Common Subexpression Algorithms for Space-Complexity Reduction of Gaussian Normal Basis Multiplication

Reza Azarderakhsh, David Jao, and Hao Lee

***Abstract*—The use of normal bases for representing elements in a binary field is attractive in some applications because it is easy to perform squaring operations in hardware. In such cases, the costs of implementing the multiplication operation become a primary concern. We present new algorithms for reducing the space complexity of Gaussian normal basis multipliers over binary fields $GF(2^m)$, where m is odd. Compared with previous results, our approach incurs no additional costs in time complexity, and achieves improvements in space complexity over a wide range of finite fields and digit sizes. For the binary fields specified in the NIST FIPS 186-3 elliptic curve digital signature algorithm standards document, our algorithms reduce by 16% (respectively, 27%) the number of XOR gates needed for the implementation of a digit-level parallel-input parallel-output multiplier over a 163-bit (respectively, 409 bit) binary field.**

***Index Terms*—Elliptic curve cryptography (ECC), binary extension field, approximation algorithm, complexity reduction algorithm, Gaussian normal basis.**

I. Introduction

Finite field arithmetic has several applications in coding theory, classical, and modern cryptography. Cryptographic algorithms such as elliptic curve cryptography (ECC) require different finite field arithmetic operations including multiplication, addition, squaring, and inversion. Among these operations, multiplication plays an important role in determining the efficiency of such algorithms for two reasons. First, its computation is complicated in comparison to other operations, and second, several applications require many multiplications. Therefore, efficient implementation of finite field multipliers is crucial. For binary fields $GF(2^m)$, field elements are typically represented using either polynomial (standard) bases or normal bases, as described in the NIST [1] and IEEE standards [2]. When working in a normal basis, squaring is simply a right-cyclic shift, which can be implemented very efficiently in hardware. A Gaussian Normal Basis (GNB) is a special class of normal basis which admits an efficient field multiplication implementation. The existence of GNBs may depend on generalized Riemann hypothesis (GRH) [3]. In [4], it has been observed that there will exist some k for which there is a GNB of type k which provides best alternative when optimal normal basis does not exist. Massey and Omura in [5] were the first to propose a bit-level normal basis multiplier with the structure of parallel-input and serial-output (PISO) for use in binary field arithmetic. Geiselmann and Gollman in [6] presented another bit-level multiplier with a parallel-input and parallel-out (PIPO) architecture, and Beth and Gollman [7] proposed a serial-input and parallel-out structure for bit-level multiplication over normal basis.

Digit-level multipliers are an alternative to bit-level and bit-parallel multipliers in which the digit size can be chosen depending on the amount of the resources available. They can be easily scaled up to perform bit-parallel multipliers and scaled down to perform as bit-level multipliers, for high performance and resource-constrained applications, respectively. Recently, Reyhani-Masoleh in [8] and Kim et al. in [9] constructed a digit-level parallel-input and parallel-output (DL-PIPO) multiplier architecture which has been employed for point multiplication of ECC by several researchers. Their multiplier is based on repeating the module which implements the multiplication matrix. Given a digit-size d with $1 \leq d \leq m$, it takes $q = \lceil \frac{m}{d} \rceil$ clock cycles to generate all coordinates of the product. In [10] and [11], Azarderakhsh et al. proposed a common subexpression elimination algorithm to reduce the space complexity of DL-PIPO and DL-SIPO/DL-PISO multiplier architectures, respectively. Recently, digit-level multipliers have been employed to develop ECC-based crypto-processors. For istance one can refer to [12]–[14].

As recommended by NIST [1], ECC over $GF(2^m)$ requires 163, 233, 283, 409, and 571-bit key sizes for 80, 112, 128, 192, and 256-bit security levels, respectively. As of 2010, the 80-bit and 112-bit security levels are considered obsolete and today's security requirements for elliptic curve cryptography demands an increase in key size to at least 283 bits to achieve the 128-bit security level [15], [16]. As the key sizes increase, the space complexity of the field multipliers increase as well and hence efficient hardware implementation becomes challenging. In this paper, we address this issue comprehensively

Manuscript received January 24, 2014; revised February 20, 2015; accepted February 28, 2015. Date of publication March 9, 2015; date of current version April 17, 2015. D. Jao was supported by the Natural Sciences and Engineering Research Council of Canada through the Collaborative Research and Development under Grant CRDPJ 405857-10.

R. Azarderakhsh is with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA (e-mail: rxaeec@rit.edu).

D. Jao and H. Lee are with the Center for Applied Cryptographic Research, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: djao@uwaterloo.ca; h96lee@uwaterloo.ca).

Communicated by N. Kashyap, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2015.2409833

and present new algorithms for complexity reduction of GNB multipliers. First, we prove conjectures that were left unproven in previous works that used special properties of Gaussian normal bases. Second, we present two new complexity reduction algorithms to efficiently reduce the space complexity and the number of required XORs in the multiplication matrix for digit-level multiplier architectures. The first algorithm produces optimal output but runs in exponential time. The second algorithm is an approximation algorithm which runs in polynomial time. Our approximation algorithm has two main advantages compared to previous work of [10] and [11]. First of all, it is more efficient in terms of finding common subexpressions and consequently reducing the number of required XORs. Second, it is scalable for larger field sizes and runs in polynomial time. Our experiments based on simulations indicate that our approximation algorithm always produces smaller multipliers compared to previous work. For instance, we require 24% fewer XOR gates to implement a bit-parallel ($d = m$) multiplier architecture over $GF(2^{283})$, which is a type 6 GNB. Moreover, for bit-level GNB multipliers our algorithms yield to requiring fewer XORs for all field sizes which is attractive for resource-constrained applications such as smart cards and RFIDs. We stress that our algorithms for finding compact multiplication matrices are not limited only to the fields in the NIST standard, but work in general for all binary fields admitting suitable GNBs.

The rest of the paper is organized as follows. In Section 2, we provide relevant background on normal basis multiplication. In Section 3, we prove some new properties of GNB multiplication matrices. In Section 4, we present our complexity reduction algorithms. In Section 5, we provide simulation results and compare them to the leading ones available in the literature for bit-level and digit-level architectures. Finally, in Section VI we conclude the paper.

II. Preliminaries

A. Gaussian Normal Basis

Elements of a finite field $GF(2^m)$ can be represented using a normal basis $N = \left\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\right\}$ where $\beta \in GF(2^m)$ is a normal element of $GF(2^m)$ (an element for which N forms a basis). For any element $A \in GF(2^m)$, we use the notation $A = (a_0, a_1, \ldots, a_{m-1})$, where $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, with $a_i \in GF(2)$ [17].

Definition 1 [4], [18], [19]: Let m and T be positive integers such that $p = mT + 1$ be a prime number and $\gcd\left(\frac{mT}{k}, m\right) = 1$, where k is the multiplicative order of 2 modulo p. Let α be a primitive $(mT+1)$-th root of unity in $GF(2^{Tm})$. Then, for any primitive T-th root of unity τ in $\mathbb{Z}_p$, $\beta = \sum_{i=0}^{T-1} \alpha^{\tau^i}$ generates a normal basis of $GF(2^m)$ over $GF(2)$ given by $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$, which is called a Gaussian normal basis (GNB) of type T.

The type T determines the space and time complexities of GNB multiplication. Multiplication over a GNB is based on a multiplication matrix $\mathbf{M}_{m\times m}$, whose entries are zeros and ones. For details on the computation of the multiplication matrix $\mathbf{M}_{m\times m}$ we refer to Ash et al. [4]. It is well known that when T is even, the multiplication matrix has the following properties: (a) the matrix $\mathbf{M}$ is symmetric, (b) its diagonal entries are all zero except for the last one, (c) the first row has just one non-zero entry, and (d) row$(m-i)$ is the i-fold left cyclic shift of row(i) for all $1 \le i \le m-1$. Since the matrix is a sparse (0, 1)-matrix, for simplicity and efficiency, it is common practice to store the column numbers of the multiplication matrix $\mathbf{M}$ in which nonzero entries appear, instead of the whole $\mathbf{M}$. Accordingly, we can store those column numbers for each row from 1 up to $m-1$, obtaining a new matrix $\mathbf{R}_{(m-1)\times T}$ having its first row removed [8]. The $\mathbf{R}$ matrix therefore satisfies $R(m-i, j) = R(i, j) + i \bmod m$, $1 \le i \le \frac{m-1}{2}$, $1 \le j \le T$. It should be noted that the $\mathbf{R}$ matrix is more correctly described as a list of lists, since not all entries are filled. In the rest of this paper, we work primarily with the $\mathbf{R}$ matrix instead of the multiplication matrix $\mathbf{M}$. Every GNB satisfies the complexity bound $C_N \le Tm - 1$, where C_N is the number of ones in the multiplication matrix $\mathbf{M}$ or the number of entries in the $\mathbf{R}$ matrix [4]. Let $A = (a_0, a_1, \cdots, a_{m-1}) = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ and $B = (b_0, b_1, \cdots, b_{m-1}) = \sum_{j=0}^{m-1} b_j \beta^{2^j}$ be two field elements over $GF(2^m)$ and assume $C \in GF(2^m)$ be their product, i.e., $C = (c_0, c_1, \cdots, c_{m-1}) = AB$.

Proposition 2 [4]: Let $p = Tm + 1$. Then every $k \in GF(p)\setminus\{0\}$ can be written uniquely as

$$k = 2^i u^j \mod p,$$

where u is a primitive T-th root of unity, and $0 \le i < m$, $0 \le j < T$.

Let F to be the map such that $F(k) = F(2^i u^j) = i$ for all $k \in GF(p)\setminus\{0\}$. This is well-defined by the previous proposition. Therefore, the first coordinate, i.e., c_0 of $C = A \times B$ can be obtained from the following summation:

$$c_0 = \sum_{k=1}^{p-2} a_{F(k+1)} b_{F(p-k)}. \tag{1}$$

If T is even, the GNB is a self-dual basis [20], so $F(k) = F(p-k)$ [21] and hence the multiplication matrix is symmetric and the summation can be simplified as follows [8]:

$$\begin{aligned} c_0 &= a_0 b_1 + \sum_{k=2}^{p-2} a_{F(k)} b_{F(k+1)} \\ &= a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{F(k)=i} b_{F(k+1)} \right). \end{aligned} \tag{2}$$

This expression is critical in our analysis of the properties of the GNB in Section 3. It will be important to study values of $F(k+1)$ given that $F(k) = i$.

From this equivalent construction, the multiplication matrix $\mathbf{R}$ can be derived [8], and one can write c_0 as

$$c_0 = a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{j=1}^{T} b_{R(i,j)} \right), \tag{3}$$

where $R(i, j)$ denotes the (i, j)-th element of the $\mathbf{R}_{(m-1)\times T}$ matrix, with $0 \le R(i, j) \le m-1$, $1 \le i \le m-1$, $1 \le j \le T$. Note that the term $a_i b_j$ simply denotes a 1 in the multiplication matrix $\mathbf{M}_{m\times m}$ and is removed from the multiplication matrix $\mathbf{R}_{(m-1)\times T}$. If a term $a_i b_{F(k+1)}$ occurs twice in equation (2), then it makes no contribution to the $\mathbf{R}$ matrix as the $\mathbf{R}$ matrix is constructed in modulo 2. Other elements of product C can be obtained by cyclic shifting the input operands. In the following, we give an illustrative example about multiplication matrix over GNB.

Example 3: Consider the finite field $GF(2^7)$ is generated over GNB with a type 4. The following multiplication matrix from [4] is given in [1]:

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}_{7\times 7},$$

$$\mathbf{R} = \begin{pmatrix} 0 & 2 & 5 & 6 \\ 1 & 3 & 4 & 5 \\ 2 & 5 & - & - \\ 2 & 6 & - & - \\ 1 & 2 & 3 & 6 \\ 1 & 4 & 5 & 6 \end{pmatrix}_{6\times 4}.$$

Based on (3) one can obtain c_0 as follows:

$$\begin{aligned} c_0 = {} & a_0 b_1 + a_1(b_0 + b_2 + b_5 + b_6) + a_2(b_1 + b_3 + b_4 + b_5) \\ & + a_3(b_2 + b_5) + a_4(b_2 + b_6) + a_5(b_1 + b_2 + b_3 + b_6) \\ & + a_6(b_1 + b_4 + b_5 + b_6). \end{aligned} \tag{4}$$

As one can see, in (4) we can reuse some repeated signals such as $(b_2 + b_5)$, $(b_2 + b_6)$, etc. which are linear combinations of input operand B and reduce the number of XORs in hardware implementations of GNB multipliers. However determining these linear combinations is challenging for larger fields sizes (recommended for ECDSA by NIST) for digit-level architectures to implement (3). We deeply investigate this subexpression sharing in this work.

B. Digit-Level GNB Multipliers

1) Digit-Level Serial Input and Parallel Output (DL-SIPO) Multiplier: In [11], a digit-level serial-input and parallel-out (DL-SIPO) GNB multiplier architecture is proposed. In this multiplier, one of the operands is fully available while the other one is available in a digit-serial fashion. A special module (called a P module) is employed to implement the multiplication matrix $\mathbf{R}_{(m-1)\times T}$ and d copies of this module $(1 \le d \le m)$ are needed in order to perform a multiplication in $q = \lceil \frac{m}{d} \rceil$ clock cycles to generate all coordinates of the product. To reduce space complexity, $\mathbf{Q}$ module is obtained with combining the d shifted versions of P module. Given a digit-size d, one can construct the $\mathbf{Q}$ module by appending to the $\mathbf{R}$ matrix a 1-time right-cyclic shifted version of $\mathbf{R}$ (but without row 0), followed by a two-time right-cyclic shifted version (having row 0 removed), continuing up to a $d-1$ times right-cyclic shifted version [11] as:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{R}^{(0)} \\ \mathbf{R}^{(1)} \\ \mathbf{R}^{(2)} \\ \vdots \\ \mathbf{R}^{(d-1)} \end{pmatrix},$$

where $\mathbf{R}^{(\ell)}$, $0 \le \ell \le d-1$ is an ℓ-times right cyclic shifted version of $\mathbf{R}$.

Finally, a common subexpression elimination algorithm is used to reduce the number of XORs needed to implement the $\mathbf{Q}$ module and hence reduce the complexity of the overall multiplier [11].

2) Digit-Level Parallel Input and Parallel Output (DL-PIPO) Multiplier: In [8] and [9], a digit-level parallel-input and parallel-output (DL-PIPO) multiplier architecture is proposed. This multiplier only requires implementing half of the multiplication matrix $\mathbf{R}_{(m-1)\times T}$ (which is called a $\mu_{(\frac{m-1}{2}\times T)}$ matrix) due to the fact that both input operands are fully available during the multiplication process and the products will be available after the last clock cycle. Recently, this multiplier has been modified by combining the modules which implement the μ matrix by appending left-cyclic shift copies of μ to itself $d-1$ times, constructing a big ρ module as follows:

$$\rho = \begin{pmatrix} \mu_{(0)} \\ \mu_{(1)} \\ \mu_{(2)} \\ \vdots \\ \mu_{(d-1)} \end{pmatrix},$$

where $\mu_{(\ell)}$, $0 \le \ell \le d-1$ indicates an ℓ-times left cyclic shifted version of the matrix μ. Then, the space complexity is reduced using a common subexpression elimination algorithm proposed in [10].

III. Properties of GNB Multiplication Matrices

In this section, we prove several attractive properties of the GNB multiplication matrix which we employ in the following sections for our new complexity reduction algorithms.

Let u be a primitive T-th root of unity in GF$(Tm+1)$. Note that all non-zero elements in GF$(Tm+1)$ can be written uniquely as $2^i u^j$ where $0 \le i < m$ and $0 \le j < T$. Therefore, we have the following theorem.

Theorem 4:

1) $F\left(2^{-1}\right) = m - 1$.
2) $F(u^i + 1) = F(u^{T-i} + 1)$ *for every* $0 \le i < T - 1$
3) *Let* T *be even. Write* $F(k) = i_1$ *and* $F(k+1) = i_2$*. Then* $F(ku^{T/2}) = i_1$ *and* $F(ku^{T/2} - 1) = i_2$
4) *Let* T *be even and assume* $F(k) = i \neq 0$ *and* $F(k+1) = j$*. Then, there exists* k' *such that* $F(k') = m - i$*, and* $F(k'+1) = j - i \mod m$

Proof: For part one, suppose $2^{-1} = 2^i u^j$, so $1 = 2^{i+1} u^j$. If one takes the T-th power of both sides to get $1 = 2^{T(i+1)}$. Notice, $m-1$ is a feasible value for i, because $\mathrm{ord}_{Tm+1}(2)$ divides Tm. By uniqueness, $i = m - 1$.

The rest are all immediate corollaries of $F(ab) = F(a) + F(b) \bmod m$ for every $a, b \in \mathrm{GF}(Tm+1)$.

For the third part of the theorem, pick k' so that $kk' = 1$, then

$$\begin{aligned} F(k'+1) &= F(k'+k'k) = F(k'(k+1)) \\ &= F(k') + F(k+1) = j - i \mod m. \end{aligned}$$

□

Remark 5: The third part of this theorem essentially means that if T is even, then the multiplication matrix is symmetric. This was already proven in [4] and [20], by showing that the GNB for T even is a dual basis.

The fourth part of this theorem indicates that a row $i \neq 0$ is a i-times right cyclic shifted version of row $m - i$. A similar result is shown in [8], but with only m odd (note that m is odd implies T to be even, but not the converse). Another proof of part four of this theorem can be also found in [22].

Lemma 6: Let $T = 4$. Then m is odd.

Proof: Assume $m = 2n$, $n \in \mathbb{N}$. By the definition of GNB, $\gcd(\frac{4(2n)}{k}, 2n) = 1$, where $k = \mathrm{ord}_{4m+1}(2)$. Hence, $k = 8n$ and 2 is a primitive root in $GF(4m+1)$. This implies that $4m + 1 \equiv 3, 5 \bmod 8$, yielding a contradiction. □

Lemma 7: Let $T = 4$. Then 2 is a primitive root in $GF(4m+1)$ and $2^{-1} \in \{2^{m-1}u, 2^{m-1}u^3\}$.

Proof: Since m is odd, $p \equiv 5 \bmod 8$ and the Legendre Symbol $\left(\frac{2}{p}\right)$ has a value of -1. Therefore $x^2 \equiv 2 \bmod p$ has no solutions. By the definition of GNB, $\gcd(\frac{4m}{\mathrm{ord}_p(2)}, m) = 1$, so $m \mid ord_p(2)$. Write $\mathrm{ord}_p(2) = km$, $k \in \{1, 2, 4\}$, and let g be a generator in $GF(p)$. Then $2^{km} = g^{p-1} = g^{4m} \bmod p$. Therefore, $2^k = g^4$. If k is 1 or 2, then $2 \equiv (g^2)^2$ or $2 \equiv g^2$ leading to a contradiction either way.

The rest is an immediate corollary. □

Theorem 8: Suppose $T = 4$. Then rows $\frac{m+1}{2}$ and $\frac{m-1}{2}$ each have at most 2 non-zero elements.

Proof: Suppose $u + 1 = 2^i u^j \bmod p$. Then $(u+1)^4 = u^4 + 4u^3 + 6u^2 + 4u + 1 = -4 = 2^{4i}$. Therefore $-1 = 2^{4i-2} \Rightarrow 1 = 2^{8i-4} \Rightarrow 4m \mid 8i - 4 \Rightarrow m \mid 2i - 1$. By the constraint on i, we have $m = 2i - 1 \Rightarrow i = \frac{m+1}{2}$. By part two of Theorem 4, we conclude that $F(u^3 + 1) = F(u+1) = \frac{m+1}{2}$.

Let $k_1 = \frac{u^3}{1+u}$ and $k_2 = k_1 u^3$. Then $k_1 + 1 = \frac{u^3+u+1}{u+1} = \frac{1}{u+1} = k_1 u$. Additionally, $k_2 + 1 = k_1 u^3 + 1 = u^2(k_1 u - 1) = k_1 u^2 = k_2 u^3$. This tell us that $F(k_1) = F(k_2) = F(k_1 + 1) = F(k_2 + 1)$. $F(k_1) = F(\frac{u^3}{1+u}) = F(\frac{1}{1+u}) = m - F(u+1) = \frac{m-1}{2}$. Therefore, row $\frac{m-1}{2}$ has at most 2 non-zero entries. By the third part of Theorem 4, row $\frac{m+1}{2}$ also only has 2 non-zero entries because it is a cyclic shifted version of row $\frac{m-1}{2}$. □

Theorem 9: For the multiplication matrix of $T = 4$, other than row 0 which has 1 non-zero element, and rows $\frac{m\pm1}{2}$ which have 2 non-zero elements, all other rows have 4 non-zero elements.

Proof: Ash, Blake, and Vanstone in [4] showed that the complexity of the multiplication matrix is $C_N = 4m - 7$. There are 7 "missing" ones. The first row has only 1 non-zero entry, so there are still 4 "missing" ones. By Theorem 8, we found that row $\frac{m-1}{2}$ and $\frac{m+1}{2}$ account for 2 "missing" ones each. All the 4 "missing" ones are accounted for; hence, all the other rows have exactly 4 non-zero elements. □

Remark 10: This theorem proves [8, Remark 3] which has been left as a conjecture. The fact that the multiplication matrix has 1 non-zero element in row 1, two rows with 2 non-zero elements and all other rows have 4 non-zero elements has already been proven in [22]. The exact rows with the 2 non-zero elements is a new result.

Proposition 11: For $T = 6$, row one of the multiplication matrix has at most 4 non-zero entries. Therefore, by the fourth part of Theorem 4, row $m - 1$ also has at most 4 non-zero entries.

Proof: First note that since $\mathrm{ord}(u) = 6$, it follows that $\mathrm{ord}(u^2) = 3$. This gives us the identity $u^2 + (u^2)^2 + (u^2)^3 = 0$, which is equivalent to $2u^4 + 2u^2 = 2u^3 = -2$. Therefore, $F(2u^2 + 1) = F(2u^4 + 1)$, and the rest follows. □

Proposition 12: Let $T = 6$, then the row $i = F(u+1)$ has at most 2 non-zero entries. Additionally, by the fourth part of Theorem 4, row $m - F(u+1)$ also has at most 2 non-zero entries.

Proof: First, note that we have $u(u^2-1)+1 = u^3 - u + 1 = u^4 + u^3 + 1 = u^4$. By the first part of Theorem 4, we can write $F(u+1) = F(u^5+1) = i$. By the symmetric property of the multiplication matrix one can get,

$$k_1 = u^3(u+1) = u^4 - 1 \quad \text{and } k_2 = u^3(u^5+1) = u^2 - 1$$
$$F(k_1) = F(k_2) = i \quad \text{and } F(k_1+1) = F(k_2+1) = 0.$$

Therefore, the row $F(u+1)$ has at most 4 non-zero entries.

We can write

$$\begin{aligned} k_1 u + 1 &= (u^4 - 1)u + 1 = u^3(u^2 + u) + 1 \\ &= (2u^3 + 1)(u^2 + u) + 1 = 2u^5 + 2u^4 \\ &\quad + u^2 + u + 1 = 2u^5 + 2u^4 + 2u \\ &= 2(1 + u^4) = 2u^5, \\ k_2 u^{-1} + 1 &= (u^2 - 1)u^5 + 1 = u^3(u^4 + u^5) + 1 \\ &= (2u^3 + 1)(u^4 + u^5) + 1 = 2u + 2u^2 \\ &\quad + (1 + u^4) + u^5 = 2(u + u^2 + u^5) \\ &= 2u \end{aligned}$$

We see that $k_1 u + 1 = 2u^5$, $k_2 u^{-1} + 1 = 2u$, which tells us $F(k_1 u + 1) = F(k_2 u^{-1} + 1) = 1$ and that row $i = F(u+1)$ must have at most 2 non-zero entries. □

Proposition 13: Let $T = 6$ and assume $u + 1 = 2^i u^j$ for some $0 \leq i < m$ and $0 \leq j < T$. Then row $i = F(\frac{k_1}{2}) = F(u+1) - 1$ has at most 4 non-zero entries. Additionally, by part four of Theorem 4, row $m - F(u+1) - 1$ also has at most 4 non-zero entries.

Proof: Let $k_1 = u^3(u+1) = u^4 - 1$ and $k_2 = u^3(u^5+1) = u^2 - 1$ as it was in the previous proof. By part one of Theorem 4, suppose $\frac{1}{2} = 2^{m-1}u^\alpha$ and

$$\frac{k_1}{2} + 1 = \frac{u^4 - 1}{2} + 1 = \frac{u^4 + 1}{2} = \frac{u^5}{2} = 2^{m-1}u^{\alpha+5}$$
$$\frac{k_2}{2} + 1 = \frac{u^2 - 1}{2} + 1 = \frac{u^2 + 1}{2} = \frac{u}{2} = 2^{m-1}u^{\alpha+1}.$$

Therefore, $F(\frac{k_1}{2}+1) = F(\frac{k_2}{2}+1) = m-1$, and the result follows. □

Theorem 14: For $T = 6$, all rows have exactly 6 non-zero entries, except:

$$\begin{cases} \text{Row } 0 & \text{with 1 non-zero} \\ \text{Row } 1 & \text{with 4 non-zeroes} \\ \text{Row } F(u+1)-1 & \text{with 4 non-zeroes} \\ \text{Row } F(u+1) & \text{with 2 non-zeroes} \\ \text{Row } m-F(u+1) & \text{with 2 non-zeroes} \\ \text{Row } m-F(u+1)+1 & \text{with 4 non-zeroes} \\ \text{Row } m-1 & \text{with 4 non-zeroes} \end{cases}$$

Proof: It is shown in [4] that for $T = 6$, the complexity is exactly $6m-21$. It is easy to check that $F(u+1) \neq \pm 1, \pm 2$. Therefore, none of the mentioned rows are the same. All the "missing" ones have now been accounted for and the proof is complete. □

Theorem 15: Let T be even, and suppose $F(u^i+1) = \gamma$ for some i and $1 \le i < \frac{T}{2}$. Then row γ and $m-\gamma$ have at most $T-2$ non-zero entries.

Proof: It is shown in Theorem 4 that $F(u^{T-i}+1) = F(u^i+1) = \gamma$. Suppose $k_1 = u^{T/2}(u^{T-i}) - 1$ and $k_2 = u^{T/2}(u^i) - 1$. By Theorem 4, we have $F(k_1) = F(k_2) = \gamma$ and $F(k_1+1) = F(k_2+1) = 0$. It is also obvious that $k_2 = k_1 u^i$. Then, one can write $k_1 = 2^\gamma u^\lambda$ and $k_2 = 2^\gamma u^{\lambda+i}$ and using part one of Theorem 4, write $\frac{1}{2} = 2^{m-1}u^\alpha$. Applying Theorem 4 to k_1 and k_2, we get $\phi_1 = 2^{m-\gamma}u^{\alpha-\lambda}$, and $\phi_2 = 2^{m-\gamma}u^{\alpha-\lambda-i}$. Then, $F(\phi_1) = F(\phi_2) = m-\gamma$, and $F(\phi_1+1) = F(\phi_2+1) = 0-\gamma = m-\gamma$. Apply part four of Theorem 4 and the result follows. □

Theorem 16: Suppose T is even and $F(u^i+1) = \gamma$ for some i and $1 \le i < \frac{T}{2}$. Then row $\gamma - 1$ has at most $T-2$ non-zero elements.

Proof: Then, one can construct $k_1 = u^{T/2}(u^{T-i}) - 1$ and $k_2 = u^{T/2}(u^i) - 1$ as described before. Then, $F(\frac{k_1}{2}+1) = F(\frac{k_2}{2}+1)$.

$$\begin{aligned} \frac{k_1}{2}+1 &= \frac{u^{T/2}u^{T-i}-1}{2}+1 = \frac{u^{T/2-i}+1}{2} \\ \frac{k_2}{2}+1 &= \frac{k_1u^i}{2}+1 = \frac{u^i(u^{T/2}u^{T-i}-1)}{2}+1 \\ &= \frac{u^{T/2}-u^i}{2}+1 = \frac{u^{i-T/2}+1}{2} \end{aligned}$$

It is easy to see that $u^{i-T/2}(\frac{k_1}{2}+1) = \frac{1+u^{i-T/2}}{2} = \frac{k_2}{2}+1$ which completes the proof. □

IV. Space-Complexity Reduction Algorithms

In this section, we present two complexity reduction algorithms based on the facts proven in the previous section. Before presenting our complexity reduction algorithms, we describe a new method for generating minimum lists of distinct distances for the multipliers discussed in Section 2. All notation used in this paper is summarized in Table 1.

TABLE I
Notations and Their Definitions

Notation	Definition
$\mathtt{D}(a,b)$	distance between a and b
$\mathtt{C}(a,b)$	center of the pair a and b
τ_i	list of centers of distance i
CD	consecutive distance
NV	neighbor value
ANV	advanced neighbor value
GN	gap number
$\mathtt{MP}_d$	minimum number of pairs required to span the ρ matrix given digit size d

A. Generating Minimum List of Distinct Distances

In this subsection, we first propose a new approach to achieve the minimum number of pairs to generate the ρ and Q modules discussed in Section 2. Since the dimension of the ρ module is smaller than the Q module, we first consider the problem of finding the minimal number of pairs in the ρ module. We note that almost all the components of the algorithm will be exactly the same when we extend it later to the Q module. The construction of the μ matrix requires m to be odd, which we assume in this subsection.

Definition 17: Let $a, b \in \mathbb{Z}_m$. We define the distance between a and b to be $\mathtt{D}(a,b) := \min\{a-b \bmod m, b-a \bmod m\}$. If $a-b < b-a \bmod m$, we say a and b have distance $a-b \bmod m$ centered at b; otherwise, they have distance $b-a \bmod m$ centered at a. Denote $C(a,b)$ as the center of the pair a, b. A pair a, b can be denoted as $\alpha; \beta$ where $\alpha = \mathtt{D}(a,b)$ and $\beta = C(a,b)$.

Definition 18: Let ψ be a module generated by the μ module where row i of ψ consists of every distinct separation of pairs made from entries in row i of μ. Suppose μ_i has entries $\{x_1,\ldots,x_{2n}\}$ for some $2n \le T$. Then ψ_{ij} is of the form

$$\mathtt{D}(y_1,y_2);\ C(y_1,y_2) \mid \cdots \mid \mathtt{D}(y_{2n-1},y_{2n});\ C(y_{2n-1},y_{2n})$$

where $\{x_1,\ldots,x_{2n}\} = \{y_1,\ldots,y_{2n}\}$. For a row of size T in the μ matrix, the corresponding row in the ψ module has length of $(T-1)(T-3)\cdots(1)$. The order in which the entries in a given row of ψ appear is irrelevant. We call every entry of ψ a choice of separation and any reduction of ψ that has only 1 choice of separation remaining in every row, a possible $\hat{\psi}$ module.

The problem of breaking down the ρ and Q matrices into pairs can effectively be reduced to choosing the best combination of separations in each row of ψ. By only using ψ in the algorithm, instead of the full ρ and Q matrices, the running time will be also reduced significantly.

Example 19: For a given $m = 7$ and $T = 4$, the ψ module can be built as follows:

$$\psi = \begin{pmatrix} 2;0 \mid 1;3 & 3;0 \mid 2;2 & 3;4 \mid 1;2 \\ 3;4 & - & - \\ 3;5 \mid 1;2 & 3;2 \mid 2;1 & 2;3 \mid 1;1 \end{pmatrix}.$$

A possible choice of a $\hat{\psi}$ is

$$\hat{\psi} = \begin{pmatrix} 2;0 \mid 1;3 \\ 3;4 \\ 3;2 \mid 2;1 \end{pmatrix}.$$

Definition 20: Let $\tau_i = \left\{k_0^{(i)}, \ldots, k_{|\tau_i|-1}^{(i)}\right\}$ indicate the list of centers (not necessarily distinct centers) of distance i in the ψ module. This list is ordered so $k_\alpha^{(i)} \leq k_\beta^{(i)}$, whenever $\alpha < \beta$. If the distance is understood to be a particular i, the superscript can be dropped.

Definition 21: For $k_j^{(i)} \in \tau_i$ and some digit-size d, we define consecutive distance (CD) as follows:

$$\mathtt{CD}(k_j^{(i)}) = \begin{cases} d & \text{if } |\tau_i| = 1 \\ \min(d,\, k_{j+1}^{(i)} - k_j^{(i)} \bmod m) & \text{otherwise} \end{cases}$$

where the subscripts are taken to be modulo $|\tau_i|$.

The concepts of neighbor value and advanced neighbor value will be used to construct weight functions for our approximation algorithm. When we cyclic shift the μ module to obtain the ρ module, the distance of a given pair is preserved while the center is decreased by 1. Suppose we break the μ matrix into pairs and store the pairs in any given way. Additionally, suppose we store the same corresponding pairs in every cyclic shifted version of μ. A reduction in storing pairs of a cyclic shifted version of μ will be possible if a pair was already stored in a less cyclic shifted version of μ. This will happen if two pairs have the same distance, and the difference between their centers is less than the number of times we cyclic shift (the d-value). More specifically, if a pair with distance i and center $k_j^{(i)}$, $k_{j+1}^{(i)}$ are stored, then $d - CD(k_j^{(i)})$ reductions will be possible. Consequently, the neighbor value will reflect how valuable (based on the number of pairs reduced) a given pair is. Why these functions yield good weight systems will be explained in further detail in Remark 28. In our approximation algorithm, the higher the weight, the higher the chances that a given pair will be deleted and resulting in a more successful subexpression elimination in the entire multiplication matrix.

Definition 22: For $k_j^{(i)} \in \tau_i$, we define neighbor value (NV) as:

$$\mathtt{NV}(k_j^{(i)}) = \begin{cases} d & \text{if } |\tau_i| = 1 \\ 0 & \text{if } |\{\alpha \in \tau_i : \alpha = k_j^{(i)}\}| > 1 \\ \mathtt{CD}(k_j^{(i)}) + \mathtt{CD}(k_{j-1 \bmod |\tau_i|}^{(i)}) \\ \quad - \min(d,\, k_{j+1 \bmod |\tau_i|}^{(i)} & \text{otherwise.} \\ \qquad - k_{j-1 \bmod |\tau_i|}^{(i)}) \end{cases}$$

Example 23: Consider a GNB over $GF(2^{15})$ with digit-size $d = 4$ and $T = 4$. Let $\tau_i = \{1, 1, 5, 6, 11\}$. By definition, the neighbor values are: $0, 0, 1, 1, 4$, respectively. It is easy to observe that the NV of a particular center is lower if its immediate neighbors are closer to it.

Definition 24: Suppose $|\tau_i| > 1$. For $k_j^{(i)} \in \tau_i$, let α be the next index obtained by adding 1 mod $|\tau_i|$ to j, with the property that $k_\alpha^{(i)}$ is either "in a different row of ψ than $k_j^{(i)}$" or "in the same row and same separation as $k_j^{(i)}$". Similarly, let β be the index obtained by subtracting 1 mod $|\tau_i|$ from j. If one set

$$x = k_\alpha^{(i)} - k_j^{(i)} \mod m$$
$$y = k_j^{(i)} - k_\beta^{(i)} \mod m.$$

We can define the advanced neighbor value (ANV) as follows:

$$\mathtt{ANV}(k_j^{(i)}) = \begin{cases} d & \text{if } |\tau_i| = 1 \\ d & \text{if } \alpha = j \text{ or } \beta = j \\ 0 & \text{if } k_j^{(i)} = k_\alpha^{(i)} \text{ or } k_j^{(i)} = k_\beta^{(i)} \\ \min(d,\, x) + \min(d,\, y) & \\ \quad - \min(d,\, x + y \bmod m) & \text{otherwise.} \end{cases}$$

This advanced weight will be the primary weight used as the idea is similar to the one used as of NV. For values of $T > 4$, the weight function based on the ANV yields better results than the more basic weight function based on the NV.

Example 25: Suppose $T = 6$, and a row of the μ module is $[1, 2, 5, 7, 10, 14]$. Notice that two possible ways of choosing pairs are: $x_1 = \{(1, 2), (5, 7), (10, 14)\}$ and $x_2 = \{(1, 2), (5, 10), (7, 14)\}$. Both of these separations have the pair $(1, 2)$. In the calculation of the consecutive distance value and the neighbor value, the weight of a given pair is determined by its immediate neighbours. Based on NV, the weight of both pairs will be zero. However, since the two pairs are in the same row, but different separations, and only one of these separations can be chosen, it is not feasible for the pairs to lower the weight of the other.

This example shows that in the calculation of the neighbour value, it is critical that the weight of a pair is calculated with a neighbour that is not in "the same row and different separations". The ANV works by finding the next immediate and the previous neighbors which are not in "the same row and different separations" to compute the weight.

Theorem 26: For given odd m, even T and digit-size d, $1 \leq d \leq m$, the number of distinct pairs in τ_i and their cyclic shifted copies in the ρ module is $\sum_{k_j \in \tau_i} \mathtt{CD}(k_j)$.

Proof: When $|\tau_i| = 1$, this is trivial. For $|\tau_i| = 2$, suppose $\tau_i = \{k_0, k_1\}$, $k_1 - k_0 = d_0$ and $k_0 + m - k_1 = d_1$. First, one needs to store k_0 and k_1. Then, after a left-cyclic shift of both, and storing any unstored values the process should be repeated. It is easy to see that after d_0 left-cyclic shifts of k_1, we obtain k_0. Therefore, all the subsequent shifts of k_0 do not need to be stored as they would have already been obtained from left-cyclic shifting of k_1. However, for $d_0 > d$, in which case, k_0 will never be obtained by cyclic-shifting of k_1. Therefore, $\min(d, k_1 - k_0)$ left-cyclic shifts of k_0 are required. Similar logic applies for k_1. We note that

$$\begin{aligned} \mathtt{CD}(k_1) &= \min(d,\, k_0 + m - k_1) \\ &\neq \min(d,\, k_0 - k_1 \bmod m), \end{aligned}$$

because one may have $k_0 = k_1$. In this case, by definition, $d_0 = 0$, $d_1 = d$, so d pairs are generated by τ_i. Using the

$\min(d, k_0 - k_1 \bmod m)$ definition will give zero pairs generated by τ_i. This argument generalizes to larger sizes of τ_i and proof completes. □

Theorem 26 is important as we use it to determine how many pairs a certain $\hat{\psi}$ module will generate (a $\hat{\psi}$ module is one with only one separation per column, as reduced from the ψ module). Based on this theorem, there is a well-defined and well-understood notion of which $\hat{\psi}$ module is better or inefficient. It is important to note that the number of pairs generated by a certain center set τ_i depends heavily on how close together the centers in the set are located.

Theorem 27: Let $|\tau_i| \neq 0$ for some i and $\alpha \in \{0, 1, \cdots, |\tau_i| - 1\}$. Define $\tau_i(\alpha)$ to be τ_i with $k_\alpha^{(i)}$ removed. Then the number of pairs generated by τ_i is exactly $\mathtt{NV}(k_\alpha)$ more then that generated by $\tau_i(\alpha)$. By Theorem 26, this is equivalent to saying

$$\sum_{k_j \in \tau_i} \mathtt{CD}_{\tau_i}(k_j) - \mathtt{NV}_{\tau_i}(k_\alpha) = \sum_{k_j \in \tau_i(\alpha)} \mathtt{CD}_{\tau_i(\alpha)}(k_j).$$

Proof: If $|\tau_i| = 1$ then this is trivial. If there exists some $\beta \in \{0, 1, 2, \cdots, |\tau_i| - 1\}$, $\beta \neq \alpha$ and $k_\alpha = k_\beta$, then the number of pairsets generated by τ_i and $\tau_i(\alpha)$ are equal, and indeed $\mathtt{NV}_{\tau_i}(k_\alpha) = 0$ by definition. Let $j \neq \alpha$, then $\mathtt{CD}_{\tau_i}(k_j) = \mathtt{CD}_{\tau_i(\alpha)}(k_j)$ if and only if $j \neq \alpha \pm 1 \bmod |\tau_i|$. Therefore,

$$\begin{aligned}\sum_{k_j \in \tau_i} \mathtt{CD}_{\tau_i}(k_j) - \sum_{k_\ell \in \tau_i(\alpha)} \mathtt{CD}_{\tau_i(\alpha)}(k_\ell) &= \mathtt{CD}_{\tau_i}(k_{\alpha-1}) + \mathtt{CD}_{\tau_i}(k_\alpha) - \mathtt{CD}_{\tau_i(\alpha)}(k_{\alpha-1}) \\ &= \mathtt{CD}_{\tau_i}(k_{\alpha-1}) + \mathtt{CD}_{\tau_i}(k_\alpha) - \min(d, k_{\alpha+1} - k_{\alpha-1}) \\ &= \mathtt{NV}_{\tau_i}(k_\alpha)\end{aligned}$$

and the proof is completed. □

Remark 28: $\mathtt{NV}$ and $\mathtt{ANV}$ will be used as weight systems in our approximation algorithm (Algorithm 2). The approximation algorithm will delete one separation in each iteration. Theorem 27 sets a well-defined way of determining which separation is best removed at each step. By deleting the pair with the highest NV, the resulting module will have the smallest number of pairs. However, a simple greedy algorithm will not work. Additionally, the weight of a given pair heavily depends on its "neighboring" centers in the distance set τ_i. This is made evident by the definition of $\mathtt{NV}$. $\mathtt{ANV}$ is the better version, since it factors in the possibility of pairs in the same row but with different separations lowering each other's weights.

Definition 29: Given T, m, and d, let $\mathtt{MP}_d$ be the optimal minimum number of pairs needed to generate the whole ρ module from an optimal $\hat{\psi}$ module.

Definition 30: Given d and a particular $\hat{\psi}$ module (where every row has exactly 1 separation remaining), we define gap number ($\mathtt{GN}$) as

$$\mathtt{GN}_d(\hat{\psi}) = \left|\{k_{j+1}^{(i)} - k_j^{(i)} > d : k_j^{(i)} \in \tau_i, \text{ for all } 1 \leq i \leq \frac{m-1}{2}\}\right|.$$

Notice that this number also corresponds to the difference of the number of pairs generated by any given $\hat{\psi}$ module from d to $d + 1$.

Theorem 31: Given T, m and $2 \leq d \leq m - 1$, we have $\mathtt{MP}_{d+1} - \mathtt{MP}_d \leq \mathtt{MP}_d - \mathtt{MP}_{d-1}$.

Proof: Let $\hat{\psi}_{d-1}$, $\hat{\psi}_d$ and $\hat{\psi}_{d+1}$ be optimal modules for digit sizes $d - 1$, d and $d + 1$, respectively. It is clear that $\mathtt{GN}_{d+1}(\hat{\psi}) \leq \mathtt{GN}_d(\hat{\psi})$ for any $\hat{\psi}$.

In order for $\hat{\psi}_d$ to be optimal for digit size d, the difference in the number of pairsets generated by $\hat{\psi}_d$ from $d - 1$ to d must be less than or equal to that generated by $\hat{\psi}_{d-1}$. Equivalently,

$$\begin{aligned}\mathtt{GN}_{d+1}(\hat{\psi}_{d+1}) &\leq \mathtt{GN}_{d+1}(\hat{\psi}_d) \\ &\leq \mathtt{GN}_d(\hat{\psi}_d) \leq \mathtt{GN}_d(\hat{\psi}_{d-1}).\end{aligned}$$

Additionally, since the change in optimal number of pairs can not be as great as the change in pairs of $\hat{\psi}_{d+1}$ and can at most be exactly equal if $\hat{\psi}_{d+1}$ is also optimal for digit size d, we get that

$$\begin{aligned}\mathtt{GN}_{d+1}(\hat{\psi}_{d+1}) &\leq \mathtt{MP}_{d+1} \\ -\mathtt{MP}_d &\leq \mathtt{GN}_{d+1}(\hat{\psi}_d).\end{aligned}$$

Similarly, we have

$$\begin{aligned}\mathtt{GN}_d(\hat{\psi}_d) &\leq \mathtt{MP}_d \\ -\mathtt{MP}_{d-1} &\leq \mathtt{GN}_d(\hat{\psi}_{d-1}).\end{aligned}$$

The result follows through the derived inequalities. □

B. Algorithms

1) Exponential Time Complexity Reduction Algorithm: We introduce two new algorithms for reducing the space complexity of GNB multiplication matrices. Both algorithms take in the same basic parameters, namely m and T, as well as the ψ module. From the formation of the ψ module, the problem of constructing the ρ module with the least number of pairs can be thought of as choosing the best combination of separations in every row of the ψ module. The goal of both algorithms is to reduce the number of separations, until every row of ψ has only one separation per row, which is called the $\hat{\psi}$ module.

An exponential time program can be formed by trying every possible combination, and computing the number of pairs generated by that combination as given by Theorem 26. This algorithm is presented in Algorithm 1. The approximation algorithm also needs to receive the digit-size d as part of its input, as it is necessary to apply Theorem 26. The index_list in the algorithm is an array whose i-th element indicates the length of row i in the ψ module. At each step, a different combination of separation in each row is used for calculation, and the index array is used to enumerate through all the possibilities. It starts out as an array of 0's, and increases at each step until it reaches the index_list. For each choice of separation, the number of pairs generated by that particular matrix is recorded according to Theorem 26. At the end, a matrix with the minimum number of pairs is outputted.

It is important to note that the exponential-time algorithm (Algorithm 1) produces the optimal minimum number of pairs when using the method of generating pairs based on the

Algorithm 1 Exponential-Time Complexity Reduction Algorithm

```
Input: $T$, $m$, $d$ and the $\psi$ module.
Output: A reduced module $\hat{\psi}$ that has only 1 separation per row.
1: opt_psi = [ ]
2: opt_val = $\infty$
3: index_list= $\left[\text{length}(\psi_i) : \text{ for } 0 \le i < \text{length}(\psi)\right]$
4: index= [0] $*$ length($\psi$)
5: $\hat{\psi}_i = \psi_{i0}$ for $0 \le i <$ length($\psi$)
6: while index$\neq$ index_list do
7:   Generate $\tau_i$
8:   pairs= $\sum_{1\le i\le\frac{m-1}{2}}\sum_{k_j^{(i)}\in\tau_i}\text{CD}(k_j^{(i)})$
9:   $k = 0$
10:  if pairs $\le$ opt_val then
11:    opt_val = pairs
12:    opt_psi = $\hat{\psi}$
13:  end if
14:  while $k \le$ length($\psi$) do
15:    index[$k$] = index[$k$] + 1
16:    if index[$k$] $\le$ index_list[$k$] then
17:      $\hat{\psi}_k = \psi_{k,\text{index[k]}}$
18:      break
19:    else
20:      index[$k$] = 0
21:      $\hat{\psi}_k = \psi_{k,\text{index[k]}}$
22:      $k = k + 1$
23:    end if
24:  end while
25: end while
26: Return opt_psi
```

module μ with depth 1. This algorithm has a run time of $O\left([(T-1)(T-3)\dots(1)]^{\frac{m-1}{2}}\right)$. We observe that the time it takes for this algorithm to finish grows very fast. At first glance, it may seem as though one does not need to worry about its efficiency, since it only needs to be computed once off-line; however, it is likely that due to the time constraints, it can not even be computed once (please refer to Fig. 1). For instance, the exponential time algorithm takes about 3,840 seconds to compute $T = 4$, $m = 37$ and as the field size increases the time of computation grows exponentially. In the following subsection, we present an approximation algorithm suitable for our implementations.

2) Polynomial Time Complexity Reduction Algorithm: The goal of the approximation algorithm (Algorithm 2) is to remove a separation from the ψ module at every step in such a way that produces a result close to that produced by Algorithm 1. The algorithm goes through every row of ζ, which is initialized as a copy of ψ. In every row, the weight function is used to give each separation a weight. The maximum and minimum weight achieved are stored. If the difference between the maximum and the minimum weight is the highest found thus far, any separation that achieves the maximum weight in that given row is subjected to possible deletion. After iterating through every row,

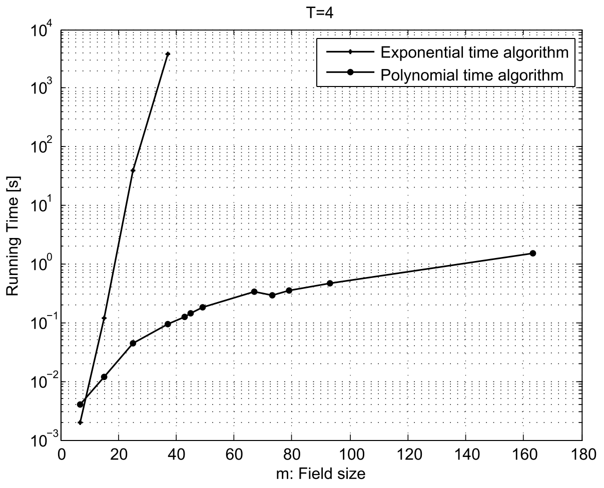

Fig. 1. Comparison of running times of exponential time (Algorithm 1) and polynomial time (Algorithm 2) algorithms for type 4 GNB and different field sizes.

the separation in the row with the highest weight disparity, and with the maximum weight in the row, is deleted. This process occurs in lines 7 to 16. The disparity is used because it is logical to delete the separation which is "the least likely to be chosen" of all the separations in the same row. The deletion continues until every row of ζ has only one separation remaining.

This algorithm does not take in any specific digit-size d. A module $\hat{\psi}$ is generated for each possible value of d, and the resulting matrices are stored in a list (lines 19 to 21). For a given digit size, the matrices in this list are compared using Theorem 26 (lines 23 to 34), and the best choice is returned. This method ensures that the values outputted from this approximation algorithm satisfy Theorem 31.

Let a separation of the ψ module be labeled as δ_1; c_1 | … | δ_n; c_n. As before, δ_i are distances, and c_i are centers. Let same_row($k_j^{(i)}$) indicate the number of centers in τ_i that are in the same row, but have different separation than τ_i. Let $\sigma_i(k_j^{(i)})$ indicate the sum of the maximum number of centers of distance i in each row other than the row $k_j^{(i)}$ is in, plus the number of centers of distance i in the same separation as $k_j^{(i)}$. In the following a list of weight functions that have been tried, and their results are given. We should note that there are exceptions to the observed patterns.

1) $\sum_{i=1}^{n}\left(\text{NV}(c_i) + \dfrac{d}{|\tau_{\delta_i}|}\right)$

2) $\sum_{i=1}^{n}\left(\text{ANV}(c_i) + \dfrac{d}{|\tau_{\delta_i}| - \text{same\_row}(c_i)}\right)$

3) $\left[\sum_{i=1}^{n}\left(\text{ANV}(c_i) + \dfrac{d}{|\tau_{\delta_i}| - \text{same\_row}(c_i)}\right)\right]/n$

4) $\sum_{i=1}^{n}\left(\text{ANV}(c_i) + \dfrac{d}{\sigma_{\delta_i}(c_i)}\right)$

5) $\left[\sum_{i=1}^{n}\left(\text{ANV}(c_i) + \dfrac{d}{\sigma_{\delta_i}(c_i)}\right)\right]/n$

The first weight function is the standard weight system. The NV gives an interpretation of how much better the reduced ψ module will be if this particular separation is deleted. However, this weight function only works well when $T = 4$.

Algorithm 2 Polynomial Time Complexity Reduction Algorithm With Deleting Separations

```
Input: $T$, $m$, a weight function $f$ and the $\psi$ module.
Output: A reduced module, $\hat{\psi}$ that has only 1 separation per row for $d$, $1 \le d \le m$.
opt = $empty$
for $1 \le d \le m$ do
    $\zeta = \psi$
    while $\exists\, i$ such that $|\zeta_i| > 1$ do
        disparity = 0
        cord = $[0, 0]$
        for all $i$ such that $|\zeta_i| > 1$ do
            $\alpha = \max_{j \in |\zeta_i|} f(\zeta_{ij}, d)$
            $\beta =$ any $k$ where $f(\zeta_{ik}, d) = \alpha$
            $\gamma = \min_{h \in |\zeta_i|} f(\zeta_{ih}, d)$
            diff = $\alpha - \gamma$
            if diff > disparity then
                disparity = diff
                cord = $[i, \beta]$
            end if
        end for
        delete $\zeta_{i\beta}$
    end while
    if $\zeta \notin$ opt then
        opt = opt $\cup \{\zeta\}$
    end if
end for
for $1 \le d \le m$ do
    min = $\infty$
    $\hat{\psi}_d = [\,]$
    for all $\zeta \in$ opt do
        Generate $\tau_i$ for all feasible $i$ based on $\zeta$
        $\alpha = \sum_{i=1}^{\frac{m-1}{2}} \sum_{k_j^{(i)} \in \tau_i} \mathrm{CD}(k_j^{(i)})$
        if $\alpha <$ min then
            min = $\alpha$
            $\hat{\psi}_d = \zeta$
        end if
    end for
    Return $\hat{\psi}_d$
end for
```

It should be noted that for a particular distance i, regardless of the size of τ_i or the d value, if there is a pair in the final $\hat{\psi}$ module with distance i, then at least d pairs are required. It makes sense to distribute this base amount d amongst all pairs of distance i, thus the term $\frac{d}{|\tau_{\delta_i}|}$. Example 25 illustrates a situation where it is vital that ANV is used instead of NV, for $T > 4$. By the same logic, it make sense to use $\frac{d}{|\tau_{\delta_i}| - \mathrm{same\_row}(c_i)}$ instead of $\frac{d}{|\tau_{\delta_i}|}$. Additionally, it may make even more sense to use $\frac{d}{\sigma_{\delta_i}(c_i)}$, because in other rows, centers of distance δ_i may be in every separation of that row, despite the fact that only one of them may be chosen. However, through computation with different m, T and d values, we find that one is better than the other in some cases and worse in others. There does not seem to be a pattern in predicting which is better and when. Therefore, predicting which score is better, is the subject for more study. For larger T values, there are many rows of varying lengths. The rows with more entries have separations that, in general, have higher weight values than separations in rows of smaller size. Therefore, it makes sense to balance the weight by dividing by the number of pairs in a particular separation. This makes no difference for $T = 4$, yields varying results for $T = 6$, and generally gives better results for $T = 10$.

For the comparison purpose, we plot the running times of both Algorithms 1 and 2 in terms of different field sizes for type 4 GNB in Fig 1. As one can see, the running time of exponential-time algorithm grows very fast and for larger field sizes it is infeasible to employ this algorithm for the complexity reduction purpose (even for $m = 55$). For instance, Algorithm 2 takes only about 1.56 seconds for the field size of $m = 163$ and it may never end if we run Algorithm 1 in a general PC. Therefore, one definitely needs to employ Algorithm 2 instead of Algorithm 1 for the space-complexity reduction discussed in this paper. In the following, we provide examples about the polynomial time algorithm.

Example 32: Suppose $m = 7$, $T = 4$ and $d = 3$. We show an example where weight function number 1) is used. Breaking the ψ module into weights gives

$$\begin{pmatrix} \frac{3}{4} + 1 + \frac{1}{4} & 1 + \frac{1}{5} + \frac{1}{4} & \frac{1}{5} + \frac{1}{4} \\ \frac{1}{5} & - & - \\ \frac{1}{5} + \frac{1}{4} & 1 + \frac{1}{5} + \frac{1}{4} & 1 + \frac{1}{4} + 1 + \frac{1}{4} \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 1 + \frac{9}{20} & \frac{9}{20} \\ \frac{1}{5} & - & - \\ \frac{9}{20} & 1 + \frac{9}{20} & 2 + \frac{1}{2} \end{pmatrix}.$$

Notice the entries with the highest weight in rows with more than one separation are the top-left most, and the bottom-right most entries. The bottom-right most entry has a larger difference in weight in comparison to the lowest weighted entry in that row. Therefore, the bottom right most entry will be deleted. A one-step-reduced ψ module is now obtained.

$$\begin{pmatrix} 2;0 \mid 1;3 & 3;0 \mid 2;2 & 3;4 \mid 1;2 \\ 3;4 & - & - \\ 3;5 \mid 1;2 & 3;2 \mid 2;1 & - \end{pmatrix}.$$

New τ_i's will be generated and the process repeated. At the end, a fully reduced $\hat{\psi}$ module will be outputted as:

$$\hat{\psi} = \begin{pmatrix} 3;4 \mid 1;2 \\ 3;4 \\ 3;5 \mid 1;2 \end{pmatrix}.$$

From Algorithm 1 and Algorithm 2, the ψ module is reduced, so that only one separation remains in every row, and outputted as the $\hat{\psi}$ module. Algorithm 3 will be used to generate the pairset when given a particular $\hat{\psi}$ module. Pairs of different distances are generated separately. Starting from the smallest

Algorithm 3 Generating Pairsets

```
Input: A simplified $\hat{\psi}$ module, $m$ and an integer $1 \leq d \leq m$.
Output: A set of pairs.
1: pairset = [ ]
2: for $1 \leq i \leq \frac{m-1}{2}$ and $|\tau_i| \neq 0$ do
3:   $\delta = |\tau_i|$
4:   start $= 0$
5:   while start $\neq \delta$ do
6:     if start $= 0$ then
7:       $\alpha = k_0^{(i)} - k_{\delta-1}^{(i)} + m$
8:     else
9:       $\alpha = k_{\text{start}}^{(i)} - k_{\text{start}-1}^{(i)}$
10:    end if
11:    $\beta = 0$
12:    while $\beta \neq \min(\alpha, d)$ do
13:      pairset = pairset $\cup$ $\left(k_{\text{start}}^{(i)} - \beta, k_{\text{start}}^{(i)} - \beta + i\right)$ mod $m$
14:      $\beta = \beta + 1$
15:    end while
16:    start = start + 1
17:  end while
18: end for
19: Return pairset
```

center of some distance i, the pair corresponding to the center, and its left-cyclic shift copies will be stored. This is done up to d times, or up to reaching the subsequent center. This process appears in lines 12 to 14. Upon reaching d, or the next center, the process repeats for the subsequent center and its corresponding pair. This is done for all distances. All pairs are stored during the process and outputted at the end. For DL-SIPO, a similar algorithm can be used by right-cyclic shifting every center, until the center becomes the next center in τ_i or at most d (reversed).

In the following examples, we show how the above algorithm works in reducing complexity and generating optimal pairsets to construct the ρ block of the multiplication module.

Example 33: Let $m = 7$, $T = 4$ and $d = 3$. From before, a possible $\hat{\psi}$ module produced is:

$$\hat{\psi} = \begin{pmatrix} 3;4 \mid 1;2 \\ 3;4 \\ 3;5 \mid 1;2 \end{pmatrix}.$$

Observe that $\tau_3 = \{4, 4, 5\}$. Starting from the first 4, Algorithm 3 will have to cyclic-shift 6 times to get from the first 4 to the previous center, 5. Therefore, only $d = 3$ left-cyclic shifts will happen giving the pairs $\{(4, 0), (3, 6), (2, 5)\}$. Next, it takes 0 shifts to get from the second 4 to the first 4, so no cyclic-shifts will happen. Finally, it takes 1 shift to get from 5 to the previous center which was 4, so one cyclic-shifted pair will be added. Hence, the final pairset generated for τ_3 is $\{(4, 0), (3, 6), (2, 5), (5, 1)\}$. The final pairset generated for the entire module will then be:

$$\{(4, 0), (3, 6), (2, 5), (5, 1), (2, 3), (1, 2), (0, 1)\}.$$

TABLE II

SPACE COMPLEXITY (NUMBER OF XORS) COMPARISON AMONG BIT-LEVEL NORMAL BASIS MULTIPLIERS FOR THE BINARY FIELDS RECOMMENDED BY NIST FOR ECDSA. IT IS NOTED THAT THE TIME COMPLEXITY REMAINED UNCHANGED IN COMPARISON TO THE LEADING ONES AND HENCE WE DO NOT INCLUDE THAT IN THIS TABLE

m, T	Architec.	Multiplier	# XORs
163,4	BL-PISO	MO [5], IMO [24]	644
		RM [8]	648
		This work	515
	BL-SIPO	BG [7]	645
		A-RM [11]	628
		This work	516
	BL-PIPO	GG [6]	404
		RM [8], A-RM [10]	404
		This Work	401
283,6	BL-PISO	MO [5], IMO [24]	1676
		RM [8]	1692
		This work	1403
	BL-SIPO	BG [7]	1629
		A-RM [11]	1512
		This work	1404
	BL-PIPO	GG [6]	980
		RM [8], A-RM [10]	964
		This Work	817
409,4	BL-PISO	MO [5], IMO [24]	1628
		RM [8]	1632
		This work	1267
	BL-SIPO	BG [7]	1629
		A-RM [11]	–
		This work	1268
	BL-PIPO	GG [6]	1019
		RM [8], A-RM [10]	1019
		This Work	1016

Example 34: Let $m = 7$, $T = 4$ and $d = 7$ (bit-parallel structure). The corresponding $\hat{\psi}$ module generated by our approximation algorithm is:

$$\hat{\psi} = \begin{pmatrix} 3;4 \mid 1;2 \\ 3;4 \\ 3;5 \mid 1;2 \end{pmatrix}.$$

From the fact that $\hat{\psi}$ has only 2 distinct distances, the pairset produced following the approximation algorithm will have size $2 \times 7 = 14$ (all cyclic-shifts of pairs with distance 1 and 3). Therefore, the pairs are as follows:

$$\left\{ \begin{matrix} (0,3) & (1,4) & (2,5) & (3,6) & (4,0) & (5,1) & (6,2) \\ (0,1) & (1,2) & (2,3) & (3,4) & (4,5) & (5,6) & (6,0) \end{matrix} \right\},$$

and one needs these 14 pairs to construct the ρ module of a DL-PIPO multiplier. It is worth mentioning that the complexity reduction algorithm presented in [10, Sec. 3.2] yields a pairset of size 18, compared to which the algorithm we presented is more efficient. In the following section we examine the performance of our algorithms for larger field sizes.

V. EXPERIMENTAL RESULTS AND COMPARISONS

In this section, we evaluate the performance of the complexity reduction algorithm we presented over the NIST recommended fields, i.e., $GF(2^m)$, where $m = 163$, 283, and 409, for ECDSA and compare our results

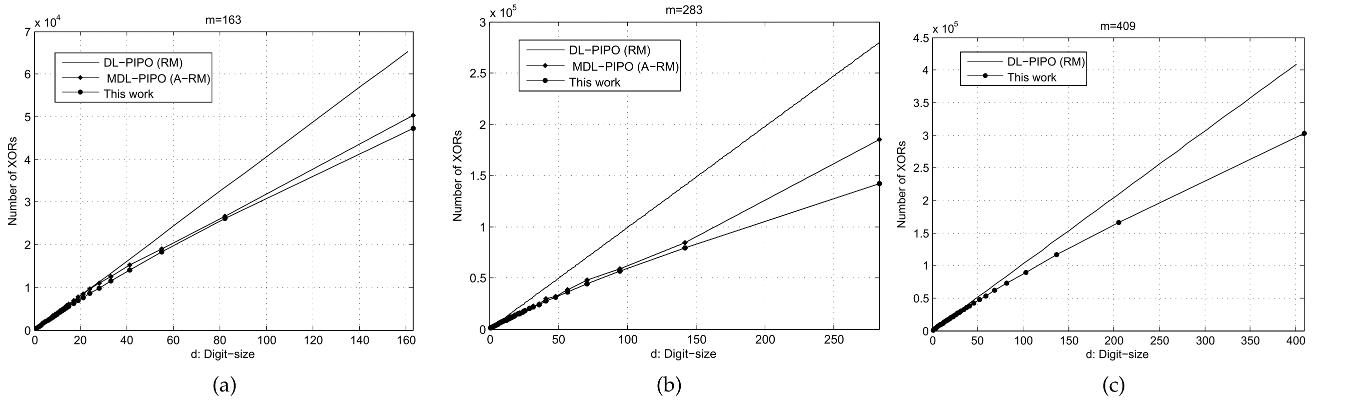

Fig. 2. Comparison of the number of XOR gates required in DL-PIPO multiplier architectures for (a): $m = 163$ ($T = 4$), (b): $m = 283$ ($T = 6$) and (c): $m = 409$ ($T = 4$). DL-PIPO (RM) [8], MDL-PIPO (A-RM) [10].

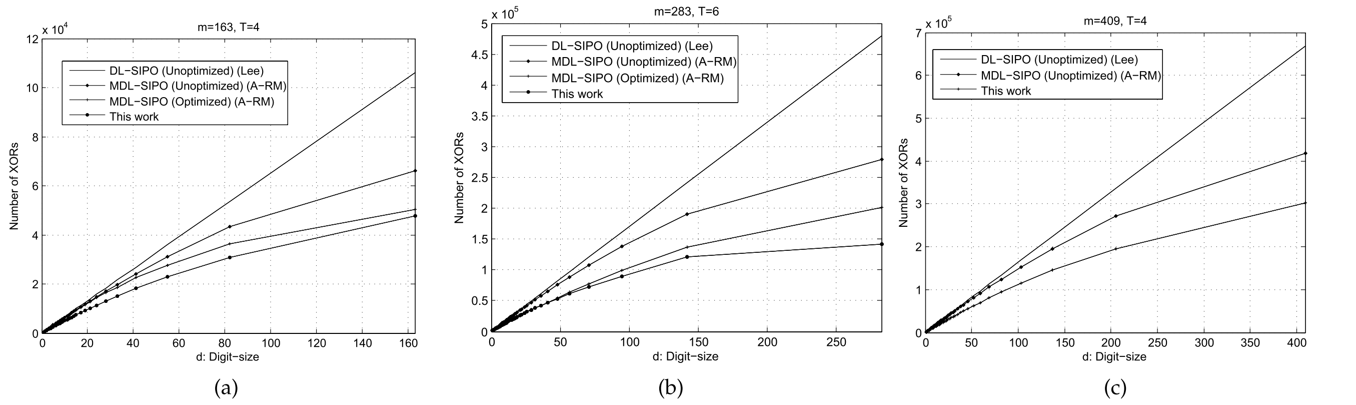

Fig. 3. Comparison of the number of XOR gates required in DL-SIPO multiplier architectures for (a): $m = 163$ ($T = 4$), (b): $m = 283$ ($T = 6$) and (c): $m = 409$ ($T = 4$). DL-SIPO (Lee) [23], MDL-SIPO (A-RM) [11].

to the ones presented in [10] and [11], for DL-PIPO and DL-SIPO multiplier architectures, respectively. Our algorithms work on any binary field with a GNB, but we limit our treatment to the NIST recommended fields for concreteness. We used Sage to run Python code for our approximation algorithm.

For the purpose of illustrating the efficiency of our algorithms in the case of bit-level multipliers, Table 2 shows the space complexity (number of XORs) of the available bit-level normal basis multipliers for the three binary fields. As one can see, for all bit-level multiplier architectures available, our algorithm always yields multipliers having fewer XORs in comparison to the counterparts. For instance, for $m = 283$ ($T = 6$ GNB) with BL-PIPO architecture our algorithm yields a multiplier of 817 XORs which is about 16% less than previous works ([8] and [10]). It is worth mentioning that the time complexity remained unchanged while the space complexity is reduced. This improvement is attractive for applications where the value of m is large but space is of concern, e.g., resource-constrained cryptographic systems on smart cards and RFIDs.

Our experimental results are illustrated in Figs. 2 and 3, for DL-PIPO and DL-SIPO architectures, respectively. We plot the required number of XOR gates in terms of the digit-size d for both multiplier architectures and counterparts. As one can see, our complexity reduction algorithm yields multipliers requiring fewer XORs in comparison to the counterparts. For instance, the unoptimized [8] and optimized [10] DL-PIPO multiplier architectures over $GF(2^{163})$ with digit-sizes of $d = 163$ require 65,852 and 50,400 XORs, while our algorithm yields 47,270 XORs. Our algorithm works better even for larger field sizes and types such as $m = 283$ which is of type $T = 6$. In comparison to the scheme presented in [10], our algorithm requires 24% fewer XOR gates for a DL-PIPO multiplier. Our approximation algorithm does not require the whole ρ module to generate the pairset; therefore, it is scalable to higher m values, effectively. We examine our algorithm for $m = 409$ ($T = 4$) and as one can see in Fig. 2 it requires 27% fewer XOR gates for the DL-PIPO architecture. It is worth mentioning that our algorithm runs in polynomial time and is more efficient than that of [10], being able to return results for large values of m. It is worth mentioning that for the filed size $m = 283$ with type $T = 6$ does not have very low complexity. Therefore, depending on the applications, it is more efficient to employ polynomial basis multiplier for this field if the overall computations of the design outperforms while employing GNB.

For DL-SIPO multiplier architectures, we obtained better results as one needs to implement the entire multiplication module (i.e., $\mathbf{R}_{(m-1)\times T}$) for unoptimized architectures. In comparison to the complexity reduction algorithm proposed for DL-SIPO multipliers in [11], our approximation algorithm yields better results for all digit sizes as one can see in Figs. 3. For instance, for $m = 163$ our algorithm yields 18% and 17% fewer XOR gates for $d = 1$ (bit-level architecture) and $d = 55$

(the most efficient digit size for ECC over GNB [9] and [12]) in comparison to the algorithm presented in [11]. Therefore, our algorithm is not only suitable for high-performance applications which require larger digit sizes but also provides efficient area complexity for resource-constrained applications with smaller digit sizes.

We further note that the space complexity reduction algorithms presented in this paper do not increase the time complexity of the multiplication architectures. For type T GNB over $GF(2^m)$ the time-complexity or critical-path delay of the multipliers depends to the architecture of the multiplier which can be bit-level ($d = 1$), digit-level ($1 < d < m$), and bit-parallel ($d = m$). The time-complexity of the architecture devised based on the presented complexity reduction algorithms for digit-level PIPO multiplier is the same as the original multiplier (without having its space complexity reduced [8]). The space-complexity reduction method proposed for the multiplier architecture decomposes the ρ module into two different modules: one includes pairs and the other one includes XORs to construct ρ module) with the delay of T_X and $\lceil \log_2 \frac{T}{2} \rceil T_X$, respectively ($T_X$ denote the delay of an XOR gate). The delay of J blocks is the delay of an AND gate denoted by T_A and the final $GF(2^m)$ adder has a delay of $\lceil \log_2(d+1) \rceil T_X$. Then, the total critical-path delay due to delays through mentioned logic gates is $T_X + \lceil \log_2 \frac{T}{2} \rceil T_X + T_A + \lceil \log_2(d+1) \rceil T_X = T_A + \left(\lceil \log_2 T \rceil + \lceil \log_2(d+1) \rceil\right) T_X$ which is the same as the original multiplier given in [8]. Note that same analysis is true for DL-PISO and DL-SIPO multiplier architectures.

VI. Conclusion

In this paper, we have presented new methods for space-complexity reduction of Gaussian normal basis multiplication. Low space-complexity multiplication is extensively needed in emerging embedded and high-performance cryptosystems. Our space-complexity reduction algorithm is based on generating minimum lists of distinct distances for Gaussian normal basis multiplication matrices for binary extension fields. Our presented space-complexity reduction algorithm outperforms the counterparts available in the literature. For instance, we have compared the number of XORs of our algorithm and the counterparts and a reduction of 15%–30% is obtained based on the architecture of the multiplier. It has been shown that the space-complexity reduction algorithm presented in this paper does not change the time-complexity of the multiplier.

Acknowledgment

The authors would like to thank the reviewers for their constructive comments.

References

[1] U.S. Department of Commerce/NIST, *Digital Signature Standard*, document FIPS 186-2, Jan. 2000.

[2] *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Standard 1363-2000, Jan. 2000.

[3] M. Christopoulou, T. Garefalakis, D. Panario, and D. Thomson, "The trace of an optimal normal element and low complexity normal bases," *Designs, Codes, Cryptogr.*, vol. 49, nos. 1–3, pp. 199–215, 2008.

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

[5] J. L. Massey and J. K. Omura, "Computational method and apparatus for finite field arithmetic," U.S. Patent 4 587 627, May 6, 1986.

[6] W. Geiselmann and D. Gollmann, "Symmetry and duality in normal basis multiplication," in *Proc. 6th Symp. Appl. Algebra, Algebraic Algorithms, Error-Correcting Codes (AAECC)*, Jul. 1989, pp. 230–238.

[7] T. Beth and D. Gollman, "Algorithm engineering for public key algorithms," *IEEE J. Sel. Areas Commun.*, vol. 7, no. 4, pp. 458–466, May 1989.

[8] A. Reyhani-Masoleh, "Efficient algorithms and architectures for field multiplication using Gaussian normal bases," *IEEE Trans. Comput.*, vol. 55, no. 1, pp. 34–47, Jan. 2006.

[9] C. H. Kim, S. Kwon, and C. P. Hong, "FPGA implementation of high performance elliptic curve cryptographic processor over $GF(2^{163})$," *J. Syst. Archit.*, vol. 54, no. 10, pp. 893–900, 2008.

[10] R. Azarderakhsh and A. Reyhani-Masoleh, "A modified low complexity digit-level Gaussian normal basis multiplier," in *Proc. 3rd Int. Workshop Arithmetic Finite Fields (WAIFI)*, vol. 6087. Jun. 2010, pp. 25–40.

[11] R. Azarderakhsh and A. Reyhani-Masoleh, "Low-complexity multiplier architectures for single and hybrid-double multiplications in Gaussian normal bases," *IEEE Trans. Comput.*, vol. 62, no. 4, pp. 744–757, Apr. 2013.

[12] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1453–1466, Aug. 2012.

[13] R. Azarderakhsh, M. Mozaffari-Kermani, and K. Jarvinen, "Secure and efficient architectures for single exponentiations in finite fields suitable for high-performance cryptographic applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 3, pp. 332–340, Mar. 2015.

[14] R. Azarderakhsh and A. Reyhani-Masoleh, "Parallel and high-speed computations of elliptic curve cryptography using hybrid-double multipliers," *IEEE Trans. Parallel Distrib. Syst.*, to be published.

[15] Communications Security Establishment Canada, *CSEC Approved Cryptographic Algorithms for the Protection of Sensitive Information and for Electronic Authentication and Authorization Applications Within GC*, document ITSA-11E, Mar. 2011. [Online]. Available: http://www.cse-cst.gc.ca/its-sti/publications/itsa-asti/itsa11e-eng.html

[16] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery, "On the security of 1024-bit RSA and 160-bit elliptic curve cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2009, p. 389, Sep. 2009.

[17] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1997.

[18] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, 2001.

[19] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Norwell, MA, USA: Kluwer, 1993.

[20] C. C. Wang, "An algorithm to design finite field multipliers using a self-dual normal basis," *IEEE Trans. Comput.*, vol. 38, no. 10, pp. 1457–1460, Oct. 1989.

[21] A. Reyhani-Masoleh and M. A. Hasan, "Fast normal basis multiplication using general purpose processors," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1379–1390, Nov. 2003.

[22] M. Christopoulou, T. Garefalakis, D. Panario, and D. Thomson, "Gauss periods as constructions of low complexity normal bases," *Designs, Codes, Cryptogr.*, vol. 62, no. 1, pp. 43–62, 2012.

[23] C.-Y. Lee, "Concurrent error detection architectures for Gaussian normal basis multiplication over $GF(2^m)$," *Integr., VLSI J.*, vol. 43, no. 1, pp. 113–123, 2010.

[24] L. Gao and G. E. Sobelman, "Improved VLSI designs for multiplication and inversion in $GF(2^m)$ over normal bases," in *Proc. 13th Annu. IEEE Int. ASIC/SOC Conf.*, 2000, pp. 97–101.

Reza Azarderakhsh received Ph.D. degree in electrical and computer engineering from the University of Western Ontario in 2011. He was a recipient of the NSERC Post-Doctoral Research Fellowship working in the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo. Currently, he is an assistant professor at the Department of Computer Engineering at Rochester Institute of Technology. His current research interests include finite field and its application, elliptic curve cryptography, pairing based cryptography, and post-quantum cryptography.

David Jao received his Ph.D in Mathematics from Harvard University in 2003. From 2003 to 2006, he worked in the Cryptography and Anti-Piracy Group at Microsoft Research, contributing cryptographic software modules for several Microsoft products. He is currently an associate professor in the Mathematics Faculty at the University of Waterloo, and the director of the Centre for Applied Cryptographic Research. His research interests include elliptic curve cryptography, protocol design, and implementation, and post-quantum cryptography.

Hao Lee is an undergraduate student at the University of Waterloo, majoring in Pure Mathematics and Combinatorics and Optimization. He is set to graduate in April of 2015. He has been an Undergraduate Research Assistant in both the Department of Combinatorics and Optimization and the Department of Pure Mathematics. His current research interests include Algebraic Geometry, Algebraic Number Theory and Cryptography.