A Scalable ECC Processor Implementation for High-Speed and Lightweight with Side-Channel Countermeasures

Ahmad Salman, Ahmed Ferozpuri, Ekawat Homsirikamol, Panasayya Yalla, Jens-Peter Kaps, Kris Gaj
Cryptographic Engineering Research Group (CERG), Department of Electrical and Computer Engineering
George Mason University Fairfax, Virginia, USA, http://cryptography.gmu.edu
{asalman, aferozpu, ehomsiri, pyalla, jkaps, kgaj}@gmu.edu

***Abstract*—The performance of Public Key Cryptosystems (PKC) based on elliptic curves is mostly dependent on the performance of the underlying field arithmetic. In this work, we present high-speed and lightweight implementations of a fully scalable architecture of an Elliptic Curve Cryptography (ECC) scalar multiplier processor. The processor supports operations over GF(p) for arbitrary values of p, and field sizes up to 521 bits. The implementations perform modular multiplication operations using fully scalable Montgomery multiplier architectures, one tailored for high-speed and one for lightweight. Both designs support different bus widths to increase flexibility and allow for a wide range of applications. Our cores include countermeasures to side-channel attacks by using the Montgomery Ladder and Exponent Randomization methods to provide resistance to Simple Power Analysis (SPA) and Differential Power Analysis (DPA) respectively. We implemented the designs on FPGA platforms from different vendors and evaluated them based on NIST recommended field lengths - 192, 224, 256, 384, and 521 bits - using several arbitrary values of prime p.**

I. Introduction

Elliptic Curve Cryptography (ECC) is considered one of the most widely used Public Key Cryptosystems (PKC). Although introduced by Koblitz [1] and Miller [2] independently in 1985, the National Institute of Standards and Technology included ECC as a public key standard only in the late 90's [3]. FIPS PUB 184-6 recommends the use of two different types of elliptic curves with certain parameter recommendations for each type. The curves are classified based on the underlying field into elliptic curves over prime fields $GF(p)$ and elliptic curves over binary fields $GF(2^m)$ with the later having a special type of curves known as Koblitz curves. The main advantage for using ECC over other PKCs such as RSA [4] is that it requires shorter key lengths to provide the same level of security which decreases the amount of memory required when implemented in software or hardware, and allows for faster generation of keys, and faster digital signatures. For these reasons, ECC has been a popular PKC choice for high-speed as well as lightweight applications.

In this paper, we present a case study of implementing two ECC processors in hardware, one suitable for high-speed applications and the other tailored for lightweight ones. The main difference between the two designs is in the Modular Multiplication unit where we use two different architectures to perform multiplication in the Montgomery domain [5]. Our ECC processors perform ECC operations over GF(p) but unlike other implementations over GF(p), they are not limited to NIST primes but rather generalized to perform ECC operations over any curve over GF(p) with any field length and any prime p. Our implementations do not make use of specialized hardware components such as Digital Signal Processing (DSP) or multiplier units which makes them the only high-speed and lightweight implementations, to the best of our knowledge, suitable for Field Programmable Gate Arrays (FPGAs) as well as Application Specific Integrated Circuits (ASICs). The high-speed ECC processor supports multiple word sizes and the lightweight design is scalable in area by using a variable number of Processing Elements (PEs) in the multiplication unit. The security of a cryptosystem also depends on the implementation itself, hence we employed countermeasures against side-channel attacks making the implementation resistant to SPA and DPA attacks. We implemented the design on FPGA and Programmable System on Chip platforms from several vendors and also analyzed power and energy consumptions for each implemented design to determine the relation between them and the area/throughput trade-off.

II. Background

An elliptic curve E over GF(p) is a set of points fulfilling the equation of the curve. When points are represented in affine coordinates (x, y), the equation of the curve is given in the Weierstrass form: $y^2 = x^3 + ax + b$, where a and b are parameters of the curve, belonging to GF(p). Two points of the curve, P and Q, can be added resulting in the third point $P = P + Q$. This operation is known as "point addition". A single point, P, can be doubled, giving $P = 2P$. This operation is known as "point doubling". Scalar multiplication kP is defined as $kP = P + P + \cdots + P$, where k is a natural number, and P appears on the right side of the equation k times. Scalar multiplication is performed using repetitive point addition and point doubling operations as shown in Algorithm 1. Point addition and point doubling operations using affine coordinates representation can be performed as follows:

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ then the new point $P = P + Q = (x_3, y_3)$ would be calculated according to the following equations:

978-1-5386-3797-5/17/$31.00

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1$$

where

$$\lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

Let $P = (x_1, y_1)$ then the new point $P = 2P = (x_3, y_3)$ would be calculated according to the following equations:

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1$$

where

$$\lambda = \frac{(3x_1^2 + a)}{(2y_1)}$$

From the previous equations it can be seen that the calculations of both point addition and point doubling operations require a division operation to calculate λ which is very costly for large operands. Projective coordinate representations of points are preferred to perform ECC calculations which does not require division operations.

A base point of the curve E, denoted as $G = (G_x, G_y)$, is a generator of a subgroup of E, of the prime order n, consisting of all points of the form, iG, including a special point, called point at infinity, denoted as O, which serves as a natural element for point addition. For the base point G, $nG = O$. The number of points on the curve is hn, for some integer h, known as the cofactor, which is not divisible by n.

FIPS PUB 186-4 [3], defines the elliptic curve equation over prime fields as $E : y^2 = x^3 - 3x + b$, fixing the value of the coefficient a to $-3 \equiv p - 3$ for efficiency optimization purposes. The standard also recommends that the value of the cofactor h should be as small as possible and fixes it to h = 1 for prime fields. Lastly, the standard recommends specific values of the prime p for each of the five recommended field sizes. The general way to calculate the scalar multiplication is by using the Double-and-Add Algorithm 1. The downside of this method is that it makes the design vulnerable to SPA [6], hence it is insecure. To prevent such an attack, different approaches can be taken such as using the Double-and-Add-Always algorithm [7] or the Montgomery Ladder Algorithm 2.

Meloni [8] explains how point addition can be accelerated if both points P and Q share the same Z coordinate. In order to make sure that both points maintain a common Z (co-Z), he introduced the addition with update (ZADDU) formula to calculate point addition fast and showed how to calculate the scalar multiplication using Fibonacci-and-add algorithm. Goundar et al. [9] showed that with the new co-Z addition formula, point subtraction comes for free and explained that point doubling can be expressed as, point addition followed by point subtraction as $Q = P+Q, \quad R = P-Q, \quad P = R+Q$. The first step can be calculated using ZADDU and the last two steps can be expressed using the add with conjugate (ZADDC) formula. Using these two formulas in addition to a single use of a double-with-update (DBLU) formula to initialize P, the Montgomery Ladder algorithm can be efficiently calculated using Algorithm 3. We use it in our cores to calculate the scalar multiplication. Modular multiplication and Modular addition steps required to calculate ZADDU, ZADDC, and DBLU can be found in [9]. Additionally, our cores are not limited to the NIST recommendations listed above. The cofactor a can be set to any value that belongs to GF(p), the cofactor h can be of values other than 1, and the prime p can be of any prime value of the length of the field size.

Algorithm 1 Calculating the Scalar Multiplication Operation Using Double-and-Add Algorithm

Input: Prime $p \in E(\mathbb{F}_q)$, $P = (x, y)$, where $x, y \in GF(p)$
$k \in \mathbb{Z}, 0 < k < \#E$, $k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
Output: $Q = (x', y')$
$Q = P$
for $i = l - 2$ **downto** 0 **do**
$Q = 2Q$
if $k_i = 1$ **then**
$Q = Q + P$
return Q

Algorithm 2 Calculating the Scalar Multiplication Operation Using Montgomery Ladder Algorithm

Input: Prime $p \in E(\mathbb{F}_q)$, $P = (x, y)$, where $x, y \in GF(p)$
$k \in \mathbb{Z}, 0 < k < \#E$, $k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
Output: $Q = (x', y')$
$Q = P$
$P = 2Q$
for $i = l - 2$ **downto** 0 **do**
if $k_i = 1$ **then**
$Q = Q + P$
$P = 2P$
else
$P = P + Q$
$Q = 2Q$
return Q

Algorithm 3 Calculating the Montgomery Ladder Algorithm with co-Z addition formulas

Input: Prime $p \in E(\mathbb{F}_q)$, $P = (X, Y, Z)$,
where $X, Y, Z \in GF(p)$, $k \in \mathbb{Z}, 0 < k < \#E$,
$k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
Output: $Q = (X', Y', Z)$
$P, Q = DBLU(P)$
for $i = l - 2$ **downto** 0 **do**
if $k_i = 1$ **then**
$Q, P = ZADDC(P, Q)$
$P, Q = ZADDU(Q, P)$
else
$P, Q = ZADDC(Q, P)$
$Q, P = ZADDU(P, Q)$
return Q

Coron [7] described a few methods to prevent DPA attacks. His first method, Randomization of the Private Exponent, calculates $Q = k'P$ instead of $Q = kP$ where $k' = k + d \cdot \#E$, d is a small random number (20-bit) and $\#E = hn$ is the

total number of points on the curve E. Based on the fact that multiplying a point P on a curve E with the total number of points on this curve $\#E$ will result in O, the result of $k'P = kP = Q$. Changing the value of k each time the scalar multiplication performed using this method will make the DPA [10] attack infeasible.

III. Previous Work

ECC has been one of the most researched topics in public key cryptography. On the implementation side, Alrimeih et al. [11] implemented a hardware/software co-design of an ECC processor to perform the scalar multiplication over GF(p). Their implementation supports all five prime fields recommended by NIST, but is also limited to and optimized for their corresponding primes by using special multiplication circuits dedicated to speed-up NIST primes. Amiet et al. [12] implemented a generalized ECC processor which supports all five of NIST prime fields for any prime p. Their implementation uses two Montgomery multiplier units which work in parallel with a Modular adder/subtractor to perform the pre-scheduled scalar multiplication operations. They also use DSP units to increase the performance.

Sasdrich and Güneysu [13] implemented a hardware accelerator for ECC point multiplication which is limited to Curve25519 using pseudo Mersenne primes. Their work was expanded later to include techniques for countermeasures against SPA and DPA attacks. Roy et al. [14] designed a lightweight ECC accelerator that makes use of DSP units on FPGAs to perform optimized modular reduction over NIST p-256 prime curves while minimizing area usage. Baldwin et al. [15] performed a comprehensive hardware, software and hardware/software co-design implementations for different ECC multiplication algorithms that use co-Z formulas.

Örs et al. [16] introduce a module-based design for an ECC processor over GF(p). The architecture is suitable for any prime field and any prime. The design uses Montgomery in a systolic array architecture. We mainly based our designs on this last architecture, however we use a different architecture for the Montgomery multiplier and apply counter measures for side channel attacks. We also use projective coordinates instead of modified Jacobean coordinates and make use of co-Z addition formula described in the previous section.

One of the most computationally intensive operations in ECC is modular multiplication. Using an optimized multiplier to perform this operation is essential to improving the performance of an ECC processor. Tenca and Koç [17] introduced a word-based algorithm for Montgomery multiplication, called Multiple-Word Radix-2 Montgomery Multiplication (MWR2MM), as well as a scalable hardware architecture capable of performing the multiplication operation using a variable number of PEs. Huang et al. [18] proposed two architectures, upon which we based our multiplier implementations, to optimize the original MWR2MM algorithm to process n-bit precision multiplication in approximately n clock cycles by precomputing intermediate S values. While implementing, we discovered small errors in [18] Algorithm 2 and 5. We show them in this paper as Algorithm 4 with the correction in line 8. Another small error was discovered in [18] Algorithm 5 which we show here as Algorithm 5 with the corrections in the Input and line 1.

Algorithm 4 Corrected Multiple-Word Radix-2 Montgomery Multiplication Algorithm [18] [17]

Input: odd $M, n = \lfloor \log_2 M \rfloor + 1$, word size w, $e = \lceil \frac{n+1}{w} \rceil$, $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $Y = \sum_{j=0}^{e-1} Y^{(j)} \cdot 2^{w \cdot j}$, $M = \sum_{j=0}^{e-1} M^{(j)} \cdot 2^{w \cdot j}$, with $0 \leq X, Y < M$
Output: $Z = \sum_{j=0}^{e-1} S^{(j)} \cdot 2^{w \cdot j} = \mathrm{MP}(X, Y, M) \equiv X \cdot Y \cdot 2^{-n} \pmod{M}$, $0 \leq Z < 2M$
1: $S = 0$
2: **for** $i = 0$ **to** $n - 1$ **do**
3: $\quad q_i = (x_i \cdot Y_0^{(0)}) \oplus S_0^{(0)}$
4: $\quad (C^{(1)}, S^{(0)}) = x_i \cdot Y^{(0)} + q_i \cdot M^{(0)} + S^{(0)}$
5: $\quad$ **for** $j = 1$ **to** e **do**
6: $\quad\quad (C^{(j+1)}, S^{(j)}) = C^{(j)} + x_i \cdot Y^{(j)} + q_i \cdot M^{(j)} + S^{(j)}$
7: $\quad\quad S^{(j-1)} = (S_0^{(j)}, S_{w-1\ldots1}^{(j-1)})$
8: $\quad S^{(e)} = (0, S_{w-1\ldots1}^{(e)})$
9: **return** $Z = S$

Algorithm 5 Corrected Computations in Task F [18]

Input: $q_i, x_i, C^{(e-1)}, Y^{(e-1)}, M^{(e-1)}, S_{w-1\ldots1}^{(e-1)}, C^{(e)}$
Output: $C^{(e)}, S_{w-1\ldots1}^{(e-1)}, S_0^{(e-1)}$
1: $(C^{(e)}, S^{(e-1)}) = (C^{(e)}, S_{w-1\ldots1}^{(e-1)}) + C^{(e-1)} + x_i \cdot Y^{(e-1)} + q_i \cdot M^{(e-1)}$

IV. Proposed Design

We used a modular approach which is divided into four main units: Scheduler, Memory Controller (Mem_Ctrl), Modular Adder Subtracter (MAS) and Modular Montgomery Multiplier (MMM) and a FIFO interface (see Fig. 1) to supply inputs and read back the output data. External memory is used to store intermediate results during the calculation of the scalar multiplication. The memory is interfaced to the processor through the Mem_Ctrl unit. The top level is similar for both high-speed and lightweight designs except for the connection between the MMM and MAS units which is absent from the lightweight design which we explain in Section V.

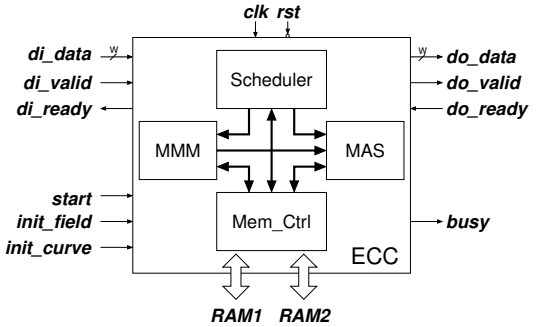

Fig. 1: Top Level Architecture

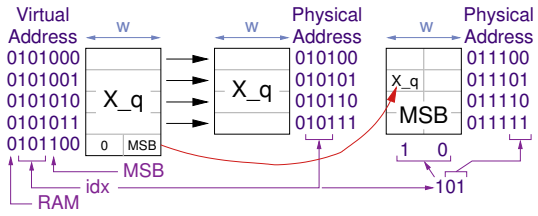

Fig. 2: Example of how the 9 MSB Bits are Stored

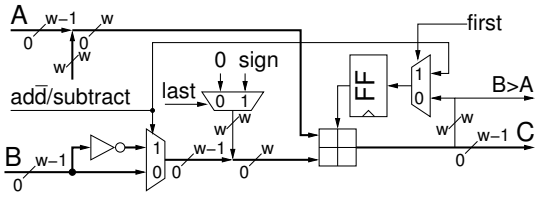

Fig. 3: Modular Adder/Subtractor (MAS) Module

Scheduler: The scheduler is the main controller unit of the ECC processor, which is composed of four high-level states: Normal to Montgomery Conversion, Scalar Multiplication, Projective to Affine Conversion, and Montgomery to Normal Conversion. Each state has its own controller making the design modular. A *start* signal triggers a state to begin operation and hands control back to the scheduler by returning a *done* signal. The Scalar Multiplication state uses the Montgomery ladder method to prevent SPA attacks during the calculation of kP. Our design makes use of Algorithm 3 to perform the point addition and point doubling operations. The processor uses an instruction ROM to implement the point doubling and point addition algorithms. The ROM is composed of 27-bit instructions broken down as follows

- 12-bit for multiplier operands (4-bit each).
- 12-bit for adder operands (4-bit each).
- 2-bit to select multiply, add, or both.
- 1-bit to select between addition and subtraction.

Memory and Memory Controller: Our designs require 17 operands, including temporary values, to be stored with the maximum size of 521-bit each. For that, we use two memories, Memory-1 (16x512 bits) to store 14 operands and Memory-2 (4x512 bits) to store 3 operands. The remaining 512-bits in each memory are used to store and pack the remaining 9 MSB bits of the 521-bit operands when using 64-bit word size. The Mem_Ctrl handles the sorting of these bits by converting a virtual address issued from the scheduler to a physical address where operands can be read or written. Fig. 2 shows an example of how operand X_q, which has an index of 5 according to the memory map, is stored in memory.

Modular Adder-Subtracter (MAS): All supported field sizes, with the exception of 521 are divisible by 16 and 32. Storing them in word-size memory does not leave space for the sign bit. To solve this problem, we created our MAS unit as shown in Fig. 3. All words of operands A and B are expanded by 1-bit $(w+1)$. When performing a subtraction operation on $e(w_{e-1}\dots w_0)$ words, operand B is inverted and the C_{in} is set to '1' by setting the *add/subtract* and *first* select inputs to be '1' for word w_0 which makes the addition operation equivalent to a subtraction in 2's complement. For all subsequent words until word w_{e-2} *first* is set to '0' making C_{out} from the previous word addition go in as C_{in} to the current word addition. For word w_{e-1}, *last* is set to '1' to provide the sign bit. If the addition operation results in a positive result, then the result C is sent to memory w-bits per clock cycle. If the result is negative, then operand B is recalled from memory as operand A and the modulus M is added as operand B. This puts the value back in the positive domain.

Modular Montgomery Multiplier (MMM): We use the algorithm from [18] to calculate the Modular Montgomery Multiplication in our MMM module. Our design uses architectures 1 and 2 for lightweight and high-speed implementations respectively which are explained in detail in the next section. The calculations for task D, E, and F are described in [18].

V. Implementations

A. Implementation Decisions

NIST *special curves* are those whose coefficients and underlying field have been selected to optimize the efficiency of the elliptic curve operations while NIST *special primes* are of a special type (called generalized Mersenne numbers) for which modular multiplication can be carried out more efficiently than for general primes. Our designs are generalized for all $GF(p)$ curves for a specified field size and not limited to NIST curves. We did not include binary fields $GF(2^m)$ as recent improvements in attacking discrete logarithms over small-characteristic fields raised security concerns about them, although these concerns apply only to pairings for the time being. Our design utilizes external memory in order to support ASIC implementations, easy mapping to embedded memories on FPGAs, and unified high-speed and lightweight storage requirements. We used projective coordinate representation with Co-Z arithmetic which calculates the Montgomery Ladder faster compared to other coordinate systems. Our designs support all 5 NIST field sizes and are not limited to special primes as these primes might be patent restricted. We are not making use of special FPGA features such as DSP units to make the designs also suitable for ASICs.

The high-speed design uses different word sizes (16, 32, and 64) and redundant representation to achieve high throughput. The lightweight design uses a variable number of PEs (2, 4, or 8) to increase flexibility while maintaining a low area. For the high-speed design we used Carry-Save Adders (CSA) inside the PEs and save the result in redundant format which is then sent to the MAS unit to be added to get the final Montgomery Multiplication result. This way we can free the multiplier faster to start the next Montgomery multiplication operation which increases the throughput. In lightweight, the PEs use Carry Propagate Adders (CPA) and the result is saved in non-redundant format which does not require a final addition operation hence there is no direct connection between MMM and MAS modules.To protect against DPA, we chose to randomize the scalar k using the "Randomization

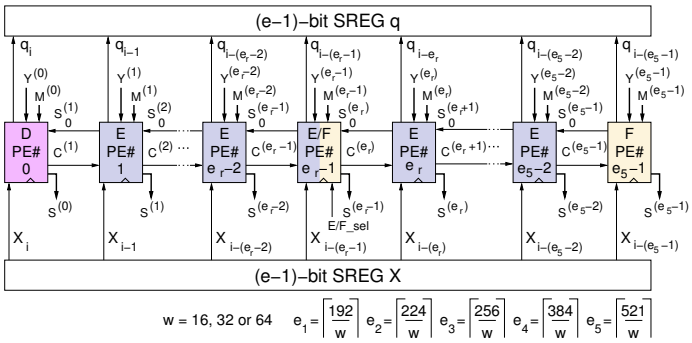

Fig. 4: High-Speed Design Supporting Different Field Sizes

of the Private Exponent" method from [7]. This provides the needed security without adding to the overhead of the scalar multiplication calculation as the randomization factor can be calculated through software in advance and it only needs to be updated whenever the curve parameters are changed.

B. High-Speed Implementation

Architecture 2 consists of p PEs forming a computation chain. Each PE works on computing a specific word of the final Montgomery multiplication result S (i.e. PE#j computes $S^{(j)}$). The maximum number of PEs in the design is the same as the number of words, e, of the operands which can be calculated according to the following formula $e = \lceil \frac{n}{w} \rceil$ where n is the operand size and w is the word size. For our design to support all target field sizes, n needs to be of the same size as the maximum supported size (521-bit). In order to achieve maximum throughput, we utilized the maximum number of PEs by choosing $p = e$ =33, 17, and 9 for w = 16, 32, and 64 respectively. The PEs are defined by the task they perform (D, E, or F). PE#0 is used for task D calculations while PE#(e-1) is used to perform task F. The remaining PEs (PE#1 ...PE#$(e-2)$) calculate task E. Operand Y and the modulus M are scanned word-by-word and are fixed for a given PE while operand X is scanned bit-by-bit. It can also be noticed that this architecture requires $T = n+(e-1)$ clock cycles to finish one Montgomery multiplication. Reducing w by half for a given n does not affect T by much. To support all 5 field sizes within a single design of architecture 2, we created a new PE (E/F) capable of performing both tasks E and F. The new PE is placed at positions e_r where r = 1, 2, 3, and 4 as shown in Fig. 4. This way, if the field is less than 521-bit, the E/F_sel signal for the corresponding PE#$(e_r - 1)$ is set to function as type F.

C. Lightweight Implementation

In the lightweight design, every PE can perform tasks D, E, and F. In order to reduce area, we limited p to be 2, 4, or 8 regardless of the size of n and fixed w to 16-bit. This limited number of PEs required us to store the intermediate values of S in a queue Q as described in [18]. The size of $Q = e - p$ and the number of clock cycles T is now calculated as $T = n+\left\lceil \frac{n}{p} \right\rceil \cdot (e-p)+p+1$. Like architecture 2, architecture 1 scans operand Y and modulus M word-by-word while operand X

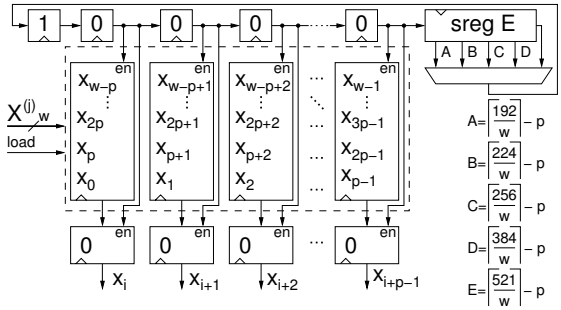

Fig. 5: Reading and Formating of X

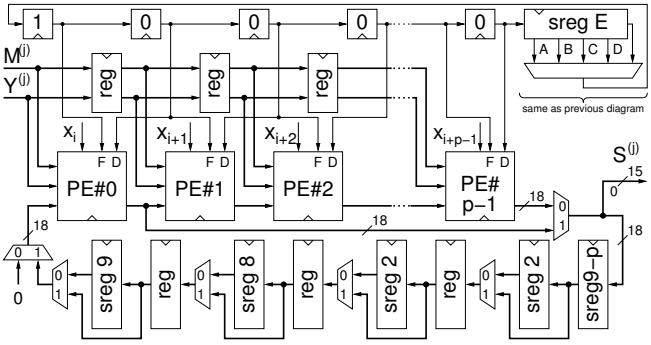

Fig. 6: Main Computational Unit

is read bit-by-bit. However, in architecture 1 the value of X is fixed for a PE until it is multiplied by every word of Y and all the words of the result S are produced from PE#p. Reducing the size of p by half leads to a penalty $\simeq 2T$, a slight increase in the size of Q, and reduces the area of PEs by half. To minimize the internal storage of the design, X, Y, and M are called from memory one word at a time. X is updated every w clock cycles while Y and M are updated every clock cycle. The maximum number of words available for processing at a given clock cycle are 1 word of X and p words of Y and M. In order to supply the correct bit of X to its corresponding PE, the X word is stored in four 4-bit registers with the enable signals for these registers controlled by the shift register $sregE$ that has variable shifts as shown in Fig. 5. Y and M are passed from one PE to the next one clock cycle at a time and the intermediate Ss are stored in Q. The values in Q are shifted using variable shifts controlled by 2-to-1 multiplexers depending on the size of n as shown in Fig. 6. Finally, the internal structure of the PE capable of performing Tasks D, E, and F is shown in Fig. 7.

VI. Results and Discussion

We implemented two different ECC processors, one suitable for high-speed applications and the other for lightweight devices. Additionally we created high-speed and lightweight versions of the ECC processor that uses the fast, but vulnerable to SPA, Double-and-Add algorithm and Modified Jacobean coordinates to compare our protected ECC processor to faster unprotected implementations. Our test setup was as follows:

- Embedded memories used only for "external RAM".

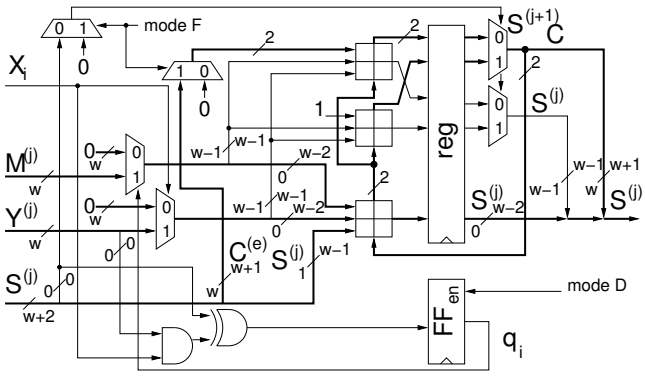

Fig. 7: Inside the PE Unit of the Lightweight Design

- All implementations are coded in VHDL and do not use any other embedded resources.
- Implemented using Xilinx ISE 14.7, Quartus Prime 16.0 and Libero SoC 11.7 and Optimized using ATHENa [19].
- All results reported are Post-Place & Route.

The latencies for different field sizes supported by the ECC processor are listed in Tables I and II for high-speed and lightweight designs respectively. The latency is defined as the number of clock cycles required for actual computation of the scalar multiplication excluding the field and curve initialization phases. Since the latency depends on a particular value of k, we also assume that k is of the same size as the operating field and has equal number of 1's and 0's in its binary representation. Since the latency includes also the time spent for receiving inputs P_x, P_y and k, and sending out outputs Q_x and Q_y, the assumption is made that these transmission do not include any stall cycles. All latency values shown in Table I have been determined using simulations for each word size w = 64, 32, or 16 for the high-speed designs and the values shown in Table II have been determined using simulations for each number of PEs p = 2, 4, or 8 for the lightweight. The throughput is calculated in terms of the number of operations (scalar multiplications) per second, assuming a hypothetical clock frequency of 100 MHz. The results show that the overhead of using Montgomery Ladder with co-Z addition formulas is very low. This is because ZADDU and ZADDC algorithms are faster than other projective coordinates point doubling and point addition algorithms.

Implementation results for the high-speed design on different families are summarized in Table III. The best performance on all target devices in terms of throughput and throughput/area ratio was recorded for $w = 32$ which is the middle value of our supported widths. The only exception to this is the Virtex-7 implementation where $w = 16$ has the best throughput/area ratio. The throughput when $w = 64$ is better than when $w = 16$ except for the Stratix-IV device. While the throughput/area ratio when $w = 16$ is better than when $w = 64$ except for the Zynq implementation. For the lightweight, the best performance was also consistent across all target devices and it was recorded for $p = 8$ as can be seen in Table IV. The number of PEs is directly proportional with

TABLE I: Latency and Throughput for a Given Field and Width for High-Speed

	Field size	Latency in clock cycles Size of word (W)			TP in Op/sec at f=100 MHz		
		W=64	W=32	W=16	W=64	W=32	W=16
Unprotected	192-bit	766,476	868,229	1,071,735	130	115	93
	224-bit	1,045,978	1,164,717	1,431,804	96	86	70
	256-bit	1,296,709	1,462,177	1,793,113	77	68	56
	384-bit	2,910,581	3,263,447	3,969,182	34	31	25
	521-bit	5,398,490	6,017,514	7,255,754	19	17	14
	Average	**2,283,646**	**2,555,216**	**3,104,317**	**71**	**63**	**52**
Protected	192-bit	824,212	939,969	1,171,483	121	106	85
	224-bit	1,125,214	1,260,229	1,564,664	89	79	64
	256-bit	1,395,369	1,584,853	1,963,821	72	63	51
	384-bit	3,123,729	3,528,699	4,338,639	32	28	23
	521-bit	5,791,134	6,502,510	7,925,454	17	15	13
	Average	**2,451,932**	**2,763,252**	**3,392,812**	**66**	**59**	**47**

TP→ Throughput; Op→ Operations; f → Frequency

TABLE II: Latency and Throughput for a Given Field and Number of PE Units for Lightweight

	Field size	Latency in clock cycles Number of PE units (#PE)			TP in Op/sec at f=100 MHz		
		#PE=8	#PE=4	#PE=2	#PE=8	#PE=4	#PE=2
Unprotected	192-bit	1,477,451	2,400,451	4,265,951	68	42	23
	224-bit	2,212,011	3,684,471	6,652,161	45	27	15
	256-bit	3,073,655	5,213,351	9,518,015	33	19	11
	384-bit	9,453,251	16,857,851	31,705,751	11	6	3
	521-bit	22,890,391	41,810,938	79,687,348	4	2	1
	Average	**7,821,351**	**13,993,412**	**26,365,845**	**32**	**19**	**11**
Protected	192-bit	1,576,957	2,557,883	4,540,489	63	39	22
	224-bit	2,358,523	3,922,551	7,074,793	42	25	14
	256-bit	3,279,973	5,555,813	10,134,373	30	18	10
	384-bit	10,057,513	17,916,721	33,676,213	10	6	3
	521-bit	24,313,723	44,374,347	84,533,037	4	2	1
	Average	**8,317,338**	**14,865,465**	**27,991,781**	**30**	**18**	**10**

TP→ Throughput; Op→ Operations; f → Frequency

throughput and throughput/area ratio.

It was not easy to compare our results to other published work in the area as not too many designs support multiple field sizes in the same ECC processor and the ones that do, are mostly limited to NIST primes with special optimizations applied making direct comparison not very accurate. As shown in Table V, we chose the designs that target the Virtex-5, Virtex-6, and Virtex-7 families and compared them to our Virtex-7 implementations. The closest design to compare our design to is the work from [12]. It should be noticed that their design is more than twice as large as ours, they use two Montgomery multiplier units, use special DSP units and multipliers and they use the fast and SPA vulnerable, Double-and-Add algorithm. The design from [11] is almost ten times as large as ours with 128 BRAMs used compared only 4 in our work. Their design achieves high throughput but the throughput/area ratio is only slightly higher than our design. The work in [20] only reports results for GF(p) 192-bit and 256-bit, their design is more than twice as large as ours. The Throughput/area ratio for their 192-bit design has a better performance than ours while the 256-bit implementation is worse which indicates that as the field size increases in their design the size gets larger and the performance will drop even more. Other results such as [21], [22], [23], [24], and [13] are implemented on families with different technology and therefore cannot be compared to our design.

We compare our lightweight design with $p = 8$ design to

TABLE III: Implementation Results of Protected High-Speed Design

Width	Slices	LUTs	FFs	BRAMs	Avg. Latency [$Clock\ cycles$]	f [MHz]	TP $[\frac{Op}{sec}]$	TP/Area $[\frac{Op}{Slices \cdot sec}]$
Xilinx FPGAs								
Virtex7:xc7vx485tffg1761-3								
64	1,269	3,036	4,678	8	2,451,931	184	75	0.059
32	1,227	2,681	4,572	4	2,763,252	214	77	0.063
16	996	3,144	4,606	2	3,392,812	243	72	0.072
Virtex6:xc6vlx240tff1156-3								
64	1,316	3,041	4,678	8	2,451,931	175	71	0.054
32	1,110	2,886	4,572	4	2,763,252	212	77	0.069
16	1,174	2,840	4,606	2	3,392,812	239	70	0.060
Zynq:xc7z020clg484-3								
64	1,135	3,072	4,678	8	2,451,931	130	53	0.047
32	1,172	2,720	4,572	4	2,763,252	168	61	0.052
16	1,224	2,765	4,606	2	3,392,812	178	52	0.043
Altera FPGAs								
Width	**ALMs**	**ALUTs**	**FFs**	**MBits**	**Avg. Latency** [$Clock\ cycles$]	f [MHz]	**TP** $[\frac{Op}{sec}]$	**TP/Area** $[\frac{Op}{ALMs \cdot sec}]$
Stratix V:5SGXEA7K2F40C3								
64	2,998	5,660	5,383	20,480	2,451,931	173	70	0.023
32	2,766	5,076	5,113	20,480	2,763,252	212	76	0.028
16	2,629	4,841	5,084	20,480	3,392,812	237	70	0.026
Stratix IV:EP4SE530H35C4								
64	4,312	3,936	4,687	20,480	2,451,931	134	55	0.013
32	3,731	4,076	4,582	20,480	2,763,252	176	64	0.017
16	3,681	4,031	4,617	20,480	3,392,812	191	56	0.015
Actel FPGAs								
Width	**LEs**	**4LUTs**	**FFs**	**RAM**	**Avg. Latency** [$Clock\ cycles$]	f [MHz]	**TP** $[\frac{Op}{sec}]$	**TP/Area** $[\frac{Op}{LEs \cdot sec}]$
IGLOO2:M2GL010TS-1FG484								
64	7,846	6,737	5,226	16	2,451,931	75	30	0.004
32	7,017	6,007	4,962	12	2,763,252	84	30	0.004
16	6,839	5,927	4,926	10	3,392,812	93	27	0.004

TABLE IV: Implementation Results of Protected Lightweight Designs

# of PEs	Slices	LUTs	FFs	BRAMs	Avg. Latency [$Clock\ cycles$]	f [MHz]	TP $[\frac{Op}{sec}]$	TP/Area $[\frac{Op}{Slices \cdot sec}]$
Xilinx FPGAs								
Zynq:xc7z020clg484-3								
8	469	1,697	1,169	2	8,317,338	179	21	0.046
4	407	1,353	1,015	2	14,865,465	182	12	0.030
2	318	1,118	939	2	27,991,781	164	6	0.020
Artix:xc7a100tcsg324-3								
8	527	1,675	1,169	2	8,317,338	186	22	0.042
4	466	1,310	1,015	2	14,865,465	187	13	0.027
2	312	1,135	939	2	27,991,781	187	7	0.021
Spartan6:xc7vx485tffg1761-3								
8	466	1,758	1,178	2	8,317,338	152	18	0.039
4	371	1,360	1,024	2	14,865,465	143	10	0.026
2	325	1,166	948	2	27,991,781	155	6	0.017
Altera FPGAs								
# of PEs	**ALMs**	**ALUTs**	**FFs**	**MBits**	**Avg. Latency** [$Clock\ cycles$]	f [MHz]	**TP** $[\frac{Op}{sec}]$	**TP/Area** $[\frac{Op}{Slices \cdot sec}]$
Stratix V:5SGXEA7K2F40C3								
8	883	1,501	1,222	20,480	8,317,338	224	27	0.030
4	676	1,162	746	20,810	14,865,465	216	14	0.021
2	588	961	647	20,480	27,991,781	224	8	0.014
Cyclone V:5CEBA4F23C7								
8	858	1,517	937	20,786	8,317,338	121	14	0.017
4	680	1,162	692	20,875	14,865,465	122	8	0.012
2	588	961	571	20,912	27,991,781	119	4	0.007
Stratix IV:EP4SE530H35C4								
8	1,007	1,554	974	20,480	8,317,338	182	22	0.022
4	835	1,216	785	20,480	14,865,465	181	12	0.015
2	899	933	952	20,480	27,991,781	185	7	0.009
Actel FPGAs								
# of PEs	**LEs**	**4LUTs**	**FFs**	**RAM**	**Avg. Latency** [$Clock\ cycles$]	f [MHz]	**TP** $[\frac{Op}{sec}]$	**TP/Area** $[\frac{Op}{LEs \cdot sec}]$
IGLOO2-M2GL005S:1FG484								
8	3,205	2,835	1,491	10	2,451,931	94	11	0.004
4	2,753	2,385	1,341	10	2,763,252	81	5	0.002
2	2,522	2,079	1,269	10	3,392,812	97	3	0.001

other implementations in Table VI. The best performance is reported in [14] with a throughput/area ratio that is almost six times ours but the design uses special DSP units and as many as 24 BRAMs compared to only 2 in our design for the external memories. The rest of the reported results are outperformed by our design when compared in terms of throughput/area ratio.

We performed power simulations on our designs using Xilinx Xpower Analyzer tool and the results are reported in Tables VII and VIII. The power simulations were performed over 5 samples per field and the average static and dynamic power for the total number of samples is taken for $w = 16, 32$ and 64 in case of high-speed and for $\#PE = 2, 4$, and 8 in case of lightweight. The simulations were performed over a clock frequency of 100 MHz.The IGLOO2 power results are the post implementation estimates reported by Libero SoC.

The results show that as w increases, in case of the high-speed designs, the dynamic power increases and vice-versa. Same thing applies for the lightweight design as the dynamic power consumption is directly proportional to the $\#PE$. Virtex-6 and Spartan-6 were the exceptions to that. The former reported a very high static power which might have affected the results and the latter reported higher dynamic power when $w = 16$ than that when $w = 32$.

VII. Conclusions

We designed two implementations of a scalable ECC processor for high-speed and lightweight. We added countermeasures to our designs to protect against SPA and DPA. Unlike many published results, our processors are not limited to NIST primes but rather generalized to any GF(p). Our throughput/area results are slightly lower than the high-speed design by [12], however, we use only half of their area, only one Montgomery Multiplier and no DSP units. In case of lightweight, our design is about 6× larger as compared to [14]. However, we do not use any DSP units and only two BRAMs. As future work we intend to perform on-board power measurements and provide ASIC results.

References

[1] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan 1987.

[2] V. S. Miller, "Uses of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85*, ser. Lecture Notes in Computer Science (LNCS), H. C. Williams, Ed., vol. 218. Springer, 1986, pp. 417–426.

[3] *Digital Signature Standard (DSS)*, National Institute of Standards and Technology (NIST), FIPS Publication 186-4, July 2013, http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[4] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management part 1: General," National Institute of Standards and Technology, Tech. Rep., Mar. 2007.

[5] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[6] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology - CRYPTO 96*, ser. Lecture Notes in Computer Science (LNCS), vol. 1109. Berlin: Springer-Verlag, 1996, pp. 104–113.

[7] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Workshop on Cryptographic Hardware and*

TABLE V: Comparison of the High-Speed Design

Work	Device	Curve Size	Curve Type	Slices	LUTs	DSPs	BRAMs	f [MHz]	TP $[\frac{Op}{sec}]$	TP/Area $[\frac{Op}{LUTs \cdot sec}]$
TW[W=32] (Protected)	VX-7	192	$GF(p)$	1,227	2,681	0	4	214	247	0.090
		224	$GF(p)$						187	0.069
		256	$GF(p)$						146	0.054
		384	$GF(p)$						67	0.024
		521	$GF(p)$						37	0.013
TW[W=64] (Protected)	VX-7	192	$GF(p)$	1,269	3,036	0	8	184	239	0.078
		224	$GF(p)$						177	0.058
		256	$GF(p)$						142	0.046
		384	$GF(p)$						63	0.020
		521	$GF(p)$						35	0.011
[12][W=32] (Unprotected)	VX-7	192	$GF(p)$	N/A	6,816	20	N/A	225	3,260	0.478
		256	$GF(p)$						1,510	0.221
		384	$GF(p)$						551	0.080
		521	$GF(p)$						231	0.033
[12][W=64] (Unprotected)	VX-7	384	$GF(p)$	N/A	8,273	64	N/A	161	759	0.091
		521	$GF(p)$						320	0.038
[11] (N/A)	VX-6	192	p-192	11,200	32,900	289	128	100	3,334	0.101
		224	p-224						2,858	0.086
		256	p-256						2,500	0.075
		384	p-384						848	0.025
		521	p-521						625	0.018
[14]	VX-5	256	p-256	81	212	8	22	172	91	0.429
[25]	VX-5	192	$GF(p)$	N/A	6,100	N/A	N/A	97	488	0.080
		256	$GF(p)$	N/A	7,800	N/A	N/A	82	248	0.031

TW→This Work; VX→ Virtex;
$GF(p)$ → any prime for a given size; p-xxx → NIST prime only

TABLE VI: Comparison of Lightweight Results

Work	Device	Curve Size	Curve Type	Slices	LUTs	DSPs	BRAMs	f [MHz]	TP $[\frac{Op}{sec}]$	TP/Area $[\frac{Op}{Slices \cdot sec}]$
TW[#PEs=8]	SN-6	192	$GF(p)$	481	1,513	0	2	165	156	0.325
		224	$GF(p)$						100	0.207
		256	$GF(p)$						67	0.139
		384	$GF(p)$						20	0.041
		521	$GF(p)$						7	0.015
[26]	SN-6	256	$p-256$	221	630	1	3	N/A	N/A	N/A
[14]	SN-6	256	$p-256$	72	193	8	24	156	82	1.139
	VX-5	256	$p-256$	81	212	8	22	172	91	1.123
[27]	VX-2 Pro	224	$p-224$	773	N/A	1^1	3	210	122	0.158
	VX-2 Pro	256	$p-256$	773	N/A	4^1	3	210	100	0.129
[28]	VX-2 Pro	256	$GF(p)$	1694	N/A	2^1	9	108	34	0.020

TW→This Work; VX→ Virtex; SN→ Spartan; ^{1} → Multipliers;
$GF(p)$ → any prime for a given size; p-xxx → NIST prime only

TABLE VII: High-speed Power measurements using Xilinx Xpower Analyzer and Libero SoC

Family	Avail. LUTs	$\mathbf{P}_{static}$ [mW] W=64	W=32	W=16	$\mathbf{P}_{dynamic}$ [mW] W=64	W=32	W=16	$\mathbf{P}_{total}$ [mW] W=64	W=32	W=16
Virtex7	303,600	241	241	241	52	36	30	293	278	271
Virtex6	150,720	3,424	3,423	3,423	86	45	54	3,510	3,468	3,477
Zynq	53,200	100	100	100	51	40	31	164	140	144
Artix7	63,400	82	82	82	47	40	39	129	122	121
Spartan6	9,112	20	20	20	36	24	26	56	44	46
IGLOO2	12,084	13	12	11	1	1	0.3	14	13	11

TABLE VIII: Lightweight Power measurements using Xilinx Xpower Analyzer and Libero SoC

Family	Avail. LUTs	$\mathbf{P}_{static}$ [mW] #PE=8	#PE=4	#PE=2	$\mathbf{P}_{dynamic}$ [mW] #PE=8	#PE=4	#PE=2	$\mathbf{P}_{total}$ [mW] #PE=8	#PE=4	#PE=2
Zynq	53,200	100	100	100	31	18	12	131	118	112
Artix7	63,400	82	82	82	39	22	17	121	104	99
Spartan6	9,112	20	20	20	19	8	6	29	28	26
IGLOO2	6,060	11	12	11	1	0.4	1	12	12	12

Embedded Systems (CHES), ser. LNCS, C. Paar and Ç. K. Koç, Eds., vol. 1717. Springer, 1999, pp. 94–108.

[8] N. Meloni, *New Point Addition Formulae for ECC Applications*. Springer, 2007, pp. 189–201.

[9] R. R. Goundar, M. Joye, and A. Miyaji, "Co-z addition formulæ and binary ladders on elliptic curves," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 6225. Springer, 2010, pp. 65–79.

[10] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO'99*, ser. Lecture Notes in Computer Science (LNCS), vol. 1666. Springer, Aug 1999, pp. 388–397.

[11] H. Alrimeih and D. Rakhmatov, "Pipelined modular multiplier supporting multiple standard prime fields," in *Int. Conf. on Application-Specific Systems, Architectures and Processors*, 2014.

[12] D. Amiet, A. Curiger, and P. Zbinden, "Flexible FPGA-based architectures for curve point multiplication over GF(p)," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 107–114.

[13] P. Sasdrich and T. Güneysu, "Implementing curve25519 for side-channel–protected elliptic curve cryptography," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 1, pp. 3:1–3:15, Nov 2015.

[14] D. B. Roy, P. Das, and D. Mukhopadhyay, "ECC on your fingertips: A single instruction approach for lightweight ECC design in gf(p)," in *Selected Areas in Cryptography – SAC 2015*, O. Dunkelman and L. Keliher, Eds. Springer, 2016, pp. 161–177.

[15] B. Baldwin, R. R. Goundar, M. Hamilton, and W. P. Marnane, "Co-z ecc scalar multiplications for hardware, software and hardware-software co-design on embedded systems." *J. Cryptographic Engineering*, vol. 2, no. 4, pp. 221–240, 2012.

[16] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of an elliptic curve processor over GF(p)," in *Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*.

[17] A. Tenca and Ç. K. Koç, "A scalable architecture for montgomery multiplication," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, C. Paar and Ç. K. Koç, Eds., vol. 1717. Springer, 1999, pp. 94–108.

[18] M. Huang, K. Gaj, and T. El-Ghazawi, "New hardware architectures for Montgomery modular multiplication algorithm," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 923–936, July 2011.

[19] K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster, "ATHENa – automated tool for hardware evaluation: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs," in *20th Int. Conf. on Field Programmable Logic and Applications - FPL 2010*, 2010, pp. 414–421.

[20] B. Baldwin, R. Moloney, A. Byrne, G. McGuire, and W. P. Marnane, "A hardware analysis of twisted edwards curves for an elliptic curve cryptosystem," in *5th Int. Workshop on Applied Reconfigurable Computing (ARC)*, 2009.

[21] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis, "An RNS implementation of an GF(p) elliptic curve point multiplier," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 56, no. 6, pp. 1202–1213, June 2009.

[22] T. Güneysu and C. Paar, "Ultra high performance ecc over nist primes on commercial fpgas," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, 2008, pp. 62–78.

[23] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime fields," *IEEE Trans. VLSI Syst.*, vol. 17, no. 8, pp. 1099–1112, Aug 2009.

[24] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, "Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks," *Computers & Electrical Engineering*, vol. 35, no. 2, pp. 329 – 338, 2009.

[25] B. Baldwin, R. R. Goundar, M. Hamilton, and W. P. Marnane, "Co-z ECC scalar multiplications for hardware, software and hardware–software co-design on embedded systems," *Journal of Cryptographic Engineering*, vol. 2, no. 4, pp. 221–240, 2012.

[26] B. Driessen, T. Güneysu, E. B. Kavun, O. Mischke, C. Paar, and T. Pöppelmann, "IPSecco: A lightweight and reconfigurable IPSec core," in *Int. Conf. on Reconfigurable Computing and FPGAs*, Dec 2012, pp. 1–7.

[27] M. Varchola, T. Güneysu, and O. Mischke, "MicroECC: A lightweight reconfigurable elliptic curve crypto-processor," in *Int. Conf. on Reconfigurable Computing and FPGAs*, 2011, pp. 204–210.

[28] J. Vliegen, N. Mentens, J. Genoe, A. Braeken, S. Kubera, A. Touhafi, and I. Verbauwhede, "A compact FPGA-based architecture for elliptic curve cryptography over prime fields," in *Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, July 2010, pp. 313–316.