Efficient Algorithm and Architecture for Elliptic Curve Cryptography for Extremely Constrained Secure Applications

Reza Azarderakhsh, Kimmo U. Järvinen, and Mehran Mozaffari-Kermani, *Member, IEEE*

***Abstract*—Recently, considerable research has been performed in cryptography and security to optimize the area, power, timing, and energy needed for the point multiplication operations over binary elliptic curves. In this paper, we propose an efficient implementation of point multiplication on Koblitz curves targeting extremely-constrained, secure applications. We utilize the Gaussian normal basis (GNB) representation of field elements over $GF(2^m)$ and employ an efficient bit-level GNB multiplier. One advantage of this GNB multiplier is that we are able to reduce the hardware complexity through sharing the addition/accumulation with other field additions. We utilized the special property of normal basis representation and squarings are implemented very efficiently by only rewiring in hardware. We introduce a new technique for point addition in affine coordinate which requires fewer registers. Based on this technique, we propose an extremely small processor architecture for point multiplication. Through application-specific integrated circuit (ASIC) implementations, we evaluate the area, performance, and energy consumption of the proposed crypto-processor. Utilizing two different working frequencies, it is shown that the proposed architecture reaches better results compared to the previous works, making it suitable for extremely-constrained, secure environments.**

***Index Terms*—Crypto-processor, Gaussian normal basis (GNB), Koblitz curves, point multiplication, RFID, security, wireless sensor networks.**

I. Introduction

NOWADAYS, Radio Frequency IDentification (RFID) technology surrounds us in several forms. RFID typically refers to wireless single chip, passive or active transponders operating at frequencies from 120 kHz (low-frequency) to 10 GHz (microwave, semi-active or active tags). They are widely used for several applications such as access control, identification control, Smartdust (for massively distributed sensor networks), and other closed system security-constrained applications. Moreover, security in wireless sensor networks is very important because sensor nodes are often deployed in security-constrained environments and are potentially exposed to hostile intruders. If even one node is captured by malicious attackers, the overall impact can be extremely harmful. These two instances are among extremely-constrained applications requiring efficient and low-energy implementations of security mechanisms.

Lately, it has been suggested that symmetric key cryptography might not be preferable for security-constrained applications in which there is a risk involved in storing symmetric keys, e.g., on an RFID tag or a sensor node [1]. This is in addition to the side-channel information attacks and the overheads for their countermeasures for block ciphers [2], [3]. Moreover, until recently, public key cryptography has been considered to be infeasible due to the high overhead it adds to the constrained devices. For instance, passive RFID tags are not equipped with batteries and due to this constraint, they cannot embed energy-demanding cryptographic algorithms. Nonetheless, elliptic curve cryptography (ECC) (as a public key cryptography scheme) has been employed recently in several applications due to its advantages over traditional schemes. The main advantage of ECC is that it offers similar security levels as other approaches but it employs smaller key sizes.

The security of ECC relies on the difficulty of solving the elliptic curve discrete logarithm problem (ECDLP) [4]. The main arithmetic computation of ECC is an operation denoted as point multiplication. The performance of point multiplication is determined by finite field arithmetic computations such as addition, squaring, inversion, and multiplication. Binary fields of characteristic two, i.e., $GF(2^m)$, are the most efficient fields for hardware implementations. Field elements can be represented using polynomial basis or normal basis with m bits. There are several implementations of point multiplication over binary fields in the literature targeting resource-constrained applications. For instance, one can refer to [5]–[9] to name a few, covering a wide variety of cases including different curve forms, e.g., generic and Edwards, and different coordinate systems, e.g., affine, projective, and mixed. Field elements have been represented using polynomial basis and application-specific integrated circuit (ASIC) hardware platforms have been employed as prototype platforms in all these implementations.

In this paper, for the first time, we consider the efficient implementation of point multiplication on Koblitz curves targeting

Manuscript received March 18, 2013; revised June 13, 2013; accepted July 03, 2013. Date of publication January 02, 2014; date of current version March 25, 2014. The work of K. U. Järvinen was supported in part by the Academy of Finland. This paper was recommended by Associate Editor M. Alioto.

R. Azarderakhsh is with the Department of Combinatorics and Optimization and the Center for Applied Cryptographic Research (CARC), University of Waterloo, Waterloo, ON N2L3G1, Canada (e-mail: razarder@uwaterloo.ca).

K. U. Järvinen is with the Department of Information and Computer Science, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland (e-mail: kimmo.jarvinen@aalto.fi).

M. Mozaffari-Kermani is with the Department of Electrical and Microelectronic Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA (e-mail: m.mozaffari@rit.edu).

Digital Object Identifier 10.1109/TCSI.2013.2283691

extremely-constrained devices. We represent the field elements using Gaussian normal basis (GNB). Bit-level multiplication using normal basis provides cheap squarings and, comparably, low area complexity, making it suitable for resource-constrained environments. The main contributions of this paper can be summarized as follows:

- We propose an efficient hardware architecture for point multiplication on binary Koblitz curves. In this regard, an efficient bit-level GNB multiplier is employed. The proposed crypto-processor requires fewer clock cycles in comparison with the counterparts.
- We share the addition/accumulation part of the bit-level multiplier to perform other field additions, resulting in lower area complexities. Moreover, employing normal basis representation provides cheap squarings which can be achieved by rewiring on hardware.
- We propose a new technique to compute point additions in affine coordinates. This technique is based on applying a recently introduced inversion algorithm [10] and it requires fewer registers to store intermediate variables than the traditional schemes.
- Finally, we synthesize the proposed architecture on the ASIC platform using a 65-nm CMOS technology. Our results show that the proposed efficient architecture for point multiplication on Koblitz curves consumes less energy compared to the previous works. The low number of clock cycles needed for the presented approach makes it suitable for high-throughput and energy-constrained applications.

The organization of this paper is as follows. In Section II, we review preliminaries of bit-level multiplication in GNB over $GF(2^m)$. In Section III, point multiplication on Koblitz curves using affine coordinates and our new point addition scheme is presented. In Section IV, the architecture of the proposed crypto-processor for point multiplication on Koblitz curves is presented. In Section V, the efficiency of the proposed crypto-processor is benchmarked through ASIC synthesis. Furthermore, we compare the area and timing results with the counterparts available in the literature. Finally, we conclude the paper in Section VI.

II. Preliminaries

A. Gaussian Normal Basis

It has been shown that there exists a normal basis for the binary extension field $GF(2^m)$ for all positive integers m. A normal basis is constructed by finding a normal element $\beta \in GF(2^m)$, where β is a root of an irreducible polynomial of degree m. Then, the set $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ is a basis for $GF(2^m)$ and its elements are linearly independent. In this case, $A \in GF(2^m)$ can be represented as $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in GF(2)$. Let $p = mT + 1$ be a prime number and $gcd(mT/k, m) = 1$, where k is the multiplicative order of 2 modulo p. Then, the normal basis $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ over $GF(2^m)$ is called the Gaussian normal basis (GNB) of type $T, T > 1$ [11]. GNB is attractive mainly because it provides efficient computations for multiplication.

For an element, say, $A = (a_0, a_1, \cdots, a_{m-1}) \in GF(2^m)$, squaring (power of two) can be written as $A^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}}$, where $\beta^{2^m} = \beta$. Hence, squaring is a linear operation and it can be obtained by a right cyclic shift as $A^2 = (a_{m-1}, a_0, a_1, \cdots, a_{m-2})$. Similar to the polynomial basis, addition can be obtained by a bit-wise XOR operation of two elements A and B as $A + B = \sum_{i=0}^{m-1} (a_i \oplus b_i) \beta^{2^i}$.

B. Multiplication Using GNB

The first bit-level normal basis multiplier was invented by Massey and Omura in [13]. It is a bit-level parallel-in serial-out (BL-PISO) multiplier in which all the coordinates of both input operands should be present throughout the multiplication operation. Bit-level serial-in parallel-out (BL-SIPO) multipliers were studied for normal basis and two different structures were proposed, namely, Least Significant Bit (LSB) first and Most Significant Bit (MSB) first structures by Beth and Gollmann in [12]. The BL-SIPO normal basis multipliers are advantageous for applications where one of the input operands is available in a bit-serial fashion. A parallel-in parallel-out (PIPO) version of this multiplier was presented in [14].

In what follows, we present the preliminaries for bit-level GNB multiplication over $GF(2^m)$. 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 in $GF(2^m)$. Then, $C \in GF(2^m)$ will be their product, i.e., $C = (c_0, c_1, \cdots, c_{m-1}) = AB = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \beta^{2^i + 2^j}$.

For GNB, the product of $A \in GF(2^m)$, given in bit-serial fashion, and $B \in GF(2^m)$ given in parallel can be written as [15]:

$$C = (\cdots((a_{m-1} \odot \rho(Y)) \gg 1 + a_{m-2} \odot \rho(Y \gg 1)) \gg 1 + \cdots) \gg 1 + a_0 \odot \rho(Y \gg m-1), \quad (1)$$

where $\rho(Y) = (y_1, s_0(1, Y), s_0(2, Y), \cdots, s_0(m-1, Y))$ and $s_0(i, Y) = \sum_{j=1}^{T} y_{R(i,j)} \in \{0, 1\}, 1 \le i \le m-1$. Note that R is an $(m-1) \times T$ multiplication matrix with the (i,j)-th element as $R(i,j)$, $0 \le R(i,j) \le m-1$, $1 \le i \le m-1$, and $1 \le j \le T$. Each row of the matrix R contains T integers in $[0, m-1]$. The architecture for the bit-level MSB-first SIPO GNB multiplication is depicted in Fig. 1(a). As one can see, every bit of operand B is available, while operand A should be available in serial with the MSB first. In this multiplier structure, registers $\langle Y \rangle$ and $\langle Z \rangle$ are initialized to $Y = (B \gg 1) = (b_{m-1}, b_0, b_1, \cdots, b_{m-2})$ and $0 = (0, 0, \cdots, 0)$, respectively. In the first clock cycle, the register $\langle Z \rangle$ contains $Z(1) = a_{m-1} \odot \rho(B \gg 1)$. Then, the registers $\langle Y \rangle$ and $\langle Z \rangle$ should be cyclically shifted to the right. Thus, after the m-th clock cycle, the register $\langle Z \rangle$ contains the coordinates of C, i.e., $Z(m) = C$. The implementation of $\rho(Y) \in GF(2^m)$ is performed by a ρ module for type T GNB as depicted in Fig. 1(b) which is a binary tree of XOR gates.

The structure of BL-SIPO GNB multiplier can be easily modified for a BL-PIPO architecture. In this case, both operands A and B should be available throughout the multiplication process. The multiplication matrix R is symmetric as $R(m-i, j) = R(i, j) + i \bmod m$, $1 \le i \le (m-1/2)$, $1 \le j \le T$. Therefore, one can reduce its size to $((m-1/2) \times T)$

TABLE I
COMPARISON AMONG BIT-LEVEL MULTIPLIERS FOR TYPE T GNB OVER $GF(2^m)$ WITH $2m-1 \leq C_N \leq Tm-T+1$. C_N DENOTES THE COMPLEXITY OF NORMAL BASIS AND IT IS MEASURED BY THE NUMBER OF ENTRIES OF MULTIPLICATION MATRIX R. FOR MORE DETAILS ABOUT ITS VALUES, ONE CAN REFER TO [16] AND [17]

Bit-level Multipliers	# AND	# XOR	# FFs	Critical path delay	Input	Output
MO [13]	C_N	C_N-1	$2m$	$T_A+\lceil\log_2 C_N\rceil T_X$	parallel	serial
IMO [18]	m	C_N-1	$2m$	$T_A+(\lceil\log_2 T\rceil+\lceil\log_2 m\rceil)T_X$	parallel	serial
BG [12]	m	C_N	$2m$	$T_A+(1+\lceil\log_2 T\rceil)T_X$	serial	parallel
GG [14]	m	$\leq\frac{C_N+m}{2}$	$3m$	$T_A+(1+\lceil\log_2(T+1)\rceil)T_X$	parallel	parallel
RH [19]	$\frac{m+1}{2}$	$\leq\frac{C_N+2m-1}{2}$	$3m$	$T_A+(1+\lceil\log_2(T+1)\rceil)T_X$	parallel	parallel
AR[1][20], [21]	m	$\leq\frac{C_N+m}{2}$	$3m$	$T_A+(1+\lceil\log_2 T\rceil)T_X$	parallel	parallel

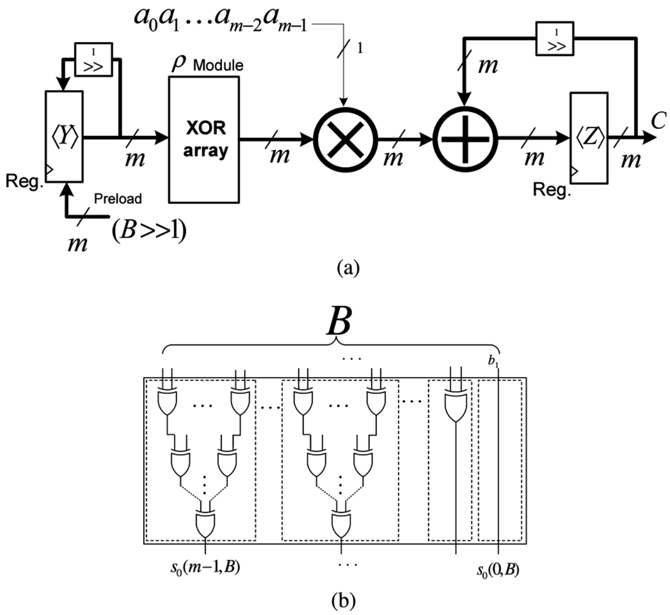

Fig. 1. (a) The architecture of MSB-first bit-level normal basis multiplier with parallel output [12] (b) The architecture of P module for type T GNB.

and hence reduce the area complexity of the ρ module and consequently the entire multiplier.

We compare the time and area complexities of the bit-level normal basis multipliers in Table I, where area complexities in terms of gate count and time complexities in the form of critical path delay are depicted. As one can see, the PIPO architecture requires smaller area in comparison with the counterparts. MO [13] and IMO [18] are two multipliers with parallel-input and serial-output architectures as the results of multiplication are available in serial in each clock cycle. It should be noted that for real applications, e.g., in point multiplication of ECC, one needs to store the products in a register to start the next finite field operation in the computation of point addition and doubling (for point multiplication). Therefore, for real applications, these multipliers also require three registers. BG [12] has a serial-input parallel-output architecture and similar to the other architectures, for real applications, one needs to have both input operands stored in a register (shift register) during the multiplication process and, hence, it requires three registers as well. However, it should be noted that if one employs bit-level or digit-level hybrid multipliers, it is possible to utilize the results from serial-out architecture as the input to the serial-in architecture. For more information, one can refer to [15] and [22]. In this paper, we employ the BL-PIPO architecture of [20] and [21].

III. POINT MULTIPLICATION ON KOBLITZ CURVES

Let $E(GF(2^m))$ be the group of points on an elliptic curve over a binary extension field $GF(2^m)$; i.e., the points (x,y) that satisfy the elliptic curve equation together with a special point called the point at infinity. The group operation $(x_3,y_3)=(x_1,y_1)+(x_2,y_2)$ is called point addition. If $x_1=x_2$ and $y_1=y_2$, point addition is called point doubling. The Frobenius map for points in $E(GF(2^m))$ can be defined as a map $\phi: E(GF(2^m))\rightarrow E(GF(2^m)), (x,y)\mapsto(x^2,y^2)$. The squaring over $GF(2^m)$ using GNB is a rewiring operation in hardware which implies that the Frobenius map can be carried out with no cost [23]. Koblitz [24] showed that the cheap Frobenius map can be used instead of point doublings if the binary curve is defined by the equation

$$y^2+xy=x^3+ax^2+1, \qquad (2)$$

where $a\in\{0,1\}$ and $x,y\in GF(2^m)$. For clarity, we denote the group of points on a Koblitz curve by $E_K(GF(2^m))$. One can show that $\phi^2(P)-\mu\phi(P)+2P=0$ for every $P\in E_K(GF(2^m))$. Let τ be the complex root of $P(T)=T^2-\mu T+2$ which is the characteristic polynomial of the Frobenius endomorphism. Then, if one represents the scalar k in τ-adic non-adjacent form (τNAF), i.e., $k=\sum_{i=0}^{l-1}k_i\tau^i$ for $k_i\in\{0,1,-1\}$ and $k_ik_{i+1}=0$, then, point multiplication can be computed as $kP=\sum_{i=0}^{l-1}k_i\phi^i(P)$ [23]. For efficient computation of τNAF conversion, one can refer to [25]–[28]. In normal basis, when $P=(x,y)$ is known, $\phi^i(P)$ can be computed by i-fold right cyclic shifts of x and y, i.e., $\phi^i(P)=(x^{2^i},y^{2^i})=(x\gg i, y\gg i)$. The faster computation of $\phi(P)=(x\gg 1, y\gg 1)$ in normal basis results in a faster point multiplication of $Q=kP=\sum_{i=0}^{m-1}k_i\phi^i(P)$ than the traditional methods [29]. In Algorithm 1, the point multiplication algorithm is presented, having k represented in τNAF [23].

Algorithm 1 Point multiplication on Koblitz curves using affine "Frobenius-and-add-or-subtract" algorithm [23]

Inputs: A point $P=(x,y)\in E_K(GF(2^m))$ on curve and an integer k, $k=\sum_{i=0}^{l-1}k_i\tau^i$ for $k_i\in\{0,\pm1\}$

Output: $Q=kP$

1: **initialize**
  i: **if** $k_{l-1}=1$ **then** $Q\leftarrow(x,y)$

ii: **if** $k_{l-1} = -1$ **then** $Q \leftarrow (x, x + y)$

2: **for** i **from** $l - 2$ **downto** 0 **do**

$Q \leftarrow \phi(Q) = (x^2, y^2)$

if $k_i \neq 0$ **then**

$Q \leftarrow Q + k_i P$

end if

end for

3: **return** Q

In Step 2 of Algorithm 1, one needs to perform point addition for every non-zero bit of the scalar k given in $\tau\mathrm{NAF}$. The point addition on Koblitz curves can be performed in different coordinates including affine, projective, and mixed coordinates. Note that it is possible to define point operations on Koblitz curves with more than three different coordinates (e.g., affine, standard projective, Jacobian projective, Lopez-Dahab, and different mixes of them) but we consider only three alternatives in this paper.

In affine coordinates, point addition $(x_3, y_3) = (x_1, y_1) + (x, y)$ is computed as follows ($P = (x, y)$ is the base point):

$$\lambda = \frac{y_1 + y}{x_1 + x}, \tag{3}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x + a, \tag{4}$$

$$y_3 = \lambda(x_3 + x) + x_3 + y. \tag{5}$$

In Table II, the costs of the combined point addition and doubling (only point addition for Koblitz curves) are given for binary generic, binary Edwards, and binary Koblitz curves. As one can see, addition in affine coordinates costs $1I+2M+1S+9A$, where I, M, S, and A are the costs of inversion, multiplication, squaring, and addition, respectively. Inversion can be efficiently computed using Fermat's Little Theorem (e.g., Itoh-Tsujii (IT) [30]) or Extended Euclidean Algorithm.

A. Selection of the Coordinate System

For our resource-constrained targets, we focus on minimizing the area as much as possible. Projective coordinates, where a point is represented with three coordinates (x, y, z), are commonly used for improving the speed of point multiplications because they allow trading expensive inversions to cheaper multiplications. On the other hand, traditional affine coordinates, where a point is represented with two coordinates (x, y), require simpler control structure and fewer registers to store the points and temporary variables and, as a result, lead to simpler and smaller (but, of course, slower) implementations.

Next, we show that the speed difference between affine and mixed coordinates is not so radical that it would advocate mixed coordinates instead of affine for resource-constrained implementations. An inversion in $GF(2^{163})$ requires at least $I = 9M + 162S$ with all known algorithms based on Fermat's Little Theorem. Because we are using a bit-level multiplier and each squaring requires one clock cycle, we have $162S \approx M$. Consequently, a point addition (or subtraction) requires $I + 2M \approx 12M$ in affine coordinates and $8M$ in mixed coordinates. With mixed coordinates, the result point needs to be converted to affine coordinates in the end of a point multiplication which requires $I + 2M \approx 12M$. Consequently, the cost of Algorithm 1 can be estimated as $(H(k) - 1) \times 12M \approx 640M$ for affine coordinates and $(H(k) - 1) \times 8M + 12M \approx 439M$ for mixed coordinates, where $H(k) \approx m/3$. We conclude that this reduction of less than one-third in the expected latency is not significant enough to support using mixed coordinates in resource-constrained implementations because it would come at the expense of increased number of registers and more complex control structure. Hence, we use affine coordinates in our implementation.

Further analysis about the advantages of using affine coordinates instead of mixed coordinates in the case of our processor is given in Section IV-A after describing the algorithms and the architecture.

TABLE II

COST OF POINT OPERATIONS ON BINARY GENERIC CURVES (BGCS) [31], BINARY EDWARDS CURVES (BECS) [32], AND BINARY KOBLITZ CURVES (BKCS) [33] OVER $GF(2^m)$

Curve	Coordinate	PA and PD
BGC	Affine	$2I + 4M + 2S + 9A$
	Projective	$8M + 5S + 9A$
	Mixed	$6M + 5S + 9A$
BEC	Affine	$2I + 3M + 3S + 7A$
	Projective	$9M + 4S + 7A$
	Mixed	$6M + 4S + 7A$
BKC[1]	Affine	$1I + 2M + 1S + 9A$
	Projective	$13M + 4S + 9A$
	Mixed	$8M + 5S + 9A$

1. Cost of point addition only.

B. Multiplicative Inverse and Point Addition Algorithm

As shown above, the bulk of point addition in affine coordinates is spent in computing the inversion. Similarly as in many other hardware implementations (see, e.g., [34]), we chose to compute inversions via exponentiations based on Fermat's Little Theorem because it leads to a simpler and smaller implementation compared to Extended Euclidean Algorithm when a multiplier and a squarer are already available.

The IT scheme [30] requires 162 squarings and 9 multiplications in $GF(2^{163})$. The recently introduced Dimitrov-Järvinen (DJ) algorithm [10] requires the same amount of multiplications and squarings but allows using fewer registers in the implementation compared to the IT scheme. The DJ algorithms are based on finding double or triple base representations for $m - 1$ and they require, on average, fewer multiplications [10]. There are also several other recent proposals that improve inversion computations [35]–[37]. They focus mainly on fast computation of successive squarings which increases area requirements and makes them impractical for extremely constrained applications. To the best of our knowledge, the DJ algorithm computes an inversion in $GF(2^{163})$ with the smallest possible number of multiplications, squarings, and temporary variables.

In the case of $GF(2^{163})$, the IT inversion algorithm is based on the following decomposition for the exponent

$2(1+2+\ldots+2^{m-2})$ of the exponentiation $A^{-1}=A^{2^m-2}=A^{2(1+2+\ldots+2^{m-2})}$ [30]:

$$\begin{aligned}1&+2+2^2+\ldots+2^{161}\\&=(1+2)\times\left(1+2^2\times(1+2^2)\times(1+2^4)\times(1+2^8)\times(1+2^{16})\right.\\&\qquad\left.\times\left(1+2^{32}\times(1+2^{32})\times(1+2^{64})\right)\right).\end{aligned}\tag{6}$$

The DJ algorithm, on the other hand, is based on the following decomposition [10]:

$$\begin{aligned}1+2+2^2+\ldots+2^{161}&=(1+2+2^2)\times(1+2^3+2^6)\\&\times(1+2^9+2^{18})\times(1+2^{27}+2^{54})\times(1+2^{81}).\end{aligned}\tag{7}$$

Both of the above decompositions require the same number of multiplications (additions in the above formulae) and squarings, but the latter saves one temporary variable [10].

This saving originates from the fact that (6) requires one long-time variable whereas (7) does not [10]. When one implements the operations that are required by a term of the form $(1+2^n)$, one needs two variables T_1 and $T_2=T_1^{2^n}=T_1\gg n$ and computes $T_1\leftarrow T_1\times T_2$. For (6), a third variable, T_3, is needed for storing a value while, for example, $1+2^{32}(1+2^{32})(1+2^{64})$ is computed. However, for (7), such a long-time variable is not needed as it comprises only terms of the form $(1+2^n+2^{2n})$ which can be computed with only T_1 and T_2 using the following sequence of operations: $T_2\leftarrow T_1\gg n$, $T_1\leftarrow T_1\times T_2$, $T_2\leftarrow T_2\gg n$, and $T_1\leftarrow T_1\times T_2$.

We employed the DJ algorithm in computing point addition in affine coordinates using (3)–(5) and the resulted algorithm is shown in Algorithm 2. We managed to reduce the number of registers (to store intermediate results) to only two, i.e., T_1 and T_2. If we had used the IT algorithm, we would have needed three registers (T_1, T_2, and T_3).

Algorithm 2 Affine point addition on Koblitz curves using DJ inversion scheme [10], [23]

Inputs: $P=(x,y)$ and $P_1=(x_1,y_1)\in E_K(GF(2^m))$

Output: $P_3=P_1+P$

1: $T_1\leftarrow x_1+x$

2: **Compute**: inversion using the DJ scheme over $GF(2^{163})$

2.1: $T_1\leftarrow T_1\gg 1$,	2.2: $T_2\leftarrow T_1\gg 1$,
2.3: $T_1\leftarrow T_1\times T_2$,	2.4: $T_2\leftarrow T_2\gg 1$,
2.5: $T_1\leftarrow T_1\times T_2$,	2.6: $T_2\leftarrow T_1\gg 3$,
2.7: $T_1\leftarrow T_1\times T_2$,	2.8: $T_2\leftarrow T_2\gg 3$,
2.9: $T_1\leftarrow T_1\times T_2$,	2.10: $T_2\leftarrow T_1\gg 9$,
2.11: $T_1\leftarrow T_1\times T_2$,	2.12: $T_2\leftarrow T_2\gg 9$,
2.13: $T_1\leftarrow T_1\times T_2$,	2.14: $T_2\leftarrow T_1\gg 27$,
2.15: $T_1\leftarrow T_1\times T_2$,	2.16: $T_2\leftarrow T_2\gg 27$,
2.17: $T_1\leftarrow T_1\times T_2$,	2.18: $T_2\leftarrow T_1\gg 81$,
2.19: $T_1\leftarrow T_1\times T_2$.	

3: $T_2\leftarrow y_1+y$

4: $T_1\leftarrow T_1\times T_2$

5: **Compute**: x_3

5.1 $T_2\leftarrow T_1\gg 1$

5.2 $T_2\leftarrow T_1+T_2$

5.3 $x_1\leftarrow x_1+T_2$

5.4 $x_1\leftarrow x_1+x$

5.5 $x_1\leftarrow x_1+1$

6: **Compute**: y_3

6.1 $T_2\leftarrow x_1+x$

6.2 $T_1\leftarrow T_1\times T_2$

6.3 $y_1\leftarrow T_1+x_1$

6.4 $y_1\leftarrow y_1+y$

7: **return** (x_1,y_1)

IV. Architecture of Point Multiplication Processor

In this section, we explain the proposed architecture for point multiplication on Koblitz curves. The architecture of the proposed crypto-processor is illustrated in Fig. 2. The architecture is composed of three main components: field arithmetic unit (FAU), register file, and control unit. Our target is to implement point multiplication on a Koblitz curve over the smallest NIST field $GF(2^{163})$ with as small area as possible. FAU performs finite field multiplication, squaring, and addition in GNB over $GF(2^{163})$. As shown in Fig. 2, we selected a BL-PIPO GNB multiplier as discussed in Section II. It employs a ρ' module for which the number of XOR gates inside it is about half of the ρ module. The J block is composed of m AND gates which perform AND operation for two m-bit inputs. Beyond that, we aim to minimize the number of registers and the area of other functions.

Algorithm 2 needs altogether six variables (x_1, y_1, x, y, T_1, and T_2) which in turn lead to six m-bit registers[1] in the processor. If we had used the decomposition of the IT algorithm, (6), instead of the decomposition of the DJ algorithm, (7), then we would have needed seven m-bit registers because T_3 would have been needed as discussed in Section III-B. In addition to these, the scalar k requires a register. We are using the Frobenius-and-add-or-subtract algorithm with τNAF because it offers significantly faster performance with negligible increase in the complexity of implementation. The maximum length of τNAF is $m+a=164$ bits [27] and $k_i\in\{-1,0,1\}$ which imply that a $2(m+1)=328$-bit register is needed for k. In [38], Joye and Tymen observed that the fact that a nonzero is always followed by a zero allows devising an encoding requiring only $m+1$ bits. We use their left-to-right encoding for τNAF [38] $(10\rightarrow 10, \bar{1}0\rightarrow 11, 0\rightarrow 0)$ and, consequently, need only a 164-bit register for k.

[1]In some applications, the base point P is fixed and x and y can be hardwired.

The field multiplier includes three m-bit registers X, Y, and Z. Multiplications in Algorithm 2 are always of the form $T_1 \times T_2$; hence, we can reuse the registers for T_1 and T_2 as registers for the operands of multiplications and effectively save the area of two 163-bit registers.

We embed functionalities required for other field operations into the multiplier data path by adding two multiplexers (s_1 and s_2). This increases the critical path of the multiplier only by a delay of a 2-input multiplexer. Based on the architecture of the proposed crypto-processor for point multiplication on Koblitz curves, the field operations are computed as follows:

- ***Addition*** $C \leftarrow A + B$: First, A is loaded to Z by selecting A with s_R and setting $s_1 = 1$ and $s_2 = 0$. Second, B is selected with s_R and the addition is computed by setting $s_1 = 1$ and $s_2 = 2$ which results in $Z = A + B$. Finally, Z is stored into C. Thus, addition takes three clock cycles. If $B = 1$, then the second operand is not needed and the latency is two clock cycles.
- **Squaring** $C \leftarrow A \gg 1$: First, A is loaded to Z by selecting A with s_R and setting $s_1 = 1$ and $s_2 = 0$. Second, the first operand of the adder is set to zero by selecting $s_1 = 2$ and the second operand is selected by setting $s_2 = 3$. Now, Z includes $0 + Z^2 = A^2$. Finally, Z is stored into C. Thus, squaring takes three clock cycles.
- **Squaring** $T_{1,2} \leftarrow T_{1,2} \gg 1$: Several times Algorithm 2 requires T_1 or T_2 to be squared and stored to itself. In these cases, we can utilize the squarers attached to T_1 and T_2 and avoid going through Z. Each of these squarings takes one clock cycle.
- **Multiplication** $T_1 \leftarrow T_1 \times T_2$: The multiplexers are set to $s_1 = 0$, $s_2 = 3$ (except $s_2 = 0$ for the first clock cycle), and $s_{T_1} = s_{T_2} = 0$. This is continued for 163 clock cycles after which Z includes $T_1 \times T_2$. Finally, Z is stored into T_1. Thus, multiplication takes 164 clock cycles.
- **Certain consecutive operations**: When Z already includes the operand for the next operation or the result of an operation is not needed after the next operation, loads and saves from and to the registers can be avoided, respectively. It is also possible to combine two operations into a single operation in certain cases. For instance, we need only six clock cycles to compute the lines 5.1–5.5 of Algorithm 2: T_1 is in Z, then, $Z \leftarrow T_1 + (Z \gg 1)$, $Z \leftarrow x_1 + Z$, $Z \leftarrow x + Z$, $x_1 \leftarrow Z$, $Z \leftarrow x_1 + 1$, and $x_1 \leftarrow Z$. Straightforward application of the above descriptions would give 14 clock cycles.

If the selection for the unity element ($s_2 = 1$) was moved to the other multiplexer (s_1), we would save one clock cycle per point addition but this would increase the critical path by another 2-input multiplexer and, therefore, we opted to use the setup of Fig. 2.

The controller executes point addition, point subtraction, and Frobenius map routines according to Algorithm 1. The first point addition (the initialization) is carried out simply by transferring (x, y) or $(x, x + y)$ into (x_1, y_1). The routines implemented in the controller are collected in Fig. 3.

The controller takes in two most significant bits (MSB) of the register k. Joye-Tymen encodings are parsed so that the MSB determines whether a digit is zero or nonzero. If the digit is nonzero, the second MSB determines its sign (0 is positive and 1 is negative). After a digit has been processed, the register k is shifted to the left once or twice for zero and nonzero digits, respectively. After a nonzero, an additional Frobenius map is computed.

The architecture computes point additions and point subtractions in $11M + 192 = 1985$ and $11M + 194 = 1987$ clock cycles, respectively. A Frobenius map $\phi(Q)$ takes 5 clock cycles. We map y_1 first because it is already in Z when the previous point addition ends. Similarly, we have x_1 in Z when the next point addition starts. The initialization takes 4 clock cycles. Consequently, point multiplication takes, on average, approximately 106,700 clock cycles, assuming that $H(k) \approx m/3$, and point additions and point subtractions divide evenly.

A. Comparison of Coordinate Systems

In Section III-A, it was estimated that mixed coordinates would give an improvement of about one-third in latency with the expense of increased area. In order to shed more light on this issue, we derived an algorithm for point addition using mixed coordinates. This algorithm and derivations of the number of registers and latencies are provided in the Appendix. Point multiplication would take, on average, approximately 76,100 clock cycles using mixed coordinates which is a 29% improvement over the variant with affine coordinates. The area improvement is significant despite the fact that the algorithm utilizes similar resource sharing that was used for affine coordinates: the number of registers increases with two 163-bit registers which is a 33% increase (or a 50% increase if $P = (x, y)$ is hardwired). Also, the complexity of the control unit grows significantly compared to the simple logic used for affine coordinates. As will be shown in Section V-A, the register file and the control unit require about half of the area of the processor and, consequently, it is essential to keep them small. Hence, the selection of affine coordinates is justified.

V. ASIC Implementation

The results of ASIC implementations for the proposed and previous works are presented in this section. In what follows, first, we present the results for our work. This includes the area complexity, timing, power, and energy consumptions on ASIC. Then, we compare the obtained results with previous counterparts.

A. Implementation Results

We have used the STM 65-nm CMOS standard technology and CORE65LPSVT standard cell library [39] for our results. This library has been optimized for low-power applications. VHDL has been used as the design entry to the Synopsys Design Compiler [40]. In addition, using the area of a NAND gate in the utilized STM 65-nm CMOS library which is 2.08 $\mu\mathrm{m}^2$ [39], we have provided the gate equivalent (GE) so that area comparisons among different technologies are meaningful. We note that although similar to the previous works presented in [3]–[5], and [7], we have not fabricated a chip on silicon, our detailed results are intended for benchmarking the metrics for the previous and proposed research works. Wire load models are generally used in the synthesis process for estimating the

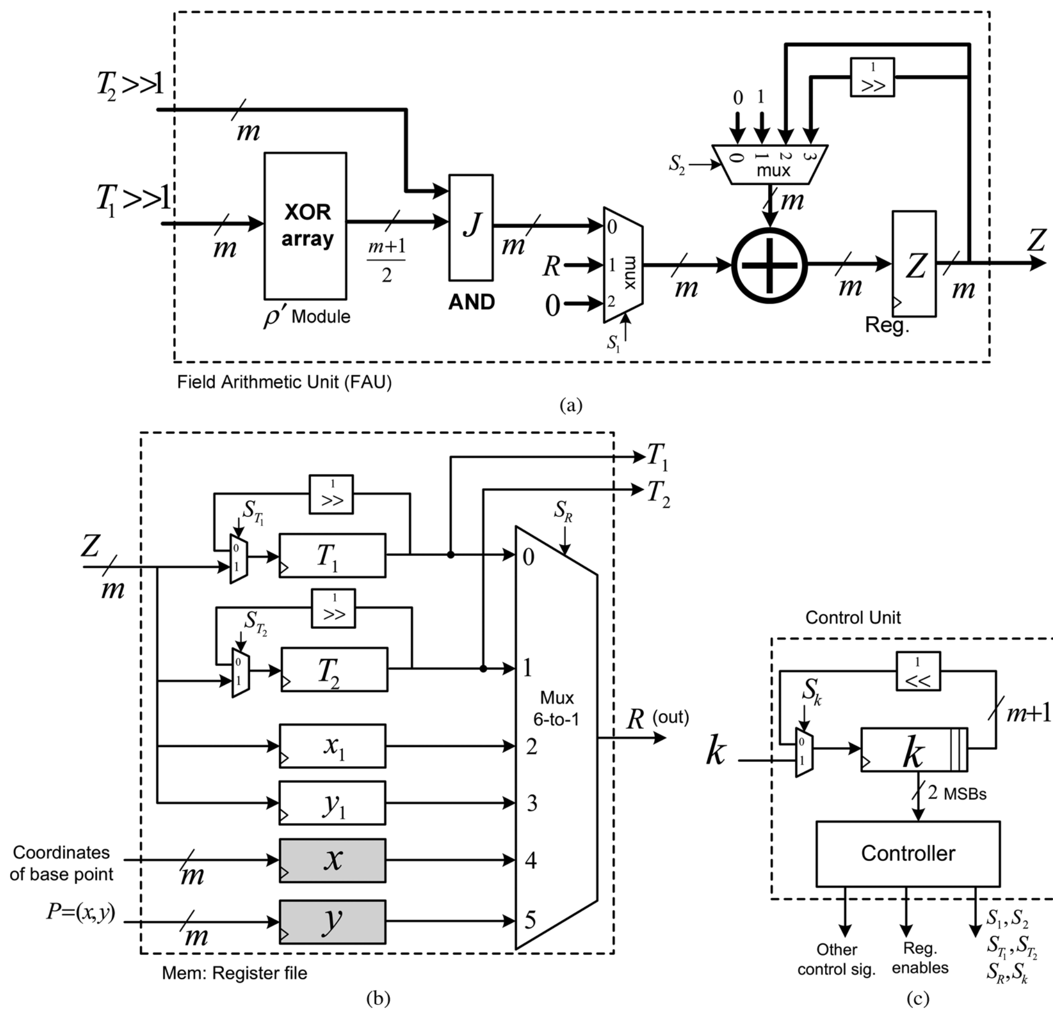

Fig. 2. The architecture of the proposed crypto-processor for point multiplication on Koblitz curves which includes (a) field arithmetic unit (FAU), (b) register file, and (c) control unit.

net delays during pre-layout in a design cycle. These data are based on statistics from physical layout parasitics. The accurate analysis of interconnect delay is derived through physical design; yet, the presented results using these models are estimates of the interconnect delays (acceptable but not exactly the real delay figures), taking into account the estimated hardwiring delays. We would like to point out that these models are pre-characterized equations, attempting to estimate (predict) gates' capacitive loads (based on fan-out and the overall design area).

As discussed in the previous section and seen in Table III, 106,700 cycles are needed for our proposed architecture with the area of 11,571 GE. From this total area, the FAU has the area of 5,328 GE and the rest is for the register file and the control unit. If the coordinates of the base point are hardwired, the area is 10,299 GE. The critical-path delay is determined by the path from the register T_1 to the register Z which are the input and output of the BL-PIPO multiplier. The critical-path delay of the multiplier is stated in Table I which is $T_A + (1 + \lceil \log_2 T \rceil) T_X$ excluding the delay of registers. Sharing add/accumulation components only adds a multiplexer (the mux with S_1) to the critical-path. The critical-path delay of the proposed architecture is 0.54 ns. We note that the achieved frequency is 1.85 GHz without over-constraining the architecture.

While this frequency is very high, for power derivation purposes, similar to [8] and taking RFID tags as reference, we set the frequency to 106 kHz which is 1/128 of 13.56 MHz. We also provide our results for the frequency of 13.56 MHz which is normally the frequency used for RFID applications (carrier signal used by the ISO-18000-3-1 RFID standard). Although the power consumptions differ significantly for these two frequencies, the energies are very close, i.e., 0.65 μJ and 0.61 μJ, respectively.

B. Comparison

In what follows, we present the ASIC results and comparisons with the previous works. There are many recent papers including [41]–[46] that study FPGA-based implementations of Koblitz curves for high-speed applications but it is impossible to compare our results with the results of those papers because both the applications and the implementation platforms are different. We are also not aware of any implementations of Koblitz curves for constrained applications (either ASIC or FPGA) and,

Initialization

$Z \leftarrow x$
$x_1 \leftarrow Z$
if $k_{l-1} = -1$ **then**
  $Z \leftarrow Z + y$
else
  $Z \leftarrow y$
$y_1 \leftarrow Z$

Frobenius map, $\phi(x_1, y_1)$

$Z \leftarrow Z \gg 1$
$y_1 \leftarrow Z$
$Z \leftarrow x_1$
$Z \leftarrow Z \gg 1$
$x_1 \leftarrow Z$

Point addition / subtraction

$Z \leftarrow Z + x$
$Z \leftarrow Z \gg 1$
$T_1 \leftarrow Z$
$Z \leftarrow Z \gg 1$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$T_2 \leftarrow T_2 \gg 1$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$Z \leftarrow Z \gg 3$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$T_2 \leftarrow T_2 \gg 3$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$Z \leftarrow Z \gg 9$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$T_2 \leftarrow T_2 \gg 9$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$Z \leftarrow Z \gg 27$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$T_2 \leftarrow T_2 \gg 27$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$Z \leftarrow Z \gg 81$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$Z \leftarrow y_1$
$Z \leftarrow Z + y$
if subtraction **then**
  $Z \leftarrow Z + x$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$T_1 \leftarrow Z$
$Z \leftarrow (Z \gg 1) + T_1$
$Z \leftarrow Z + x_1$
$Z \leftarrow Z + x$
$x_1 \leftarrow Z$
$Z \leftarrow x_1 + 1$
$x_1 \leftarrow Z$
$Z \leftarrow Z + x$
$T_2 \leftarrow Z$
$Z \leftarrow T_1 \times T_2$
$Z \leftarrow Z + x_1$
$Z \leftarrow Z + y$
if subtraction **then**
  $Z \leftarrow Z + x$
$y_1 \leftarrow Z$

Fig. 3. Routines used for computing point additions, point subtractions, Frobenius maps, and the initialization. Each line takes one clock cycle, except multiplications ($\times$) take $m = 163$ clock cycles and rotations ($\gg e$) take e clock cycles. Point addition/subtraction routine was derived from Algorithm 2.

TABLE III
COMPARISON OF DIFFERENT POINT BIT-LEVEL MULTIPLICATIONS OVER $GF(2^{163})$ ON ASIC

Work	Curve	Tech. (nm)	Mult.	# of clock cycles	Coord.	Area (GE)	Freq.[1] (kHz)	Time (ms)	Power (μW)	Energy[2] (μJ)
[9]	BEC	130	Bit-serial	219,148	Mixed	11,720	400	547.87	7.3	3.98 (1.30)
[7]	BGC	130	Bit-serial	275,816	Mixed	12,506	1,130	244.08	36.6	8.94 (2.93)
[6][3]	BGC	130	Bit-serial	353,710	Mixed	8,214[3]	500	707.42	< 30	< 21.22 (< 6.96)
[5]	BGC	350	Bit-serial	376,864	Affine	16,207	13,560	27.90	–	–
[8]	BGC	180	Bit-serial	296,299	Affine	13,250	106	2,792	8.57	23.92 (3.26)
This work	BKC	65	Bit-serial	106,700	Affine	11,571 [10,299][4]	106	1,006.6	0.66	0.65 (1.06)
							13,560	7.87	77.2	0.61 (1.00)

1. The frequency used for power derivations (not the maximum achievable frequency).
2. Energy for one point multiplication. The numbers in parentheses are estimations of the normalized energies, considering 65-nm as the base.
3. The synthesis results do not include RAM. For a typical synthesis, the total area is reported to be roughly 12 kGE.
4. This is the area considering the hardwired coordinates of the base point in Fig. 2(b).

hence, we compare our processor only to compact ASIC implementations using binary general or Edwards curves.

In Table III, we have included the performance metrics of the previous work: the standard cell library and technology used, the number of clock cycles for point multiplications, the coordinates, i.e., mixed or affine, the frequency used for power derivations, the total time needed for computations, and the power/energy consumptions (considering parasitics and node transitions).

In [9], using a 0.13 μm technology and mixed coordinates, the implementation results have been derived (see first row in Table III for digit-size of one). In total, as seen in this table, 219,148 clock cycles are needed and the area in terms of GE is 11,720. For the frequency of 400 kHz, the dynamic power and energy for performing one point multiplication are 7.3 μW and 3.98 μJ, respectively. In [7], the squaring operations use the same logic as the multiplication and, hence, each squaring requires m clock cycles. Also, the multiplication and addition operations share the XOR array. In this work, the results were derived using a 0.13 μm technology and mixed coordinates (see second row in Table III). As seen in the table, for the Type 1 architecture in [7] (which is the minimal version and has the least gate area and the most number of cycles), the total area (GEs) is 12,506. Moreover, for the frequency of 1,130 kHz, the dynamic power and the energy for one multiplication are 36.6 μW and 8.94 μJ, respectively.

It is noted that in [6], the conversion from projective coordinates to affine coordinates is not considered in the computation of the point multiplication. Also, squaring is considered as a special case of multiplication in order to minimize the area. Therefore, the number of clock cycles is computed as $(l-1) \times (13(M+3)+12))$. In [6], 353,710 clock cycles and 8,214 GE are obtained using a 0.13 μm technology and mixed coordinates. Moreover, for the frequency of 500 kHz, dynamic power and energy of less than 30 μW and 21.22 μJ are reported, respectively. In [5], using a 0.35 technology, with 376,864 clock cycles and area of 16,207 GE, the total time needed for the point multiplication is 27.90 ms with a frequency of 13,560 kHz. Finally, in [8] (0.18 μm technology), the operation is done in 296,299 clock cycles and the area is 13,250 GE. Moreover, with the utilized frequency of 106 kHz, the power and energy are 8.57 μW and 23.92 μJ, respectively.

Comparing energies among different technologies is necessary when previous works are considered. Yet, exact comparison is not possible when technologies are different (this could be the reason that in previous works this comparison through power conversions among technologies is not performed). One can roughly estimate and compare the energy equivalents among different works. For this purpose, according to [47], if we consider full voltage scaling, the energy conversion ratio of s^3 (from smaller to larger technologies) is obtained. However, if constant voltage is considered, the energy conversion ratio of

s is derived, where s is the scaling factor [47]. We consider the average energy conversion ratio of these two, i.e., $(s+s^3)/2$. We would like to emphasize that this conversion is only a rough estimate and does not intend to exactly benchmark the energy variations among different technologies. Based on these conversions, the energy consumption of the proposed approach is significantly lower than the counterparts (the closest is the architecture presented in [9] which is 30% more energy-demanding; refer to the estimated normalized energies shown in parentheses in the energy column of the table). This is mainly because of the much lower number of clock cycles needed for our scheme (at least half of the other schemes in Table III).

In addition to voltage scaling, one needs to note that frequency scaling might be considered when different technologies are compared. In this regard, if power consumption is of concern, higher frequencies lead to higher power consumptions (linear relation) in a typical technology or when technologies are changed (in the latter case, the effects of other factors such as voltage or switching factors need to be taken into account). Nevertheless, concerning energy consumption, frequency scaling does not necessarily affect the battery usage or energy drainage due to the independence of energy consumption from the operating frequency. For instance, in higher frequencies, power consumption is higher but the total energy consumption is intact as the time needed for realizing a specific operation is shortened.

Based on the above observations, the proposed architecture for point multiplication on Koblitz curves is suitable for energy-constrained and efficient applications.

VI. Conclusions

In this paper, we have proposed an efficient implementation of point multiplication on Koblitz curves for extremely-constrained applications such as RFIDs and sensor networks. We have represented the field elements by GNB over $GF(2^m)$ and used bit-level multiplications. One main advantage of these multipliers is their cheap squarings in hardware and providing low area complexity suitable for resource-constrained and secure environments. Through sharing the addition/accumulation part of the bit-level multiplier to perform other field additions, lower area complexities have been achieved. We have also proposed a new technique for computing point addition in affine coordinate. This approach needs fewer registers for storing the intermediate variables compared to the traditional schemes. Comparing the results of our ASIC implementation using a 65-nm CMOS library to the previously presented works shows that our work has lower energy consumption and requires less than half of the clock cycles to compute point multiplications. Consequently, the proposed efficient implementation of point multiplication on Koblitz curves is suitable for extremely-constrained environments.

Next, we analyze and discuss two possible directions for future research.

Implementation attacks are cryptanalytic attacks that utilize information obtained from a physical implementation of a cryptographic algorithm through a side-channel, e.g., timing [48], power [49], or electromagnetic radiation [50] or by analyzing results from the implementation after deliberately injected faults [51]. They form a serious threat for cryptosystems in practice. Countermeasures against these attacks typically come with significant costs in area and latency which makes their use challenging in extremely-constrained applications. As a consequence, the most appropriate countermeasures must be selected based on a careful risk analysis of the application. Because our target was to minimize the resource requirements and to propose a general processor architecture that could be useful in a large variety of applications (perhaps after small modifications), our processor does not implement any specific countermeasures. However, certain simple countermeasures could be added with small overheads. These include, e.g., blinding the sign of k_i by using dummy additions for point additions (lines 37 and 52 in Fig. 3) and blinding the positions of nonzero k_i by using dummy Frobenius maps. More generally, developing and applying low-cost countermeasures for our processor and other extremely-constrained processors is an important topic for future research.

Elliptic curve cryptosystems (e.g., ECDSA [4]) commonly require modular integer arithmetic in addition to elliptic curve operations. While this requirement for two types of arithmetic (binary field and modular integer) creates challenges for all systems using binary curves in constrained environments, the challenges can become greater for Koblitz curves because conversions between integers and τ-adic representations may be needed too. High-speed hardware architectures of these conversions have been proposed [25], [26], [28] but none of them is suitable for extremely constrained applications. Even the smallest converter [26] requires too many resources (e.g., 648 ALUTs and 683 registers on a Stratix II FPGA for $GF(2^{163})$). In many applications, it is possible to use cryptosystems where conversions can be avoided by generating τ-adic representations at random [27] or by pre-computing and hardwiring them. Extremely-constrained applications typically require careful fine-tuning (e.g., by fixing certain parameters, etc.) of the cryptosystems that are used in the application and these aspects should be taken into account when considering different cryptosystems for the application. Nevertheless, techniques for computing conversions with minimal resources and/or with resources shared with the architecture computing point multiplications should be studied in the future.

Appendix

In mixed coordinates, a point is given in Lopez-Dahab coordinates (x_1, y_1, z_1) and the other in affine coordinates (x, y) and the point addition $(x_3, y_3, z_3) = (x_1, y_1, z_1) + (x, y)$

Algorithm 3 Point addition in mixed coordinates with register sharing

Inputs: $P = (x, y)$ and $P_1 = (x_1, y_1, z_1) \in E_K(GF(2^m))$

Output: $P_3 = (x_1, y_1, z_1) = P_1 + P$

1: $T_1 \leftarrow z_1 \gg 1$

2: $T_2 \leftarrow y$

3: $T_1 \leftarrow T_1 \times T_2$

4: $T_3 \leftarrow y_1 + T_1$ (A)

5: $T_1 \leftarrow x$
6: $T_2 \leftarrow z_1$
7: $T_2 \leftarrow T_1 \times T_2$
8: $T_2 \leftarrow x_1 + T_2$ (B)
9: $x_1 \leftarrow T_2 \gg 1$
10: $x_1 \leftarrow x_1 + T_3$ $(A + B^2)$
11: $T_1 \leftarrow z_1$
12: $T_2 \leftarrow T_1 \times T_2$ (C)
13: $x_1 \leftarrow x_1 + T_2$ if $a = 1$ $(A + B^2 + aC)$
14: $T_1 \leftarrow x_1$
15: $x_1 \leftarrow T_1 \times T_2$ $(C(A + B^2 + aC))$
16: $T_1 \leftarrow T_3$
17: $y_1 \leftarrow T_1 \times T_2$ (AC)
18: $T_3 \leftarrow T_3 \gg 1$ (A^2)
19: $x_1 \leftarrow x_1 + T_3$ (x_3)
20: $T_2 \leftarrow T_2 \gg 1$ (z_3)
21: $z_1 \leftarrow T_2$
22: $T_1 \leftarrow x$
23: $T_2 \leftarrow T_1 \times T_2$ (D)
24: $T_2 \leftarrow x_1 + T_2$ $(D + x_3)$
25: $T_1 \leftarrow y_1 + z_1$ $(AC + z_3)$
26: $y_1 \leftarrow T_1 \times T_2$ $((D + x_3)(AC + z_3))$
27: $T_1 \leftarrow x + y$
28: $T_2 \leftarrow z_1 \gg 1$
29: $T_1 \leftarrow T_1 \times T_2$ $((x + y)z_3^2)$
30: $y_1 \leftarrow y_1 + T_1$ (y_3)
31: **return** (x_1, y_1, z_1)

is computed with the following formulae [33]:

$$\begin{aligned} A &= y_1 + yz_1^2; \quad B = x_1 + xz_1; \quad C = Bz_1 \\ z_3 &= C^2; \quad D = xz_3 \\ x_3 &= A^2 + C(A + B^2 + aC) \\ y_3 &= (D + x_3)(AC + z_3) + (x + y)z_3^2 \end{aligned} \tag{8}$$

An algorithm can be constructed from (8) so that only two extra variables (T_1 and T_2) are needed. However, register sharing with the input registers of the multiplier requires that all multiplications are of the from $T_1 \times T_2$ which is not satisfied in this algorithm. Hence, if this algorithm was implemented instead of Algorithm 2, the register count would increase with three 163-bit registers (two input registers and z_1).

Algorithm 3 shows a process that requires three variables (T_1, T_2, and T_3) but where all multiplications are of the form $T_1 \times T_2$ which enables register sharing. Consequently, only two extra 163-bit registers (T_3 and z_1) are needed compared to affine coordinates. Achieving this requires eight register copies, e.g., on lines 5 and 6 of Algorithm 3. Comparing Algorithms 2 and 3 reveals that using affine coordinates results in a considerably simpler algorithm, especially, because using mixed coordinates would also require a routine for inversion in the end of the point multiplication (e.g., lines 1–19 in Algorithm 2). Consequently, the size of the control unit will be significantly larger if mixed coordinates are used instead of affine coordinates.

Using the latencies given in Section IV and the fact that a register copy can be done in two clock cycles, we can estimate that Algorithm 3 can be executed in 1,368 clock cycles. The exact latency can be a few clock cycles shorter because this estimate neglects the benefits that may be achievable for certain consecutive operations. The Frobenius maps should be computed in the order y_1^2, x_1^2, and z_1^2 and then one Frobenius map requires 7 clock cycles. The coordinate conversion in the end requires 1,968 clock cycles. These result in a point multiplication latency of approximately 76,100 clock cycles.

Acknowledgment

The first author would like to thank A. Azarderakhsh for the support and help during this research work. The authors would like to thank the reviewers for their constructive comments.

References

[1] S. Kumar, T. Wollinger, and C. Paar, "Optimum digit serial $GF(2^m)$ multipliers for curve-based cryptography," *IEEE Trans. Comput.*, vol. 55, no. 10, pp. 1306–1311, 2006.
[2] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A low-power high-performance concurrent fault detection approach for the composite field S-box and inverse S-box," *IEEE Trans. Comput.*, vol. 60, no. 9, pp. 1327–1340, 2011.
[3] M. Mozaffari Kermani and R. Azarderakhsh, "Efficient fault diagnosis schemes for reliable lightweight cryptographic ISO/IEC standard CLEFIA benchmarked on ASIC and FPGA," *IEEE Trans. Ind. Electron.*, vol. 60, no. 12, pp. 5925–5932, Dec. 2013.
[4] *Federal Information Processing Standards Publications*, (FIPS 186-3), U.S. Department of Commerce/NIST, 2009 [Online]. Available: csrc.nist.gov/publications/fips/fips186-3/fips186-3.pdf, Digital Signature Standards (DSS)
[5] S. Kumar and C. Paar, "Are standards compliant elliptic curve cryptosystems feasible on RFID?," in *Proc. Workshop RFID Security (RFIDSec 2006)*, 2006.
[6] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Low-cost elliptic curve cryptography for wireless sensor networks," in *Proc. Security and Privacy in Ad-Hoc and Sensor Networks*, 2006, pp. 6–17.
[7] Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-curve-based security processor for RFID," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1514–1527, 2008.
[8] D. Hein, J. Wolkerstorfer, and N. Felber, "ECC is ready for RFID-A proof in silicon," in *Proc. Workshop on Selected Areas in Cryptography (SAC 2009)*, 2009, pp. 401–413, Springer.
[9] U. Kocabas, J. Fan, and I. Verbauwhede, "Implementation of binary Edwards curves for very-constrained devices," in *Proc. 21st Int. Conf. Application-Specific Systems Architectures and Processors (ASAP 2010)*, 2010, pp. 185–191.
[10] V. Dimitrov and K. Järvinen, "Another look at inversions over binary fields," in *Proc. 21st IEEE Int. Symp. Computer Arithmetic (ARITH-21)*, 2013, pp. 211–218.
[11] A. Menezes, I. Blake, S. Gao, R. Mullin, S. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. New York, NY, USA: Kluwer, 1993.

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

[13] J. Massey and J. Omura, "Computational Method and Apparatus for Finite Arithmetic," U.S. Patent 4 587 627, May 6, 1986.

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

[15] 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, 2013.

[16] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal normal bases in $GF(p^n)$," *Discrete Appl. Math.*, vol. 22, no. 2, pp. 149–161, 1989.

[17] 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.

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

[19] A. Reyhani-Masoleh and M. A. Hasan, "A new construction of Massey-Omura parallel multiplier over $GF(2^m)$," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 511–520, 2002.

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

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

[22] A. Reyhani-Masoleh, "A new bit-serial architecture for field multiplication using polynomial bases," in *Proc. Cryptographic Hardware and Embedded Systems—CHES 2008*, E. Oswald, and P. Rohatgi, Eds., 2008, vol. 5154, ser. Lecture Notes in Computer Science, pp. 300–314.

[23] D. Hankerson, S. Vanstone, and A. Menezes*, Guide to Elliptic Curve Cryptography*. New York, NY, USA: Springer-Verlag, 2004.

[24] N. Koblitz, "CM-curves with Good Cryptographic Properties," in *Advances in Cryptology (CRYPTO 1991)*. New York, NY, USA: Springer, 1992, pp. 279–287.

[25] K. Järvinen, J. Forsten, and J. Skyttä, "Efficient circuitry for computing τ-adic non-adjacent form," in *Proc. 13th IEEE Int. Conf. Electron., Circuits Syst., (ICECS 2006)*, 2006, pp. 232–235.

[26] B. B. Brumley and K. U. Järvinen, "Conversion algorithms and implementations for Koblitz curve cryptography," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 81–92, 2010.

[27] J. A. Solinas, "Efficient arithmetic on Koblitz curves," *Des. Codes Cryptography*, vol. 19, pp. 195–249, March 2000.

[28] J. Adikari, V. Dimitrov, and K. Järvinen, "A fast hardware architecture for integer to τ-NAF conversion for Koblitz curves," *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 732–737, May 2012.

[29] M. A. Hasan, "Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems," *IEEE Trans. Comput.*, vol. 50, no. 10, pp. 1071–1083, 2001.

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

[31] J. López and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999)*, 1999, pp. 316–327.

[32] D. Bernstein, T. Lange, and R. Farashahi, "Binary Edwards curves," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008)*, 2008, vol. 5154, pp. 244–265.

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

[34] C. Rebeiro and D. Mukhopadhyay, "High speed compact elliptic curve cryptoprocessor for FPGA platforms," in *Progress in Cryptology, INDOCRYPT 2008*. New York, NY, USA: Springer-Verlag, 2008, vol. 5365, ser. Lecture Notes in Computer Science, pp. 376–388.

[35] C. Rebeiro, S. S. Roy, D. S. Reddy, and D. Mukhopadhyay, "Revisiting the Itoh-Tsujii inverions algorihtm for FPGA platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 8, pp. 1508–1512, 2011.

[36] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical modeling of the Itoh-Tsujii inversion algorithm for enhanced performance on k-LUT based FPGAs," in *Proc. Design, Automation & Test in Europe (DATE 2011)*, 2011, pp. 1–6.

[37] K. Järvinen, "On repeated squarings in binary fields," in *Proc. the 16th International Workshop on Selected Areas in Cryptography (SAC 2009)*. New York, NY, USA: Springer-Verlag, 2009, vol. 5867, ser. Lecture Notes in Computer Science, pp. 331–349.

[38] M. Joye and C. Tymen, "Compact encoding of non-adjacent forms with applications to elliptic curve cryptography," in *Public Key Cryptography—PKC 2001*. New York, NY, USA: Springer-Verlag, 2001, vol. 1992, ser. Lecture Notes in Computer Science, pp. 353–364.

[39] STMicroelectronics [Online]. Available: www.st.com

[40] Synopsys [Online]. Available: www.synopsys.com

[41] R. Azarderakhsh and A. Reyhani-Masoleh, "High-performance implementation of point multiplication on Koblitz curves," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 1, pp. 41–45, 2013.

[42] O. Ahmadi, D. Hankerson, and F. Rodriguez-Henrìquez, "Parallel formulations of scalar multiplication on Koblitz curves," *J. Univers. Comput. Sci.*, vol. 14, no. 3, pp. 481–504, 2008.

[43] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "A parallel architecture for Koblitz curve scalar multiplications on FPGA platforms," in *Proc. 15th Euromicro Conf. Digital Syst. Des. 2012*, 2012, pp. 553–559.

[44] K. Järvinen, "Optimized FPGA-based elliptic curve cryptography processor for high-speed applications," *Integr. VLSI J.*, vol. 44, no. 4, pp. 270–279, 2011.

[45] K. Järvinen and J. Skyttä, "On parallelization of high-speed processors for elliptic curve cryptography," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 9, pp. 1162–1175, 2008.

[46] V. S. Dimitrov, K. U. Järvinen, M. J. Jacobson, Jr., W. F. Chan, and Z. Huang, "Provably sublinear point multiplication on Koblitz curves and its hardware implementation," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1469–1481, 2008.

[47] J. Singh*, Semiconductor Devices: Basic Principles*. New York, NY, USA: Wiley, 2001.

[48] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology, CRYPTO '96*. New York, NY, USA: Springer-Verlag, 1996, vol. 1109, ser. Lecture Notes in Computer Science, pp. 104–113.

[49] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology, CRYPTO '99*. New York, NY, USA: Springer-Verlag, 1999, vol. 1666, ser. Lecture Notes in Computer Science, pp. 388–397.

[50] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *Cryptographic Hardware and Embedded Systems, CHES 2002*. New York, NY, USA: Springer-Verlag, 2002, vol. 2523, ser. Lecture Notes in Computer Science, pp. 29–45.

[51] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proc. IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

Reza Azarderakhsh received the B.Sc. degree in electrical and electronic engineering in 2002, the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2005, and the Ph.D. degree in electrical and computer engineering from Western University, London, ON, Canada, in 2011.

He joined the Department of Electrical and Computer Engineering, Western University, as a Limited Duties Instructor, in September 2011. He has been an NSERC Post-Doctoral Fellow with the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada. As of 2013, he has been also an Assistant Professor with the Department of Computer Engineering at Rochester Institute of Technology, Rochester, NY, USA. His current research interests include finite field and its application, elliptic curve cryptography, and pairing based cryptography.

Dr. Azarderakhsh was a recipient of the prestigious Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellowship in 2013.

Kimmo U. Järvinen received the M.Sc. (Tech) degree in 2003 and the D.Sc. (Tech) degree in 2008, both in electrical engineering, from the Helsinki University of Technology (TKK), Finland.

He was with the Signal Processing Laboratory at TKK from 2002 to 2008. Since June 2008, he has been with the Cryptology Group in the Department of Information and Computer Science, Aalto University School of Science. Since the beginning of 2011, he has had a respected three-year postdoctoral researcher's project funded by the Academy of Finland. His research interests include applied cryptography, cryptographic engineering (especially, hardware realizations of cryptographic algorithms), general computer arithmetic, and FPGAs.

Mehran Mozaffari-Kermani (M'11) received the B.Sc. degree in electrical and computer engineering from the University of Tehran, Tehran, Iran, in 2005, and the M.E.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, Western University, London, ON, Canada, in 2007 and 2011, respectively.

He joined the Advanced Micro Devices as a Senior ASIC/Layout Designer, integrating sophisticated security/cryptographic capabilities into a single accelerated processing unit. In 2012, he joined the Electrical Engineering Department, Princeton University, Princeton, NJ, USA, as an NSERC Post-Doctoral Research Fellow. As of 2013, he has been an Assistant Professor with the Department of Electrical and Microelectronic Engineering at Rochester Institute of Technology, Rochester, NY, USA. His current research interests include emerging security/privacy measures for deeply embedded systems, cryptographic engineering, fault diagnosis and tolerance in cryptographic hardware and embedded systems, VLSI reliability, and low-power secure and efficient FPGA and ASIC designs.

Dr. Mozaffari-Kermani is also a member of the IEEE Computer Society. He was a recipient of the prestigious Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellowship in 2011.