^n - \lambda$ the roots α_i modulo m (resp. α'_i modulo m') of E are of the form

$$\alpha_i = \mu\omega^j \quad (\text{resp. } \alpha'_i = \mu'\omega'^j)$$

where μ and μ' are arbitrary roots of E and ω, ω' are primitive n-th roots of unity modulo m and m', respectively. This implies that the ChangeLR can be performed through two consecutive DFT (or FFT if $n = 2^k$) as given in Algorithm 11.

Algorithm 11 ChangeLR$_{m \to m'}$

Require: $\bar{u}$
Ensure: $\bar{\bar{u}}$
$\widetilde{\mathbf{u}}(x) \leftarrow DFT^{-1}(n, \omega, \bar{\mathbf{u}}) \mod m$
$\mathbf{u}(x) \leftarrow \mathbf{u}(\mu^{-1}x) \mod m$
$\widetilde{\mathbf{u}}(x) \leftarrow \mathbf{u}(\mu' x) \mod m'$
$\bar{\bar{\mathbf{u}}} \leftarrow DFT(n, \omega', \widetilde{\mathbf{u}}(x)) \mod m'$

In [16] we evaluated the cost of an AMNS multiplication in Lagrange representation in terms of the number of additions and multiplications modulo m and m'. For the sake of simplicity, we assumed that that m and m' have the same size (generally m is bigger since $m \geq 2\lambda\rho$ and $m' \geq 2\rho$) and operations modulo m and m' are assumed to have the same cost. In Table 5.3 below we give the cost of each step of the Lagrange AMNS multiplication and the cost of the overall algorithm: we assumed that n is a power of 2 and that FFT is used in the *ChangeLR* routine.

Table 5.3: Complexity of basic operations

Computation	# Multiplications	# Additions
$\bar{U}\bar{V}\bar{M}'$	$2n$	0
$ChangeLR_{m \to m'}(\bar{Q})$	$n\log_2(n) + 2(n-1)$	$2n\log_2(n)$
$(\bar{U}\bar{V} + \bar{\bar{Q}}\bar{M})m_1^{-1}$	$3n$	n
$ChangeLR_{m' \to m}(\bar{Q})$	$n\log_2(n) + 2(n-1)$	$2n\log_2(n)$
Total	$2n\log_2(n) + 9n - 4$	$4n\log_2(n) + n$

Let us briefly compare our scheme with a strategy involving *Montgomery Multiplication using Schönhage-Strassen for integer multiplication*, which seems to be the best strategy comparable to the proposed scheme. Recall that Montgomery algorithm has a cost of 3 integer multiplications of size $\cong p$.

In Schönhage-Strassen [81] in the first recursion, FFT is done modulo an integer $m \cong p^{2/n}$ and correspond to Lagrange approach modulo $E = x^{2n} - 1$. Each integer multiplication requires $3FFT$ (counting only the first recursion) at $2n$ points with a modulo m with size $p^{2/n}$. Consequently, for the overall Montgomery multiplication, we have $9FFT$ in $2n$ points computations with coefficients size $p^{2/n}$ in FFT compared to $4FFT$ in n points with coefficients of size $p^{1/n}$ for the proposed approach in AMNS.

Further study might be pursued since in Schönhage-Strassen [81] the multiplications by the roots of unity have a cost of one addition and this is not the case of the proposed approach.

5.1.3 Multiplication in $\mathbb{F}_{p^k}$ with improved DFT

We focus now on arithmetic over prime field extension $\mathbb{F}_{p^k}$ used in pairing computation: the prime p is not sparse and the degree k is in the range $[6, 30]$. Such field is generally constructed as $\mathbb{F}_p[T]/(f(T))$ where $f(T)$ is a sparse polynomial, i.e., a binomial or a trinomial. Multiplication in the prime field $\mathbb{F}_p$ is usually performed with the Montgomery approach and a multiplication

in the extended field $\mathbb{F}_{p^k}$ consists in a polynomial multiplication computed with Karatsuba and Toom-Cook formulas [87, 88]. The reduction modulo $f(T)$ consists in $O(k)$ additions and multiplications by small constants in $\mathbb{F}_p$.

We present an alternative approach [17] for the implementation in prime field extension $\mathbb{F}_{p^k}$ used in pairing cryptography. This work was done in collaboration with Nadia El Mrabet while she was preparing her PhD thesis in Montpellier. The proposed approach takes advantage of the AMNS for prime field representation.

For the multiplication in $\mathbb{F}_{p^k}$ we use the DFT approach: given $U(T), U(T) \in \mathbb{F}_p[T]$ we compute $\hat{U} = DFT(U)$ and $\hat{V} = DFT(V)$, then multiply terms by terms $\hat{W} = \hat{U} \times \hat{V}$ and then we reconstruct $W = DFT^{-1}(\hat{C})$. In [17] we proposed to focus on fields $\mathbb{F}_p$ with ℓ-th roots of unity such that $\ell \in \{2k-1, 2k-2, 2k-3\}$ and such that the multiplication by these roots of unity are almost free of computation for well chosen AMNS representation of $\mathbb{F}_p$. Specifically, we proposed to consider fields $\mathbb{F}_{p^k}$ satisfying the following definition.

Definition 3 (DFT friendly field)**.** *We call a DFT friendly field an extension field $\mathbb{F}_{p^k}$ such that $\mathbb{F}_p$ admits an AMNS $\mathcal{B} = (p, n, \gamma, \rho)_E$ of length n and such that one of the following conditions holds*

1. *$\lambda = 1$ and $n \in \{2k-1, 2k-2, 2k-4\}$ and γ is a primitive ℓ-th root of unity where $\ell = n$.*
2. *$\lambda = -1$ and $n \in \{k-1, k-2\}$ and γ is a primitive ℓ-th root of unity where $\ell = 2n$.*

The most important property of a DFT friendly field is that the ℓ-th roots of unity are the elements $\pm\gamma^i$. The multiplication by these roots is computed with the formula stated in the following Lemma.

Lemma 2. *Let an AMNS $\mathcal{B} = (p, \ell, \gamma, \rho)_E$ and $\mathbf{u} = \sum_{i=0}^{n-1} u_i x^i$ be expressed in $\mathcal{B}$. The multiplication of $\mathbf{u}(x)$ by the power γ^i of γ is given by*

$$\mathbf{u}\gamma^i = \lambda a_{n-i} + \lambda a_{n-i+1}x + \cdots + \lambda a_{n-1}x^{i-1} + a_0 x^i + \cdots + a_{n-i-1}x^{n-1}$$

In other words a multiplication by γ^i is almost free of computations since $\lambda = \pm 1$. This leads to a simplified DFT computation which essentially consists in $O(\ell^2)$ additions. When ℓ is power of 2, this leads also to a simplified FFT computation which consists in $O(\ell \log_2(\ell))$ additions in $\mathbb{F}_p$.

In practice it is not easy to have ℓ being equal to $2k-1$: we have to relax this constraint to $\ell = 2k-2$ and $\ell = 2k-3$ and slightly adapt polynomial multiplication based on DFT to this special case. In [17] we discussed how to compute some missing coefficients output by the DFT when $\ell < 2k-1$. For example, we provided Lemma 3 for $\ell = 2k-2$.

Lemma 3. *Let $\mathbb{F}_p$ be a prime field, ω be a primitive ℓ-th root of unity and Ω and Ω^{-1} be the matrices for DFT_ω and DFT_ω^{-1} computation. We consider $U = \sum_{i=0}^{k-1} \mathbf{u}_i T^i$, $V = \sum_{i=0}^{k-1} \mathbf{v}_i T^i$ and $W = U \times V$ and we assume that $\ell = 2k-2$. Then W can be computed as follows.*

1. *$\hat{U} = DFT_\omega(U), \hat{V} = DFT_\omega(V)$*
2. *$\mathbf{w}_{2k-2} = \mathbf{u}_{k-1} \times \mathbf{v}_{k-1}$*
3. *The coefficients $\mathbf{w}_i$ for $i = 0, \ldots, 2k-3$ are computed as*

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{2k-3} \end{bmatrix} = \Omega^{-1} \cdot \begin{bmatrix} \hat{\mathbf{v}}_0 \times \hat{\mathbf{v}}_0 - \mathbf{w}_{2k-2} \\ \hat{\mathbf{u}}_1 \times \hat{\mathbf{v}}_1 - \mathbf{w}_{2k-2} \\ \vdots \\ \hat{\mathbf{v}}_{2k-3} \times \hat{\mathbf{v}}_{2k-3} - \mathbf{w}_{2k-2} \end{bmatrix}. \tag{5.6}$$

We present in Table 5.4 the complexities for a multiplication in DFT friendly fields $\mathbb{F}_{p^k}$ for the sizes of k used in pairing cryptography. For comparison purpose, we also provide in Table 5.4 the complexity of the multiplication when using Karatsuba and Toom Cook formulas [88].

Table 5.4: Complexity comparison for practical extension degree k

Method	k	Cost of $Mult_{F_{p^k}}$	
		# Add. in $\mathbb{F}_p$	# Mult. in $\mathbb{F}_p$
Karatsuba/Toom-Cook [87, 88]	6	60	15
Karatsuba/Toom-Cook [87, 88]	8	72	27
Proposed with FFT and $E = t^8 + 1$	8	192	16
Karatsuba/Toom-Cook [87, 88]	9	160	25
Proposed with FFT and $E = t^8 + 1$	9	208	18
Proposed with FFT and $E = t^8 + 1$	10	240	23
Proposed with $E = \sum_{i=0}^{10}(-t)^i$	11	902	22
Karatsuba/Toom-Cook [87, 88]	12	180	45
Proposed with $E = \sum_{i=0}^{10}(-t)^i$	12	1408	24
Proposed with $E = \sum_{i=0}^{10}(-t)^i$	13	1430	28
Karatsuba/Toom-Cook [87, 88]	16	248	81
Proposed with FFT and $E = t^{16} + 1$	16	480	32
Proposed with FFT and $E = t^{16} + 1$	17	512	34
Proposed with FFT and $E = t^{16} + 1$	18	576	39
Karatsuba/Toom-Cook [87, 88]	24	588	135

We remark that even for small values of k, DFT approach seems competitive regarding to the number of multiplications. When no FFT can be used, the number of additions increases significantly and might render inefficient the proposed approach in these special cases.

5.2 Trade-off approach in leak resistant arithmetic in residue number system

We present in this section a work done with Guilherme Perin [19] on randomized modular arithmetic in residue number system. The residue number system (RNS) is a system which uses t coprime integers $a_1, \ldots, a_t$. An integer u such that $0 \leq u < A = \prod_{i=1}^{t} a_i$ is represented in RNS by the t residues

$$u_i = u \mod a_i \text{ for } i = 1, \ldots, t.$$

Moreover, u can be recovered from its RNS expression using the Chinese remainder theorem (CRT) as follows

$$u = \left(\sum_{i=1}^{t} \left[u_i \times A_i^{-1} \right]_{a_i} \times A_i \right) \mod A \tag{5.7}$$

where $A_i = \prod_{j=1, j \neq i}^{t} a_i$ and the brackets $[\ \cdot\ ]_{a_i}$ denote a reduction modulo a_i. The set $\mathcal{A} = \{a_1, \ldots, a_t\}$ is generally called an RNS basis.

Let $u = (u_1, \ldots, u_t)_{\mathcal{A}}$ and $v = (v_1, \ldots, v_t)_{\mathcal{A}}$ be two integers given in an RNS basis $\mathcal{A}$. Then, the CRT provides that an integer addition $u + v$ or multiplication $u \times v \mod A$ in RNS consists

in t independent additions/multiplications modulo a_i

$$\begin{aligned} u+v &= ([u_1+v_1]_{a_1},\ldots,[u_t+v_t]_{a_t}),\\ u\times v &= ([u_1\times v_1]_{a_1},\ldots,[u_t\times v_t]_{a_t}). \end{aligned}$$

The main advantage is that these operations can be implemented in parallel since each operation modulo a_i are independent from the others. Only comparisons and Euclidean divisions are not easy to perform in RNS and require partial reconstruction of the integers u and v.

Montgomery multiplication in RNS. In [55] Posch and Posch noticed that the Montgomery multiplication can be efficiently implemented in RNS: they use the fact that we can modify the Montgomery multiplication (Algorithm 7) as

$$\begin{aligned} q &\leftarrow -u\times v\times p^{-1} \mod A\\ w &\leftarrow (uv+qp)A^{-1} \mod B \end{aligned}$$

where B is an integer coprime with A and p and greater than $2p$. Furthermore, Posch and Posch propose to perform this modified version of the Montgomery multiplication in RNS. Specifically, they choose two RNS bases $\mathcal{A}=\{a_1,\ldots,a_t\}$ and $\mathcal{B}=\{b_1,\ldots,b_t\}$ such that $\gcd(a_i,b_j)=1$ for all i,j. They compute $w=uvA^{-1} \mod p$ as it is shown in Algorithm 12: the multiplications modulo A are done in the RNS basis $\mathcal{A}$ and the operations modulo B are done in $\mathcal{B}$.

Algorithm 12 RNS-MontMul$(u,v,\mathcal{A},\mathcal{B})$

Require: u,v in $\mathcal{A}\cup\mathcal{B}$
Ensure: $uvA^{-1} \mod p$ in $\mathcal{A}\cup\mathcal{B}$
1: $[q]_{\mathcal{A}}\leftarrow[-uvp^{-1}]_{\mathcal{A}}$
2: $BE_{\mathcal{A}\to\mathcal{B}}([q]_{\mathcal{A}})$
3: $[w]_{\mathcal{B}}\leftarrow[(uv+qp)A^{-1}]_{\mathcal{B}}$
4: $BE_{\mathcal{B}\to\mathcal{A}}([w]_{\mathcal{B}})$
5: **return** $(w_{\mathcal{A}\cup\mathcal{B}})$

The second and fourth steps are necessary since if we want to compute $w\leftarrow(uv+qp)A^{-1} \mod B$ in $\mathcal{B}$ we need to convert the RNS representation of q from the basis $\mathcal{A}$ to the basis $\mathcal{B}$: the base extension (BE) performs this conversion. The fourth step is also necessary to have z represented in both bases $\mathcal{A}$ and $\mathcal{B}$.

Leak resistant arithmetic in RNS. The authors in [57] notice that the use of RNS facilitates the randomization of the representation of an integer and consequently the randomization of a modular multiplication. Indeed, if a modular exponentiation $u^e \mod p$ is computed using the RNS-MontMul algorithm the element u is first set in Montgomery representation

$$\widetilde{u}=u\times A \mod N$$

and in the RNS bases $\mathcal{A}$ and $\mathcal{B}$, i.e., $[\widetilde{u}]_{\mathcal{A}\cup\mathcal{B}}$. The Montgomery representation induces a multiplicative masking of the data u by the factor A. The authors in [57] propose to randomly construct the basis $\mathcal{A}$ to get a random multiplicative mask A on the data.

Specifically, the authors in [57] propose two levels of such randomization: random initialization of the bases $\mathcal{A}$ and $\mathcal{B}$ at the very beginning of a modular exponentiation and random permutations of the RNS bases $\mathcal{A}$ and $\mathcal{B}$ all along the modular exponentiation. They also provide a method to perform the conversion of $\widetilde{u}$ into the new set of RNS base $\mathcal{A}$ and $\mathcal{B}$ at a cost of two Montgomery multiplications in RNS each time $\mathcal{A}$ and $\mathcal{B}$ are modified.

Proposed trade-off randomization approach for leak resistant arithmetic in RNS. We present the strategy of [19] elaborated with G. Perin which modifies only one modulus in $\mathcal{A}$ while keeping $\mathcal{B}$ unchanged during each update of the RNS bases. This requires an additional set $\mathcal{A}'$ of spare moduli where we randomly pick the new modulus for $\mathcal{A}$. We have three sets of moduli:

- The first RNS basis $\mathcal{A} = \{a_1, \ldots, a_{t+1}\}$ which is modified after each loop iteration.
- The set $\mathcal{A}' = \{a'_1, \ldots, a'_{t+1}\}$ of spare moduli.
- The second RNS basis $\mathcal{B} = \{b_1, \ldots, b_{t+1}\}$ which is fixed at the beginning of the exponentiation.

The integers a_i, b_i and a'_i are all pairwise co-prime. The proposed update of the RNS bases and RNS representation of u are the following:

- *Update of the basis $\mathcal{A}$.* Updating the basis $\mathcal{A}$ is quite simple: we just swap one element of $\mathcal{A}$ with one element of $\mathcal{A}'$ as follows

$$\begin{array}{l} r \leftarrow \text{ random in } \{1, \ldots, t, t+1\}, \\ r' \leftarrow \text{ random in } \{1, \ldots, t, t+1\}, \\ a_{r,new} \leftarrow a'_{r',old}, \\ a_{r',new} \leftarrow a_{r,old}. \end{array}$$

  In the sequel we will denote $\mathcal{A}_{old}$ and $\mathcal{A}_{new}$ the base $\mathcal{A}$ before and after the update, we will use similar notation for other updated data.

- *Update of the Montgomery-RNS representation.* The modification of the basis requires at the same time the corresponding update of the Montgomery representation of u. Indeed we need to compute $\widetilde{u}_{new} = [(uA_{new} \mod p)]_{\mathcal{A}_{new} \cup \mathcal{B}}$ from its old Montgomery representation $\widetilde{u}_{old} = [(uA_{old} \mod p)]_{\mathcal{A}_{old} \cup \mathcal{B}}$. In [19] we showed that this update of $\widetilde{u}$ can be computed as follows:

$$\begin{array}{rcl} \lambda & = & [-\widetilde{u}_{a_{r,old}} \times p^{-1}]_{a_{r,old}}, \\ \widetilde{u}_{new} & = & [(\widetilde{u}_{old} + \lambda \times p) \times a_{r,old}^{-1} \times a_{r,new}]_{\mathcal{A}_{new} \cup \mathcal{B}}. \end{array} \tag{5.8}$$

In [19] we presented a variant of the above randomization which avoids the use of the set of spare moduli $\mathcal{A}'$: the modified modulus in $\mathcal{A}$ is randomly picked in $\mathcal{B}$. The complexities of the update of the RNS bases $\mathcal{A}, \mathcal{B}$ and the update of the Montgomery representation are sightly larger compared to the first approach but the memory requirement is reduced and the number of required moduli is also reduced.

In Table 5.5 we report the complexity of the randomization by either changing only one modulus in the basis $\mathcal{A}$ or by modifying s moduli in $\mathcal{A}$ at each loop turn.

These complexities show that we get a cheaper randomization by changing only one modulus, at a cost of a lower level of randomization. When we increase this level of randomization by changing more than one modulus at each loop turn, we obtain a trade-off between randomization and complexity. For the average randomization of $s = t/2$ moduli changed per loop turn of the algorithm for exponentiation, our method requires $6t^2 + O(t)$ multiplications and $2t^2 + 3t$ additions: this is better than the complexity of [57]. Another advantage of our technique is that it works in the cox-rower architecture [56] which is the most popular architecture for RNS implementation.

Table 5.5: Cost of the randomization in one loop iteration of randomized Montgomery-ladder

Randomization	Method	# Mul.	# Add.	Memory
1 modulus per loop	Proposed with a set of spare moduli	$18t + 10$	$4t + 2$	$8t^2 + 24t + 18$
1 modulus per loop	Proposed without a set of spare moduli	$24t + 26$	$4t + 4$	$8t^2 + 19t + 20$
s moduli per loop	$s\times$ proposed with a set of spare moduli	$s(12t + 20) + 6t + 8$	$s(4t + 6)$	$8t^2 + 42t + 52$
$t/2$ moduli per loop (in average)	LRA of Bajard *et al.* [57]	$8t^2 + 32t$	$8t^2 + 12t - 4$	$8t^2$

5.3 Conclusion

In this chapter, we considered arithmetic in prime field $\mathbb{F}_p$ and prime field extension $\mathbb{F}_{p^k}$. In the non-conventional AMNS representation of elements of $\mathbb{F}_p$, we proposed a Montgomery like approach for multiplication modulo p when p has no special structure. We also adapted this approach in order to use a multi-evaluation strategy for modular multiplication. We then considered polynomial multiplication in $\mathbb{F}_p[T]$ using DFT: we showed that using an AMNS with a generator which is a root of unity leads to a faster DFT. This idea was later implemented by N. El Mrabet and N. Gamma in [89] which showed that, when FFT can be used, the proposed approach is competitive compared to conventional approach.

We considered at the end of this chapter modular arithmetic in Residue Number System. Specifically, we looked at the leak resistant strategy for randomizing arithmetic and data encoding presented in [57]. We investigated a trade-off variant of the approach of [57] which reduces the level of randomization but has a smaller complexity.

Chapter 6

Parallel scalar multiplication over $E(\mathbb{F}_{2^n})$ and $E(\mathbb{F}_{3^n})$

In this chapter we consider scalar multiplication over an elliptic curve: given a point P on the curve, the goal is to compute

$$kP = \underbrace{P + \cdots + P}_{k \text{ times}}$$

using the group law given by the chord and tangent rules. Specifically, we focus on parallel scalar multiplication of elliptic curve defined over $\mathbb{F}_{2^n}$ and $\mathbb{F}_{3^n}$. Recent processors include a number of cores, it is thus interesting to adapt the implementation of a scalar multiplication in order to execute it in parallel on these multiple cores. Usual algorithms for scalar multiplication, i.e., variants of the double-and-add approach, cannot be easily parallelized: they involve a long chain of dependent doublings and additions. In the literature only two approaches makes it possible to break this chain of dependent operations:

- *Parallel approach based on GVL [90].* The GLV approach [90] uses an endomorphism ϕ to rewrite $kP = k_1P + k_2\phi(P)$ where k_1 and k_2 have a bit length $\cong \log_2(k)/2$. If $\phi(P)$ can be easily computed, the two scalar multiplications k_1P and $k_2\phi(P)$ can be concurrently evaluated and then added.
- *(Double,halve)-and-add parallelization.* This approach rewrites

$$k = \underbrace{\sum_{i=0}^{\ell} k'_i 2^i}_{k'} + \underbrace{\sum_{i=0}^{\ell} k''_i 2^{-i}}_{k''}$$

  and concurrently performs $k'P$ through a double-and-add scalar multiplication, and $k''P$ through a halve-and-add scalar multiplication and then add them to get kP. This approach is effective only on $\mathbb{F}_{2^n}$ [91, 92, 93].

In the remainder of this chapter we present two approaches extending the parallel (double,halve)-and-add parallelization. These results [6] were done in collaboration with Jean-Marc Robert while he was preparing his PhD thesis in the team DALI (2012-2015). In Section 6.1 we present a parallel version of the Montgomery-ladder extending the (double,halve)-and-add parallelization but ensuring security against simple side channel attacks (SPA [44], SEMA [45] and timing attacks [43]). In Section 6.2 we present a thirding operation over $E(\mathbb{F}_{3^n})$ and we use it in a parallel (triple,third)-and-add scalar multiplication.

6.1 Parallel Montgomery-ladder over binary elliptic curves

In this section, we consider an extended binary field $\mathbb{F}_{2^n} = \mathbb{F}_2[t]/(f(t))$, where $f(t) \in \mathbb{F}_2[t]$ is an irreducible polynomial of degree n, and an elliptic curve $E(\mathbb{F}_{2^n})$ defined by the following Weierstrass equation:

$$E : y^2 + xy = x^3 + ax^2 + b \text{ with } a, b \in \mathbb{F}_{2^n}. \tag{6.1}$$

There is on such curve an underlying group law provided by the chord and tangent rules. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_{2^n})$, this group law can be expressed as an algebraic expression in terms of the coordinates of the points. Specifically, addition, doubling and halving on $E(\mathbb{F}_{2^n})$ are performed as follows:

- *Addition.* The coordinates (x_3, y_3) of $P_3 = P_1 + P_2$ are computed as follows:

$$\left\{ \begin{array}{rcl} x_3 & = & \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 & = & (x_1 + x_3)\lambda + x_3 + y_1, \end{array} \right. \text{ where } \lambda = \left\{ \begin{array}{ll} \frac{y_1+y_2}{x_1+x_2} & \text{if } P_1 \neq P_2, \\ \frac{y_1}{x_1} + x_1 & \text{if } P_1 = P_2. \end{array} \right. \tag{6.2}$$

- *Doubling.* The coordinates (u, v) of $P = 2 \cdot Q$ are expressed in terms of the coordinates of $Q = (x, y)$ as follows

$$\lambda = x + y/x, \tag{6.3}$$
$$u = \lambda^2 + \lambda + a, \tag{6.4}$$
$$v = x^2 + u(\lambda + 1). \tag{6.5}$$

- *Halving.* The halving formula is derived from the doubling formula: let $P = (x, y)$ and $Q = (u, v)$ be two points on $E(\mathbb{F}_{2^n})$ of odd order such that $Q = 2 \cdot P$ which gives $P = \frac{1}{2}Q$. Following [91, 92] we recover the coordinates (x, y) in terms of u and v as follows:

  1. We first solve the quadratic equation (6.4) to get λ, with a half-trace (HT) computation
  $$HT(u + x) = \sum_{i=0}^{\frac{n-1}{2}} (u + x)^{2^{2i}}.$$
  2. Then using (6.5) we compute $x = \sqrt{v + u(\lambda + 1)}$
  3. Finally, we compute y with (6.3) as $y = (\lambda + x)x$.

The most widely used method to implement scalar multiplication is the classical *double-and-add* approach which computes $k \cdot P$ as a sequence of doubling followed by an addition if the scanned bit k_i is equal to 1. Optimized variants of this approach rewrite the scalar k with non-adjacent form (NAF or NAF$_w$) in order to reduce the number of additions.

Knudsen in [91] proposed to perform the scalar multiplication as a sequence of halvings and additions to take advantage of the efficiency of the halving on $E(\mathbb{F}_{2^n})$. Moreover, as mentioned in [94, 93], double-and-add and halve-and-add approaches can be combined in order to parallelize the scalar multiplication in a (double,halve)-and-add approach.

All these previously mentioned approaches are sensitive to side channel attack like timing attack and simple power analysis. Indeed these attacks analyze either the computation time or the power consumption to recover the sequence of additions and doublings performed during the computation of $k \cdot P$. This leaks some information of the coefficients of k since when an addition is performed this means that $k_i \neq 0$ otherwise $k_i = 0$.

It is thus recommended to implement scalar multiplication with an algorithm which is regular and has a constant time. One popular algorithm which achieves this goal is the Montgomery-ladder: this approach compute the scalar multiplication through a regular sequence of doublings always followed by an addition as shown in Algorithm 13.

Algorithm 13 Montgomery-ladder scalar multiplication

Require: $P \in E(\mathbb{F}_{2^n})$ and a scalar $k \in [0, 2^\ell[$
Ensure: $Q = k \cdot P$
1: $Q_0 \leftarrow \mathcal{O}$
2: $Q_1 \leftarrow P$
3: **for** i from $\ell - 1$ downto 0 **do**
4: **if** $(k_i = 0)$ **then**
5: $T \leftarrow Q_0, Q_0 \leftarrow 2T, Q_1 \leftarrow T + Q_1$
6: **else**
7: $T \leftarrow Q_1, Q_1 \leftarrow 2T, Q_0 \leftarrow Q_0 + T,$
8: **end if**
9: **end for**
10: **return** Q_0

In [6], we proposed a parallelized variant of the Montgomery ladder. To reach this goal we first proposed a Montgomery-halving scalar multiplication which performs a regular sequence of halvings and additions to compute $k \cdot P$. This approach is depicted in Algorithm 14.

Algorithm 14 Montgomery-halving

Require: $P \in E(\mathbb{F}_{2^n})$ of odd order N and a scalar $k \in [0, N-1]$
Ensure: $Q = k \cdot P$.
1: Compute $k' = 2^{\ell-1} \cdot k \mod N = \sum_{i=0}^{\ell-1} k'_i 2^i$ with $\ell = \lfloor \log_2(N) \rfloor + 1$
2: $Q_0 \leftarrow k'_0 \cdot P, Q_1 \leftarrow k'_0 \cdot P - 2 \cdot P$
3: **for** i from 1 to $\ell - 1$ **do**
4: **if** $(k'_i = 1)$ **then**
5: $T \leftarrow Q_0/2, Q_0 \leftarrow T, Q_1 \leftarrow Q_1 - T$
6: **else**
7: $T \leftarrow Q_1/2, Q_1 \leftarrow T, Q_0 \leftarrow Q_0 - T$
8: **end if**
9: **end for**
10: **return** (Q_0)

The method to parallelize the computations is similar to the technique used in the (double,halve)-and-add approach. The scalar k is recoded as follows:

$$k = k' \times 2^{-s} \mod N = \underbrace{(k'_{\ell-1} 2^{\ell-1-s} + \ldots + k'_s)}_{k_1} + \underbrace{(k'_{s-1} 2^{-1} + \ldots + k'_0 2^{-s})}_{k_2} \mod N$$

where N is the order of P and is of bit length ℓ. Then the scalar multiplication $k_1 \cdot P$ is computed with the original Montgomery-ladder and the second part $k_2 \cdot P$ is performed in parallel through a Montgomery-halving approach. The final result is obtained with a final addition $k \cdot P = (k_1 \cdot P) + (k_2 \cdot P)$.

Implementation results. We have implemented the proposed approach on a DELL Optiplex 990 equipped with an Intel Core i7-2600 processor and a Nexus 7 tablet with a Qualcomm

Snapdragon processor (ARM-v7 architecture). On the Intel Core i7 processor we implemented the underlying finite field operations using mostly the strategies used in the work of Taverne *et al.* [94, 93]. Specifically, multiplications were computed by combining a few Karatsuba recursions and PCLMUL instruction and squarings and square roots were implemented with a few PSHUFB instructions. The half-trace was implemented with a sequence of look-up table.

For the implementation over the Qualcomm Snapdragon we used the strategy of [95]. We used the algorithm of Lopez-Dahab [96] for polynomial multiplication which consists of a sequence of shifts, bitwise XORs and look-up table. Half-trace, squaring and square-root were implemented with a sequence of look-up table.

The timings obtained are reported in Table 6.1. We provide timings for Mongomery-ladder, Mongomery-halving and its parallel version.

Table 6.1: Timings of scalar multiplication on an Intel Core i7 and a Qualcomm SnapDragon processors

Algorithm	#Core	Intel Core i7				Qualcomm SnapDragon			
		B233		B409		B233		B409	
		$\#CC/10^3$	split	$\#CC/10^3$	split	$\#CC/10^3$	split	$\#CC/10^3$	split
Montgomery-Ladder	1	149	-	694	-	3407	-	16311	-
Montgomery-Halving	1	689	-	3826	-	11337	-	59810	-
Montgomery-Parallel	2	143	39	624	59	2756	56	13155	86
Relative speed-up Mont-Par./Mont-Ladder		5.09 %		10.5 %		19 %		19 %	

One drawback of the proposed approach is that the Montgomery-halving algorithm have to be computed in affine coordinates which involve costly field inversions. We can notice that our proposed parallelized version of the Montgomery-ladder provides a speed-up of 5% to 10% compared to the Montgomery-ladder approach on an Intel Core i7 processor and a speed-up of 19% on a Snapdragon processor. The parallelization on the Snapdragon processor is better, which can be partly explained by the fact that the ratio I/M is smaller over the Snapdragon processor.

6.2 Parallel Montgomery-ladder over $E(\mathbb{F}_{3^n})$

In this section, we present the results obtained in [6] for parallel scalar multiplication over $E(\mathbb{F}_{3^n})$. We focus on elliptic curves defined over characteristic three fields $\mathbb{F}_{3^n}$. Such elliptic curves are generally separated into two kinds of curves: supersingular elliptic curves and ordinary elliptic curves. Here we will focus on ordinary elliptic curves. Such curves can be defined by a Weierstrass equation:

$$y^2 = x^3 + ax^2 + b$$

where $a, b \in \mathbb{F}_{3^n}$ and $a, b \neq 0$ since the curve equation must be non-singular. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_{3^n})$. The addition, doubling and tripling formulas on this curve are as follows:

Addition and doubling. The coordinates (x_3, y_3) of the point $P_3 = P_1 + P_2$ are:

$$\begin{cases} x_3 &= \lambda^2 - a - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{cases} \text{ where } \begin{cases} \lambda &= \frac{y_2 - y_1}{x_2 - x_1} \text{ if } P_1 \neq P_2, \\ \lambda &= \frac{ax_1}{y_1} \text{ if } P_1 = P_2. \end{cases}$$

Tripling. The coordinates (x_3, y_3) of $P_3 = 3 \cdot P_1$ are computed as follows:

$$\begin{cases} x_3 &= \frac{y_1^6}{a^2(x_1^3+b)^2} - \frac{ax_1^3}{x_1^3+b}, \\ y_3 &= \frac{y_1^9}{a^3(x_1^3+b)^3} - \frac{y_1^3}{x_1^3+b}. \end{cases} \tag{6.6}$$

A common strategy to improve the efficiency of the addition, doubling and tripling operations consists in using a projective coordinate system in order to remove field inversions which are costly operations. The following set of projective coordinates were proposed in the context of curves defined over field of characteristic three: Jacobian projective coordinates where (X, Y, Z) corresponds to the affine points $(X/Z^2, Y/Z^3)$, ML-projective coordinates [97] where (X, Y, Z, T) corresponds to $(X/Z^2, Y/T)$ and the scaled projective coordinates [98] with (X, Y, T) corresponding to $(X/bT, Y/bT)$.

In [6] we extended the halve-and-add scalar multiplication to the context of $E(\mathbb{F}_{3^n})$. We get thirding formulas in similar fashion as the halving was obtained on $E(\mathbb{F}_{2^n})$. This thirding formula leads to a third-and-add scalar multiplication.

6.2.1 Thirding formula

There are two types of ordinary curves $E(\mathbb{F}_{3^n})$: one having $a = 1$ and one having $a = -1$. We focus only on the method used to get a thirding formula for $a = 1$. The case $a = -1$ is treated in a similar manner in [6]. We consider two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ on an elliptic curve $E(\mathbb{F}_{3^n})$ given by an equation $y^2 = x^3 + x^2 + b$, i.e., with $a = 1$. The tripling formula in affine coordinates is as follows: if $P = 3 \cdot Q$, the expression of (x_P, y_P) in terms of (x_Q, y_Q) is

$$x_P = \frac{y_Q^6}{(x_Q^3+b)^2} - \frac{x_Q^3}{x_Q^3+b}, \qquad y_P = \frac{y_Q^9}{(x_Q^3+b)^3} - \frac{y_Q^3}{x_Q^3+b}. \tag{6.7}$$

If we set

$$B = \frac{y_Q^3}{x_Q^3 + b} \text{ and } A = \frac{x_Q^3}{x_Q^3 + b}, \tag{6.8}$$

the previous equation (6.7) rewrites as

$$x_P = B^2 - A, \qquad y_P = B^3 - B. \tag{6.9}$$

Computing the thirding of P, i.e., $Q = \left[\frac{1}{3}\right] \cdot P$, consists of the computation of the coordinates x_Q and y_Q in terms of x_P and y_P. We first have to find A and B from (6.9), then deduce x_Q and y_Q from (6.8). We stated in [6] some results concerning the solutions of the equation in the variable B of the form $B^3 - B = u$ where u is a fixed element of $\mathbb{F}_{3^n}$ satisfying $Trace(u) = 0$.

Lemma 4 (ThirdTrace)**.** *We assume that the field $\mathbb{F}_{3^n}$ has a degree satisfying $n \neq 0 \mod 3$. The three solutions of the equation $B^3 - B = u$ where $u \in \mathbb{F}_{3^n}$ satisfies $Trace(u) = 0$ are:*

$$B_0 = \begin{cases} -\sum_{i=0}^{(n-1)/3-1} (u^3 - u)^{3^{3i+1}} & \text{if } n \equiv 1 \mod 3, \\ \sum_{i=0}^{(n-2)/3} (u^3 - u)^{3^{3i}} & \text{if } n \equiv 2 \mod 3, \end{cases} \tag{6.10}$$

and $B_1 = B_0 + 1$ and $B_2 = B_0 + 2$.

In the sequel, for n an odd integer, we will call the third-trace of $B \in \mathbb{F}_{3^n}$ the element

$$ThirdTrace(B) = \begin{cases} -\sum_{i=0}^{(n-1)/3-1} B^{3^i} & \text{if } n \equiv 1 \mod 3, \\ \sum_{i=0}^{(n-2)/3} B^{3^i} & \text{if } n \equiv 2 \mod 3. \end{cases}$$

From the previous lemma, we know how to compute B in terms of y_P from (6.9). We then could extend in [6] this result to get a full thirding formula as stated in Lemma 5.

Lemma 5. *We consider an elliptic curve $E(\mathbb{F}_{3^n})$ defined by $y^2 = x^3 + x^2 + b$ over $\mathbb{F}_{3^n}$ such that $n \not\equiv 0 \mod 3$. We assume that the order N of $E(\mathbb{F}_{3^n})$ is such that $N = 3N'$ and $N' \not\equiv 0 \mod 3$. Then the two following assertions hold:*

i) Let $P = (x_P, y_P) \in E(\mathbb{F}_{3^n})$ satisfying $Trace(y_P) = 0$ then there are three points $Q_0 = (x_0, y_0), Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ on the curve such that $3 \cdot Q_i = P$. Their coordinates can be computed as follows:

$$\boxed{\begin{array}{l} B_0 \leftarrow ThirdTrace(y_P^3 - y_P), \quad B_1 \leftarrow B_0 + 1, \quad B_2 \leftarrow B_0 + 2. \\ A_0 \leftarrow B_0^2 - x_P, \quad A_1 \leftarrow A_0 + 2B_0 + 1, \quad A_2 \leftarrow A_0 + 4B_0 + 4. \\ x_0 \leftarrow \sqrt[3]{\frac{bA_0}{1-A_0}}, \quad x_1 \leftarrow \sqrt[3]{\frac{bA_1}{1-A_1}}, \quad x_2 \leftarrow \sqrt[3]{\frac{bA_2}{1-A_2}}. \\ y_0 \leftarrow \sqrt[3]{\frac{bB_0}{1-A_0}}, \quad y_1 \leftarrow \sqrt[3]{\frac{bB_1}{1-A_1}}, \quad y_2 \leftarrow \sqrt[3]{\frac{bB_2}{1-A_2}}. \end{array}} \tag{6.11}$$

ii) If $P = (x_P, y_P) \in E(\mathbb{F}_{3^n})$ satisfies $Trace(y_P) = 0$ it has an order N', and if $P = (x_P, y_P) \in E(\mathbb{F}_{3^n})$ satisfies $Trace(y_P) \neq 0$ then it has an order equal to $3N'$.

6.2.2 Parallel scalar multiplication in $E(\mathbb{F}_{3^n})$ and implementation results

We consider a curve $E(\mathbb{F}_{3^n})$ given by $y^2 = x^3 + ax^2 + b$ where $a \in \{1, -1\}$. The thirding formulas presented in [6] provide some new approaches for implementing a scalar multiplication on $E(\mathbb{F}_{3^n})$. Indeed, let k be a scalar and let P be a point on $E(\mathbb{F}_{3^n})$ of order $N < 3^\ell$. If we denote $k' = 3^{\ell-1} \times k \mod N$ and if we write $k' = \sum_{i=0}^{\ell-1} k'_i 3^i$ in base 3, i.e., with $k'_i \in \{0, 1, 2\}$ we obtain the following:

$$\begin{aligned} k &= 3^{-(\ell-1)} k' \mod N \\ &= \textstyle\sum_{i=0}^{\ell-1} k'_i 3^{i-\ell+1} \mod N \\ &= \textstyle\sum_{i=0}^{-(\ell-1)} k'_{i+\ell-1} 3^i \mod N. \end{aligned}$$

Consequently, the scalar multiplication $k \cdot P$ can be performed through a sequence of thirdings and additions:

$$k \cdot P = \sum_{i=0}^{-(\ell-1)} k'_{i+\ell-1} 3^i \cdot P.$$

In [6] we extended this idea to a third-and-add approach which uses a $SWR_{3,w}$ recoding of k' which is the equivalent to the NAF_w recoding in base 3 (cf. [23] for details). This approach is depicted in Algorithm 15.

The third-and-add approach is not interesting in practice since the thirding operation appears to be a bit costly since it involves a field inversion. But we can take advantage of the third-and-add approach to implement scalar multiplication in parallel fashion. Indeed, we can split the integer k into two parts k_1 and k_2 as follows: we set a split value $0 < s < \ell = \lceil \log_3(N) \rceil$ and we compute $k' = k \cdot 3^s \mod N$. Then if we write $k' = \sum_{i=0}^{\ell} k'_i 3^i$ in base 3 we can rewrite k as follows:

$$\begin{aligned} k &= 3^{-s} k' \mod N \\ &= \left(\textstyle\sum_{i=0}^{s-1} k'_i 3^{i-s}\right) + \left(\textstyle\sum_{i=s}^{\ell} k'_i 3^{i-s}\right) \mod N \\ &= \underbrace{\left(\sum_{i=1}^{s} k'_{s-i} 3^{-i}\right)}_{k_1} + \underbrace{\left(\sum_{i=0}^{\ell-s} k'_{i+s} 3^i\right)}_{k_2} \mod N. \end{aligned}$$

Algorithm 15 Third-and-add with $SWR_{3,w}$

Require: A curve $E(\mathbb{F}_{3^n})$, a point $P \in E(\mathbb{F}_{3^m})$ of prime order $N < 3^\ell$ and a scalar $k \in [1, N]$.
Ensure: $Q = k \cdot P$
$k' \leftarrow k \cdot 3^\ell \mod N$
$(k'_\ell, \ldots, k'_0) \leftarrow SWR_{3,w}(k')$.
Precomputations. **for** $i = 0, \ldots, \frac{3^w-1}{2}$ and $i \not\equiv 0 \mod 3$ **do** $T[i] \leftarrow iP$.
$Q \leftarrow \mathcal{O}$
for $i = \ell$ **to** 0 **do**
$Q \leftarrow \left[\frac{1}{3}\right] \cdot Q$.
$Q \leftarrow Q + sign(k_i)T[\,|k_i|\,]$
end for

The computation of a scalar multiplication can be split into two concurrent algorithms: a triple-and-add or a double-and-add algorithm which performs $k_2 \cdot P$ and a third-and-add algorithm which performs $k_1 \cdot P$. The result is obtained after a final addition $k \cdot P = k_1 \cdot P + k_2 \cdot P$.

Implementation results. The platforms used for our experimentations are the same as in Subsection 6.1: an Intel Core i7-2400 processor with Ubuntu 12.04 and gcc 4.6.3 and a Qualcomm Snapdragon processor with Ubuntu touch 14.04 and gcc 4.8.2. The two fields considered in our implementations are $\mathbb{F}_{3^{127}} = \mathbb{F}_3[t]/(f(t))$ with $f(t) = (t^{128} + t^{77} + 1)/(t - 1)$ and $\mathbb{F}_{3^{251}} = \mathbb{F}_3[t]/(t^{251} + t^{26} - 1)$. For the implementation of the finite field operations we followed the strategy presented in [99]: logical bit-wise operation for addition, shift-and-add approach for multiplication, look-up table for cubing and cube-root. We implemented the double-and-add, triple-and-add and third-and-add algorithms for scalar multiplication along with their parallel counterparts, in $E(\mathbb{F}_{3^{127}})$ and $E(\mathbb{F}_{3^{251}})$. The curves chosen have either $a = 1$ and order $3N$ where N is prime, either $a = -1$ and a prime order N.

Table 6.2: Timings of scalar multiplication in $E(\mathbb{F}_{3^{127}})$ and $E(\mathbb{F}_{3^{251}})$

Curve type	Algorithm	NAF Wind. size	Nb. of core	Intel Core i7				Qualcomm Snapdragon			
				$n = 127$		$n = 251$		$n = 127$		$n = 251$	
				#CC/10^3	split	#CC/10^3	split	#CC/10^3	split	#CC/10^3	split
$a = 1$	DnA	4	1	615	-	3478	-	3218	-	24238	-
	TnA	2	1	696	-	4128	-	3492	-	27122	-
	TnA	3	1	697	-	3876	-	3479	-	25558	-
	ThnA	2	1	2774	-	12315	-	15427	-	118092	-
	ThnA	3	1	2782	-	12326	-	15679	-	117640	-
	Parallel (TnA,ThnA)	(2,2)	2	615	24	3735	34	3131	22	22697	44
	Parallel (TnA,ThnA)	(3,3)	2	631	24	3588	32	3226	18	22157	40
	Parallel (DnA,ThnA)	(4,3)	2	587	20	3297	28	2957	18	21193	40
$a = -1$	DnA	1	1	1320	-	8929	-	6936	-	54484	-
	TnA	2	1	845	-	5529	-	4331	-	32931	-
	TnA	3	1	847	-	5169	-	4347	-	30748	-
	ThnA	2	1	3053	-	11728	-	12781	-	94324	-
	ThnA	3	1	2751	-	11010	-	13006	-	94255	-
	Parallel (TnA,ThnA)	(2,2)	2	728	28	4660	47	3500	30	25254	63
	Parallel (TnA,ThnA)	(3,3)	2	727	30	4407	48	3681	25	25101	59

- For $a = 1$ the best non-parallelized approach is the double-and-add scalar multiplication while the triple-and-add approaches are slightly slower. The improvement provided by the parallelization is of 4.88% for $n = 127$ and 5.73% for $n = 251$ on an Intel Core i7 processor and of 8% for $n = 127$ and 12% for $n = 251$ on a Snapdragon processor.
- For $a = -1$ the best non-parallelized approach is the triple-and-add method with $w = 2$ for $n = 127$ and $w = 3$ for $n = 251$. The parallelization with $w = 3$ provides a speed-up of 12.3% for $n = 127$ and 13.8% for $n = 251$ on an Intel Core i7 processor and 14% for $n = 127$ and 18% for $n = 251$ on a SnapDragon processor compared to the best non-parallelized approaches.

6.3 Conclusion

In this chapter we presented some results on the parallelization of elliptic curve scalar multiplication. We first presented a Mongtomery-halving approach for scalar multiplication which is a regular and constant time algorithm. This Montgomery-halving is interesting when it is used concurrently to the Montgomery-ladder to get a parallelized version of the Montgomery-ladder scalar multiplication. We then presented a thirding operation in $E(\mathbb{F}_{3^n})$. This lead to a parallel (third,triple)-and-add scalar multiplication in $E(\mathbb{F}_{3^n})$. Experimentation done on an Intel Core i7 and a Qualcomm Snapdragon showed that, since the proposed parallelization involves a large number of inversion, the speed-up is limited when a field inversion is costly compared to a field multiplication.

Chapter 7

Conclusion and future works

7.1 Conclusion

In this HDR thesis we presented some results obtained during the past ten years on efficient and secure implementation of cryptographic protocols.

We started our research work on improving multiplication in finite fields based on quadratic complexity methods. This was the topic of our Phd thesis, we proposed new bases for field representation: the modified polynomial basis [24] and quasi-normal bases [8]. These bases were chosen in order to reduce the complexity of the computation of the matrix entries involved in the matrix-vector formulation of finite field multiplication (cf. Section 3.1.1). We also proposed compact matrix representation in order to reduce the delay of these quadratic space complexity multipliers [9].

During the years following the end of the Phd thesis, we pursued our research on multipliers in finite fields. At this time we focused on multiplication algorithms with subquadratic complexity. The first work done in this direction was on arithmetic modulo a prime integer in the AMNS system (cf. Chapter 5). We introduced a method based on DFT and FFT to multiply element of $\mathbb{F}_p$ represented in an AMNS. We pursued this idea to improve multiplication in $\mathbb{F}_{p^k}$. We used the fact that an AMNS can be constructed with a base which is a root of unity. This approach renders efficient the DFT and FFT computation for polynomial multiplication in $\mathbb{F}_p[T]$ since the multiplication by the considered root unity is really cheap.

We also studied subquadratic methods for multiplication in $\mathbb{F}_{2^n}$ (cf. Chapter 3). These works started with the collaboration with Anwar Hasan in two articles extending the TMVP approach initiated by Fan and Hasan in [46] to multiplication modulo a NAOP [1] and field represented in a Dickson basis [2]. At this time parallel multipliers based on TMVP were the best ones among parallel multipliers for field $\mathbb{F}_{2^n}$ of cryptographic size. Our work extends the use of this approach to a larger class of fields.

We pursued our collaboration with Anwar Hasan in order to improve subquadratic space complexity multiplier based on TMVP for field represented in ONB-II. We analyzed the entire circuitry produced by recursively applying TMVP formula. We identified parts of the circuit which can be removed or replace by another circuit which is less expensive. We called this method the block recombination approach. We get an ONB-II multiplier which reduced significantly the space complexity compared to the original multiplier of [58].

The previous approach was based on the analysis of the entire circuitry of the considered TMVP subquadratic multiplier. We looked at other subquadratic multipliers: the ones based

on polynomial multiplication and Karatsuba formula. This lead us to notice that the optimization initially proposed by Bernstein on two recursions of Karatsuba formula in [48] could be generalized to an arbitrary number of recursions. The resulting multiplier get the expected reduction of the space complexity, but it also get a significant improvement of its critical path delay. This rendered multipliers based on Karatsuba better than the one based on TMVP.

The last works on finite field arithmetic concerned combined operations $AB+AC$ or AB, AC. The technique was to rewrite multiplication algorithm for these combined operation in order to remove some redundant computation involved in two or more multiplication. The specific pattern of operations lead to redundant computations. We did it in [25] for $AB + CD$ with algorithm crafted for software implementation of multiplication $\mathbb{F}_2[x]$. In a recent work [18] we could rewrite the word level form of Montgomery modular multiplication applied to the computation of AB, AC.

Additionally to the works on finite field arithmetic we obtained several results on the improvement of scalar multiplication in an elliptic curve. The first works [23, 20] were done during the preparation of the Phd thesis and during the few years which followed. We provided in these works some improvements for point tripling and addition in curves $E(\mathbb{F}_{3^m})$. Recently we proposed in [6] a thirding formula (division by 3), in order to parallelize part of the computations of a scalar multiplication. We proposed also a parallel version of the Montgomery-ladder for the scalar multiplication.

7.2 Future works

We now present some directions we plan to pursue during the next few years.

7.2.1 Toward further recombined algorithms

One of the main direction of our recent works was the analyzis of recursive algorithm in order to look at the whole recursions and try to identify redundant computations involved in these computations. These strategies were presented in Chapter 3 and Chapter 4. We would like to pursue these strategies in the following directions.

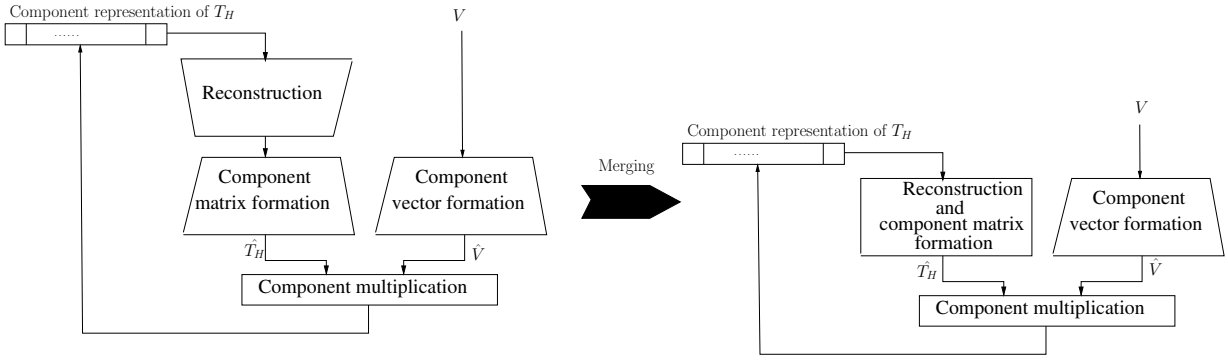
A first direction would be to extend the approach of Bernstein for polynomial multiplication (Section 3.3.1). In this case some computations done in distinct recursions are the same (multiplication by the level factor). Merging the reconstruction of each recursion leads to some saving in complexity. But this is specific to polynomials since these redundant computation are due to the overlap of subsequent products. We plan to look at other algorithms from symbolic calculus which are recursive and also involve polynomial multiplications at some level and see if they can lead to similar improvement.

In a second direction we plan to investigate some strategies related to block recombination of the GHASH multiplier reviewed in Section 3.2.2. We can first notice that when some consecutive operations (CVF,CMF,R,...) are done separately, the delay of the computation is the sum of the delay of each operation. We plan to re-express such consecutive computations, when it is possible, in order to reduce this delay. In counter part this might increase a bit the space complexity. We plan to look at the following two cases:

- *Merging blocks in the multiply-and-add architecture.* We consider the multiply-and-add architecture in the left part of Figure 7.1. This architecture can be used for the implementation of a GHASH function. The temporary results are stored in component formation in a larger register ($n^{\log_2(3)}$ flip-flops). We plan to modify this multiplier by merging the reconstruction block and the CMF block in order to have a single block with

a reduced delay. This is shown in the right side of Figure 7.1. The advantage of this approach is that it will give a multiply-and-add architecture with an expected reduced delay of $\log_2(n)D_X + D_A$. This will make this architecture competitive with quadratic approaches, at least theoretically.

Figure 7.1: Merging blocks in multiply-and-add architecture



- *Merged blocks in optimal-polynomial multiplication.* The multiplier of Bernstein and Lange [71, 72] proceeds as follows:

  1. the elements U and V expressed in a permuted ONB-II $\mathcal{N}' = (\beta^i + \beta^{-i})$ are converted into polynomials $U'(\alpha), V'(\alpha)$ in $\alpha = \beta + \beta^{-1}$
  2. The product of $W'(\alpha) = U'(\alpha) \times V'(\alpha)$ is computed with a subquadratic approach for polynomial multiplication.
  3. We convert $W'(\alpha)$ back to a representation in $\mathcal{N}'$.

  We propose to merge the forward conversion with the component polynomial formation of the Karatsuba multiplier. We will also try to merge the backward conversion with the reconstruction. If both mergings are effective we expect a reduction of the delay from $4\log_2(n)$ to $2\log_2(n)$.

7.2.2 Randomization

During the past twenty years an important amount of work was done on side channel analysis. Some recent results showed that this kind of attacks is an important threat against embedded devices performing cryptographic computations. Specifically, recent attacks called horizontal attacks [100, 101, 102] showed that usual counter-measures like the ones proposed by Coron in [103] (exponent randomization, data blinding) are not efficient. These attacks try to correlate some portions of a single trace in order to find the sequence of operations and then derive the key. They use the fact that if a data is used several times and is encoded in the same way then each time the data is used in a computation this produces a correlated trace. One counter measure to prevent horizontal analysis is to dynamically randomize data involved in the sensitive algorithms. But unfortunately these randomizations are costly: the one involved in [57] require two multiplications per randomization and per data, and the one from [101] require twice the computational effort compared to a regular multiplication.

We have done a first work in this direction [19] which reduces the cost of the dynamic randomization in RNS proposed in [57]. We would like to pursue this approach in a different

way. We can let the multiplicative masking $\widetilde{u} = u \times M \mod p$ evolve freely: we randomize the bases $\mathcal{A}$ and $\mathcal{B}$ but we do not correct the multiplicative mask. This save the costly update of the multiplicative mask, but the randomization is kept effective by the random update of the bases. At the end we will have a multiplicative mask of the form $M = \prod_{i=1}^{n} a_i^{\alpha_i} \times \prod_{i=1}^{n} b_i^{\beta_i}$. So we need a strategy to keep the exponents α_i and β_i in a limited range $< \gamma$, in order to to be able to remove the mask with a reasonable computational effort.

We plan to pursue this direction in order to develop other strategies for dynamic randomization remaining efficient in practice: for example in usual binary representation or in other non-conventional representation system like RNS.

A second important concerns related to randomization for the protection of embedded devices is the quality of the randomization. This problem is, for now, not sufficiently understood. To illustrate this problematic we just remind that the classical blinding of a secret ECC scalar is done as follows: $k' = k + r \times N$ where $N = \#E(\mathbb{F}_q)$ is the order of the group and r is a 20 bit random integer. But Joye *et al.* in [104] noticed that since $\#E(\mathbb{F}_q) = q - 2t + 1$ with $|t| \leq \sqrt{q}$ then the upper half bits of N are those of q. Then, when q is sparse, these bits are mostly zeros and no blinding are done on the most significant bits of $k' = k + r \times N$.

Another remark to illustrate how tricky is this problematic: we did a quick experiment on an usual randomization strategy on elliptic curve which is the projective representation. We have generated all possible representatives of a point and then have evaluated the distribution of Hamming weight of these representatives. We noticed that there is a small bias in this distribution. This bias is really small so it cannot be exploited to mount attack but it is present and this should not be neglected for security purpose.

Consequently, we believe that a theoretic and practical evaluation of some proposed randomization strategies have to be conducted in order to have a better evaluation of the robustness of these randomizations have to be conducted.

Chapter 8

Bilbiography

8.1 Personnal publications

Journal articles

[1] M.A. Hasan, A.H. Namin, and C. Negre. "Toeplitz Matrix Approach for Binary Field Multiplication Using Quadrinomials". In: *IEEE Trans. VLSI Syst.* 20.3 (2012), pp. 449–458.

[2] M.A. Hasan and C. Negre. "Low Space Complexity Multiplication over Binary Fields with Dickson Polynomial Representation". In: *IEEE Trans. Computers* 60.4 (2011), pp. 602–607.

[3] M.A. Hasan, N. Meloni, A.H. Namin, and C. Nègre. "Block Recombination Approach for Subquadratic Space Complexity Binary Field Multiplication Based on Toeplitz Matrix-Vector Product". In: *IEEE Trans. Computers* 61.2 (2012), pp. 151–163.

[4] J. Adikari, A.F. Barsoum, M.A. Hasan, A.H. Namin, and C. Negre. "Improved Area-Time Tradeoffs for Field Multiplication Using Optimal Normal Bases". In: *IEEE Trans. Computers* 62.1 (2013), pp. 193–199.

[5] C. Negre. "Efficient binary polynomial multiplication based on optimized Karatsuba reconstruction". In: *J. Cryptographic Engineering* 4.2 (2014), pp. 91–106.

[6] C. Negre and J.-M. Robert. " New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic". In: *IEEE Trans. Computers* to appear (2015).

[7] N. Meloni, C. Negre, and M.A. Hasan. "High performance GHASH and impacts of a class of unconventional bases". In: *J. Cryptographic Engineering* 1.3 (2011), pp. 201–218.

[8] C. Negre. "Finite field arithmetic using quasi-normal basis". In: *Finite Fields and Their Applications* 13 (2007), pp. 635–647.

[9] C. Negre. "Efficient parallel multiplier in shifted polynomial basis". In: *Journal of Systems Architecture* 53.2-3 (2007), pp. 109–116.

[10] M. Cenk, M.A. Hasan, and C. Negre. "Efficient Subquadratic Space Complexity Binary Polynomial Multipliers Based on Block Recombination". In: *IEEE Trans. Computers* 63.9 (2014), pp. 2273–2287.

[11] M. Cenk, C. Negre, and M.A. Hasan. "Improved Three-Way Split Formulas for Binary Polynomial and Toeplitz Matrix Vector Products". In: *IEEE Trans. Computers* 62.7 (2013), pp. 1345–1361.

[12] M.A. Hasan and C. Negre. "Multiway Splitting Method for Toeplitz Matrix Vector Product". In: *IEEE Trans. Computers* 62.7 (2013), pp. 1467–1471.

[13] M.A. Hasan and C. Nègre. "Sequential multiplier with sub-linear gate complexity". In: *J. Cryptographic Engineering* 2.2 (2012), pp. 91–97.

[14] J.-C. Bajard, C. Negre, and T. Plantard. "Subquadratic Space Complexity Binary Field Multiplier Using Double Polynomial Representation". In: *IEEE Trans. Computers* 59.12 (2010), pp. 1585–1597.

[15] J.-C.Bajard, L. Imbert, and C. Negre. "Arithmetic Operations in Finite Fields of Medium Prime Characteristic using Lagrange Representation". In: *IEEE trans. comp.* (2006).

Conference articles

[16] C. Negre and T. Plantard. "Efficient Modular Arithmetic in Adapted Modular Number System using Lagrange Representation". In: *Proceedings of Australasian Conference on Information Security and Privacy (ACISP 08)*. Vol. 5107. LNCS. 2008, pp. 463–477.

[17] N. El Mrabet and C. Negre. "Finite field multiplication combining AMNS and DFT approach for pairing cryptography". In: *Proceedings of Information Security and Privacy, 14th Australasian Conference (ACISP 2009)*. Vol. 5594. LNCS. Springer, 2009.

[18] C. Negre, T. Plantard, and J.-M. Robert. "Efficient Modular Exponentiation Based on Multiple Multiplications by a Common Operand". In: *IEEE Symposium on Computer Arithmetic 2015*. 2015, pp. 119–126.

[19] C. Negre and G. Perin. "Trade-Off Approaches for Leak Resistant Arithmetic in RNS". In: *ACISP 2015*. to appear. Springer, 2015.

[20] K.-H. Kim and C. Negre. "Point Multiplication on Supersingular Elliptic Curves Defined over Fields of Characteristic 2 and 3". In: *SECRYPT'08, Porto, Portugal*. 2008, pp. 373–376. ISBN: 978-989-8111-59-3.

[21] A. Hasan and C. Negre. "Subquadratic Space Complexity Multiplier for a Class of Finite Fields Using Toeplitz Matrix Approach". In: *ARITH 19, 19th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society, 2009, pp. 67–75.

[22] N. Meloni, C. Negre, and M.A. Hasan. "High Performance GHASH Function for Long Messages". In: *Applied Cryptography and Network Security, 8th International Conference (ACNS 2010)*. Vol. 6123. LNCS. 2010, pp. 154–167.

[23] C. Negre. "Scalar Multiplication on Elliptic Curves Defined over Fields of Small Odd Characteristic". In: *Indocrypt 2005, Bangalore India*. Vol. 3797. LNCS. Dec. 2005, pp. 389–402.

[24] C. Negre. "Quadrinomial Modular Multiplication using Modified Polynomial Basis". In: *Proceedings of ITCC 2005, Las Vegas USA*. LNCS. Apr. 2005, pp. 550–555.

[25] C. Negre and J.-M. Robert. "Impact of Optimized Field Operations AB, AC and $AB+CD$ in Scalar Multiplication over Binary Elliptic Curve". In: *AFRICACRYPT 2013*. Vol. 7918. LNCS. Springer, 2013, pp. 279–296.

[26] J. Adikari, M.A. Hasan, and C. Negre. "Towards Faster and Greener Cryptoprocessor for Eta Pairing on Supersingular Elliptic Curve over $\mathbb{F}_{2^{1223}}$". In: *Selected Areas in Cryptography (SAC 2012)*. Vol. 7707. LNCS. Springer, 2012, pp. 166–183.

[27] M. Cenk, C. Negre, and M.A. Hasan. "Improved Three-Way Split Formulas for Binary Polynomial Multiplication". In: *Selected Areas in Cryptography (SAC 2011*. Vol. 7118. LNCS. Springer, 2011, pp. 384–398.

[28] Y. Li and C. Negre. "An efficient multiplication algorithm using binomial residue representation". In: *SECRYPT'08, Porto, Portugal*. 2008, pp. 319–324. ISBN: 978-989-8111-59-3.

[29] A. Hasan and C. Negre. "Subquadratic Space Complexity Multiplication over Binary Fields with Dickson Polynomial Representation". In: *Proceedings of WAIFI 2008, Sienna, Italy*. Vol. 5130. LNCS. 2008, pp. 88–102.

[30] P. Giorgi, C. Negre, and T. Plantard. "Subquadratic Binary Field Multiplier in Double Polynomial System". In: *SECRYPT'07, Barcelona, Spain*. 2007.

[31] C. Negre. "Parallel Multiplication in GF(2^n) using Condensed Matrix Representation". In: *SECRYPT'06, Setúbal,Portugal*. Aug. 2006.

[32] C. Negre. "Finite Field Multiplication in Lagrange Represenation Using Fast Fourrier Transform". In: *SECRYPT'06,Setúbal,Portugal*. Aug. 2006.

[33] C. Negre. "Exponentiation to the power p in GF(p^k) using Variants of Montgomery Modular Arithmetic". In: *Proceedings of Nordsec 2005, Tartu Estonia*. Oct. 2005.

8.2 Other references

[34] D.R. Stinson. *Cryptography - theory and practice*. Discrete mathematics and its applications series. CRC Press, 1995.

[35] P. Gallagher. "Digital Signature Standard (DSS)". In: *Federal Information Processing Standards Publications*. Vol. FIPS 186-3. Federal Information Processing Standards Publications (NIST). National Institute of Standards and Technology, 2009, p. 93.

[36] W. Diffie and M.E. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[37] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126.

[38] V. Miller. "Use of elliptic curves in cryptography". In: *Advances in Cryptology, proceeding's of CRYPTO'85*. Vol. 218. LNCS. Springer-Verlag, 1986, pp. 417–426.

[39] N. Koblitz. "Elliptic curve Cryptosystems". In: *Mathematics of Computation* 48 (1987), pp. 203–209.

[40] J. M. Pollard. "Monte Carlo methods for index computation mod p". In: *Mathematics of Computation* 32 (1978), pp. 918–924.

[41] A. Joux. "Faster Index Calculus for the Medium Prime Case Application to 1175-bit and 1425-bit Finite Fields". In: *EUROCRYPT 2013*. Vol. 7881. LNCS. Springer, 2013, pp. 177–193.

[42] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. "A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic". In: *Advances in Cryptology - EUROCRYPT 2014*. Vol. 8441. LNCS. Springer, 2014, pp. 1–16.

[43] P.C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems." In: *CRYPTO'96*. Vol. 1109. LNCS. 1996, pp. 104–113.

[44] P.C. Kocher, J. Jaffe, and B. Jun. "Differential Power Analysis". In: *CRYPTO'99*. Vol. 1666. LNCS. 1999, pp. 388–397.

[45] J.-J. Quisquater and D. Samyde. "ElectroMagnetic Analysis (EMA): Measures and Couter-Measures for Smard Cards". In: *Smart Card Programming and Security (E-smart 2001)*. Vol. 2140. Lecture Notes in Computer Science. Springer-Verlag, Sept. 2001, pp. 200–210.

[46] H. Fan and M.A. Hasan. "A New Approach to Sub-quadratic Space Complexity Parallel Multipliers for Extended Binary Fields". In: *IEEE Trans. Computers* 56.2 (2007), pp. 224–233.

[47] S. Winograd. *Arithmetic Complexity of Computations*. Society For Industrial & Applied Mathematics, U.S., 1980.

[48] D. J. Bernstein. "Batch Binary Edwards". In: *Proceedings of Advances in Cryptology - CRYPTO 2009*. Vol. 5677. LNCS. Springer, 2009, pp. 317–336.

[49] D. A. McGrew and J. Viega. "The Security and Performance of the Galois/Counter Mode (GCM) of Operation". In: *INDOCRYPT*. Vol. 3348. LNCS. 2004, pp. 343–355.

[50] J.-C. Bajard, L. Imbert, and T. Plantard. "Modular Number Systems: Beyond the Mersenne Family". In: *Selected Area in Cryptoraphy*. Vol. 3357. LNCS. Springer, 2004, pp. 159–169.

[51] P. L. Montgomery. "Modular Multiplication Without Trial Division". In: *Mathematics of Computation* 44.170 (1985), pp. 519–521.

[52] A. Bosselaers, R. Govaerts, and J. Vandewalle. "Comparison of Three Modular Reduction Functions". In: *CRYPTO'93*. Vol. 773. LNCS. 1993, pp. 175–186.

[53] M. Joye and S.-M. Yen. "The Montgomery Powering Ladder". In: *CHES 2002*. Vol. 2523. LNCS. 2002, pp. 291–302.

[54] M. Joye and M. Tunstall. "Exponent Recoding and Regular Exponentiation Algorithms". In: *AFRICACRYPT 2009*. Vol. 5580. LNCS. 2009, pp. 334–349.

[55] K.C. Posch and R. Posch. "Modulo Reduction in Residue Number Systems". In: *IEEE Trans. Parallel Distrib. Syst.* 6.5 (1995), pp. 449–454.

[56] S. Kawamura, M. Koike, F. Sano, and A. Shimbo. "Cox-Rower Architecture for Fast Parallel Montgomery Multiplication". In: *Proc. EUROCRYPT 2000*. Vol. 1807. LNCS. Springer Verlag, 2000, pp. 523–538.

[57] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia. "Leak Resistant Arithmetic". In: *CHES*. Vol. 3156. LNCS. Springer, 2004, pp. 62–75.

[58] H. Fan and M.A. Hasan. "Subquadratic Computational Complexity Schemes for Extended Binary Field Multiplication Using Optimal Normal Bases". In: *IEEE Trans. Computers* 56.10 (2007), pp. 1435–1437.

[59] E.D. Mastrovito. "VLSI Architectures for Computation in Galois Fields". PhD thesis. Linkoping, Sweden: Linkoping University, Department of Electrical Engineering, 1991.

[60] H. Fan and Y. Dai. "Fast Bit-Parallel GF(2^n) Multiplier for All Trinomials". In: *IEEE Trans. Computers* 54.4 (2005), pp. 485–490.

[61] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson. "Optimal Normal Bases in GF(p^n)". In: *Discrete Appl. Math.* 22.2 (1989), pp. 149–161.

[62] S. Gao and H. W. Lenstra Jr. "Optimal Normal Bases". In: *Designs, Codes and Cryptography* 2.4 (1992), pp. 315–323.

[63] Ç.K. Koç and B. Sunar. "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields". In: *IEEE Trans. Computers* 47.3 (1998), pp. 353–356.

[64] M.A. Hasan, M. Wang, and V.K. Bhargava. "A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields". In: *IEEE Trans. Computers* 42.10 (1993), pp. 1278–1280.

[65] C.K. Koc and B. Sunar. "An Efficient Optimal Normal Basis Type II Multiplier". In: *IEEE Trans. Computers* 50 (2001).

[66] E.R. Berlekamp. "Bit-serial Reed-Solomon encoder". In: *IEEE Trans. Information Theory* IT-28 (1982).

[67] M.A. Hasan H. Wu and L.F. Blake. "New Low-Complexity Bit-Parallel Finite Field Multipliers Using Weakly Dual Bases". In: *IEEE Trans. Computers* 47 (1998).

[68] R. Katti and J. Brennan. "Low Complexity Multiplication in a Finite Field Using Ring Representation". In: *IEEE Trans. Comput.* 52.4 (2003), pp. 418–427.

[69] M.A. Hasan and V.K. Bhargava. "Division and bit-serial multiplication over GF(q^m)". In: *IEE Proc., part E* 139 (1992), pp. 230–236.

[70] R. Furness, M. Benaissa, and S.T.J. Fenn. "GF(2^m) multiplication over triangular basis for design of Reed-Solomon codes". In: *IEE Proc.-Compt. Digit. Tech.* 145.6 (1998), pp. 437–443.

[71] J. von zur Gathen, A. Shokrollahi, and J. Shokrollahi. "Efficient Multiplication Using Type 2 Optimal Normal Bases". In: *WAIFI '07*. LNCS. Madrid, Spain, 2007, pp. 55–68.

[72] D.J. Bernstein and T. Lange. "Type-II Optimal Polynomial Bases". In: *WAIFI 2010*. Vol. 6087. LNCS. 2010, pp. 41–61.

[73] J. Sun, M. Gu, K.-Y. Lam, and H. Fan. "Overlap-free Karatsuba-Ofman Polynomial Multiplication Algorithm". In: *IET Information Security* 4 (2010), pp. 8–14.

[74] G. Zhou and H. Michalik. "Comments on "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Field"". In: *IEEE Trans. Computers* 59.7 (2010), pp. 1007–1008.

[75] M. Quercia. *Calcul multiprécision*. 2000.

[76] B. Sunar. "A Generalized Method for Constructing Subquadratic Complexity GF(2^k) Multipliers". In: *IEEE Trans. Computers* 53.9 (2004), pp. 1097–1105.

[77] A. Satoh. "High-speed hardware architectures for authenticated encryption mode GCM". In: *IEEE International Symposium on Circuits and Systems - ISCAS*. 2006, pp. 4831–4834.

[78] A. Satoh. "High-Speed Parallel Hardware Architecture for Galois Counter Mode". In: *IEEE International Symposium on Circuits and Systems - ISCAS*. 2007, pp. 1863–1866.

[79] J. A. Gordon. "A Very simple method to find the minimum polynomial of an arbitrary nonzero element of a finite field". In: *Electronics Letters* 12.25 (1976), pp. 663–664.

[80] H.L. Garner. "The Residue Number System". In: *IRE Trans. on Elctronic Computers* 8 (1959), pp. 140–147.

[81] A. Schonhage and V. Strassen. "Schnelle multiplikation grosser zahlen". In: *Computing* 7 (1971), pp. 281–292.

[82] P. Barrett. “Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor”. In: *Advances in Cryptology – CRYPTO '86*. Ed. by A. M. Odlyzko. Vol. 263. LNCS. Springer-Verlag, 1986, pp. 311–326.

[83] H. Cohen. *A course in computational algebraic number theory*. Vol. 138. Graduate Texts in Mathematics. Berlin: Springer-Verlag, 1993.

[84] J.-C. Bajard, L. Imbert, and T. Plantard. “Arithmetic Operations in the Polynomial Modular Number System”. In: *IEEE Symposium on Computer Arithmetic (ARITH-17 2005)*. IEEE Computer Society, 2005, pp. 206–213.

[85] H. Minkowski. *Geometrie der Zahlen*. Leipzig: B. G. Teubner, 1896.

[86] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovàsz. “Factoring polynomials with rational coefficients”. In: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[87] N. Koblitz and A. Menezes. “Pairing-based cryptography at high security levels”. In: *Proceedings of the Tenth IMA International Conference on Cryptography and Coding*. Vol. 3796. LNCS. 2005, pp. 13–36.

[88] J.C. Bajard and N. El Mrabet. “Pairing in cryptography: an arithmetic point of view”. In: *Advanced Signal Processing Algorithms, Architectures and Implementations XVI, SPIE*. 2007.

[89] Nadia El Mrabet and Nicolas Gama. “Efficient Multiplication over Extension Fields”. In: *WAIFI 2012*. Vol. 7369. LNCS. Springer, 2012, pp. 136–151.

[90] R.P. Gallant, R.J. Lambert, and S.A. Vanstone. “Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms”. In: *Advances in Cryptology - CRYPTO 2001*. Vol. 2139. LNCS. Springer, 2001, pp. 190–200.

[91] E.W. Knudsen. “Elliptic Scalar Multiplication Using Point Halving”. In: *Advances in Cryptology - ASIACRYPT '99*. 1999.

[92] K. Fong, D. Hankerson, J. López, and A. Menezes. “Field Inversion and Point Halving Revisited”. In: *IEEE Trans. Computers* 53.8 (2004), pp. 1047–1059.

[93] J. Taverne, A. Faz-Hernández, D.F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López. “Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction”. In: *J. Cryptographic Engineering* 1.3 (2011), pp. 187–199.

[94] J. Taverne, A. Faz-Hernández, D.F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López. “Software Implementation of Binary Elliptic Curves: Impact of the Carry-Less Multiplier on Scalar Multiplication”. In: *CHES 2011*. Vol. 6917. LNCS. Springer, 2011, pp. 108–123.

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

[96] J. López and R. Dahab. “High-Speed Software Multiplication in $\mathbb{F}_{2^m}$”. In: *INDOCRYPT 2000*. Vol. 1977. LNCS. Springer, 2000, pp. 203–212.

[97] K.-H. Kim, S.I. Kim, and J.S. Choe. *New Fast Algorithms For Arithmetic on Elliptic Curves over Finite Fields of Characteristic Three*. Tech. rep. Report 2007/179. Cryptology ePrint Archive, 2007.

[98] R.R. Farashahi, H. Wu, and C. Zhao. “Efficient Arithmetic on Elliptic Curves over Fields of Characteristic Three”. In: *Selected Areas in Cryptography (SAC 2012)*. Vol. 7707. LNCS. Springer, 2013, pp. 135–148.

[99] O. Ahmadi, D. Hankerson, and A. Menezes. "Software Implementation of Arithmetic in $\mathbb{F}_{3^m}$". In: *WAIFI 2007*. Vol. 4547. LNCS. 2007, pp. 85–102.

[100] C.D. Walter. "Sliding Windows Succumbs to Big Mac Attack". In: *CHES*. LNCS Generators. 2001, pp. 286–299.

[101] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. "Horizontal Correlation Analysis on Exponentiation". In: *Proceedings of ICICS 2010*. Vol. 6476. LNCS. Springer, 2010, pp. 46–61.

[102] G. Perin, L. Imbert, L. Torres, and P. Maurine. "Attacking Randomized Exponentiations Using Unsupervised Learning". In: *Proceedings of COSADE 2014*. Vol. 8622. LNCS. Springer, 2014, pp. 144–160.

[103] J.-S. Coron. "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems". In: *CHES*. 1999, pp. 292–302.

[104] M. Ciet and M. Joye. "(Virtually) Free Randomization Techniques for Elliptic Curve Cryptography". In: *ICICS 2003*. Vol. 2836. LNCS. Springer, 2003, pp. 348–359.