Efficient Architectures for Elliptic Curve Cryptography Processors for RFID

Lawrence Leinweber, Christos Papachristou, and Francis G. Wolff

Abstract— **RFID tags will supplant barcodes for product identification in the supply chain. The capability of a tag to be read without a line of sight is its principal benefit, but compromises the privacy of the tag owner. Public key cryptography can restore this privacy. Because of the extreme economic constraints of the application, die area and power consumption for cryptographic functions must be minimized. Elliptic curve processors efficiently provide the cryptographic capability needed for RFID.**

This paper proposes efficient architectures for elliptic curve processors in $GF(2^m)$. One design requires six *m*-bit registers and six Galois field multiply operations per key bit. The other design requires five *m*-bit registers and seven Galois field multiply operations per key bit. These processors require a small number of circuit elements and clock cycles while providing protection from simple side-channel attacks. Synthesis results are presented for power, area, and delay in 250, 130 and 90 nm technologies. Compared with prior designs from the literature, the proposed processors require less area and energy. For the B-163 curve, with bit-serial multiplier, the first proposed design synthesized in an IBM low-power 130 nm technology requires an area of 9613 gate equivalents, 163,355 cycles and 4.14 µJ for an elliptic curve point multiplication. The other proposed design requires 8756 gate equivalents, 190,570 cycles and 4.19 µJ.

Index Terms—Elliptic Curve Cryptography, RFID, security

I. Introduction

Radio Frequency Identification (RFID) has evolved with falling semiconductor prices and power requirements. Low-cost RFID tags now include modest computing capability using power coupled by antenna from the reader [1]. When tags reach a sufficiently low price, perhaps $0.05, RFID systems will become viable replacements for barcode technology used to track products through the supply chain [2].

The capability of a tag to identify itself without a line of sight to the reader is the principal benefit of RFID over barcode systems; however, this capability also compromises the privacy of the tag's owner. Public key cryptography on the RFID tag can protect the owner.

To make the economic cost of tags very low, die area and power consumption must be minimized. The cost of the silicon needs to be low, about $0.01. The tag's antenna must be small, severely limiting the power available to the silicon.

The Rivest-Shamir-Adleman (RSA) system was the first practical implementation of a public key cryptosystem. RSA is still the most widely used system but elliptic curve cryptography (ECC) provides the same security for fewer resources, so is better for small devices such as RFID [3]. Much of its efficiency is due to the definition of elliptic curves over Galois fields, especially those of characteristic two, $GF(2^m)$, which can be implemented efficiently in specialized digital hardware.

II. Contributions

A. Six Register, Six Multiply Elliptic Curve Processor

A processor is proposed that requires 6 *m*-bit registers and 6 Galois field multiply operations per key bit. The processor requires $6m + O(\log m)$ flip-flops and $6m^2 + O(m \cdot \log m)$ cycles per encryption. No processor in the literature that requires as few multiply operations has as few flip-flops (if inversion has at least twice the cost of multiplication).

B. Five Register, Seven Multiply Elliptic Curve Processor

A processor is proposed that requires 5 *m*-bit registers and 7 Galois field multiplies per key bit. It requires $5m + O(\log m)$ flip-flops and $7m^2 + O(m \cdot \log m)$ cycles. No processor in the literature that has as few flip-flops requires as few multiplies.

C. Efficient Integration of Elliptic Curve Technologies

The proposed designs combine well-known algorithms (Lopez-Dahab addition and doubling, Montgomery ladder multiplication, Itoh-Tsujii inversion) in efficient architectures, while preserving resistance to side-channel attacks.

D. Processor Synthesis Case Study in 3 Technology Scales

Synthesis results are presented at 250, 130 and 90 nm nodes, demonstrating the practicality of public key cryptography for RFID tags, while illustrating the difficulty of comparing gate equivalent area across technology scales.

III. Background

Galois field background is given in McEliece [4]. The proposed processors use characteristic two Galois extension fields, $GF(2^m)$. Itoh and Tsujii provided an inversion algorithm that requires $O(\log m)$ multiplies and $O(m)$ squarings [5].

An elliptic curve is the graph of an equation of the form of the generalized Weierstrass equation [6]:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Defined over $GF(2^m)$, if $a_1 \neq 0$, and by change of variable:

$$y^2 + xy = x^3 + a'_2x^2 + a'_6$$

The Group Law defines a way of adding two points, P_1 and P_2, that satisfy the elliptic curve equation, to produce a third point, P_3, also on the curve [7]. The Group Law defines the additive inverse, $-P = -(x, y) = (x, x + y)$. Graphically, P_1, P_2,

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

and $-P_3$ are collinear. A point at infinity is also needed (and is the additive identity). Since rational numbers are maintained to postpone division, a denominator of zero indicates infinity.

The line through P_1 and P_2 has $\lambda = (y_1 + y_2) / (x_1 + x_2)$ as its slope if $x_1 \neq x_2$, or $\lambda = x_1 + y_1 / x_1$, if $x_1 = x_2$. Therefore:

$$x_3 = \lambda^2 + \lambda + a'_2 + x_1 + x_2$$
$$y_3 = \lambda(x_1 + x_3) + y_1 + x_3$$

The projective coordinate system is used to represent a point $P = (x, y, z)$, which is equivalent to the affine $(x/z, y/z)$.

The Montgomery ladder performs point multiplication by maintaining two points, kP and $(k+1)P$ and either doubling the first and adding the two to get $2kP$ and $(2k+1)P$ or adding the two and doubling the second to get $(2k+1)P$ and $(2k+2)P$ [8].

Lopez and Dahab gave efficient adding and doubling [9]:

$$x_{2k} = x_k^4 + a'_6 \cdot z_k^4 = \left(x_k^2 + a'_6{}^{1/2} \cdot z_k^2\right)^2$$
$$z_{2k} = x_k^2 \cdot z_k^2$$
$$x_{2k+1} = x_1 \cdot z_{2k+1} + (x_k \cdot z_{k+1}) \cdot (x_{k+1} \cdot z_k)$$
$$z_{2k+1} = (x_k \cdot z_{k+1} + x_{k+1} \cdot z_k)^2$$

where $(x_k, y_k, z_k) = kP$ and x_1 is the x coordinate of P.

There is no known polynomial time method for recovering k given kP and P, though the elliptic curve discrete logarithm problem (ECDLP) has been studied extensively [10].

The Montgomery ladder is regarded as immune to timing attacks and SPA. To protect against DPA, it is recommended that a random r is generated and used in the conversion from affine (x, y) to projective $(r{\cdot}x, r{\cdot}y, r)$ [11]. This would require an extra register in the proposed architecture. A result in projective coordinates can be weak because it is not supported by the study of ECDLP. In the extreme case, an affine (x, y) could be represented as projective $(k{\cdot}x, k{\cdot}y, k)$, revealing k. This can be corrected by multiplying the projective result by a random r, producing $(r{\cdot}x, r{\cdot}y, r{\cdot}z)$ to represent (x, y, z) [12]. This alleviates the RFID tag from the need to perform division, but requires transmitting both $r{\cdot}x$ and $r{\cdot}z$ to the reader. Our solution is to divide and transmit x/z. This requires $O(m \cdot \log m)$ machine cycles to avoid transmitting m bits.

IV. Related Works

Architecture for ECC is given in Fournaris and Koufopavlou [13]. Most of the background and design decisions of elliptic curve processors were presented in a paper in 2006 [14]. Subsequently, a series of processors was proposed by the Computer Security and Industrial Cryptography (COSIC) research group. These processors would use projective coordinates, most-significant-bit-first Galois field multiplication, versions of the Lopez-Dahab formulas and the Montgomery ladder. All omitted inversion. Most would not include a dedicated squarer [15][16][17].

Another processor performed modified Lopez-Dahab formulas in seven Galois field multiply operations per key bit and required only six registers [18]. A modified version of the projective coordinate system was used in which the two Montgomery ladder points shared one z coordinate. The ALU had no dedicated registers, but shared the processor's elliptic curve registers. Datapaths were reduced to a bare minimum to save area and interconnect.

V. Description of the Proposed ECC Processors

A. Data Flows

Fig. 1 and Fig. 2 illustrate the data flows required to perform Lopez-Dahab point adding and doubling. Point multiplication is formed by repeatedly performing point adding and doubling, according to the Montgomery ladder. There are two versions of the processor, *R6* and *R5*. Fig. 1 gives the adding and doubling data flow diagram for the *R6* processor, which requires 6 m-bit registers. Fig. 2 gives that of the *R5* processor, which uses 5 m-bit registers.

In Fig. 1, the top of the diagram shows the initial register contents and the bottom, the final. At any one time, no more than five variables are live. Usually, additions and multiplications are between one x and one z variable. It can be arranged so that this is always true by introducing a third z variable and by grouping $a'_6{}^{1/2}$ and x_1 with the x variables.

The right side of Fig. 1 gives the data flow for point adding, which must be carried out before point doubling on the left side of the diagram. The left column of Table 1 gives the algorithm for point adding for the *R6* processor, which requires four Galois field multiplications. Squaring is performed with a dedicated circuit of combinatorial logic, which requires only one cycle to complete. The left column of Table 2 gives the algorithm for point doubling, which requires two more Galois field multiplications. The five m-bit registers are x_A, x_B, z_A, z_B and z_C. The sixth register is the multiplier product. There are $O(\log m)$ flip-flops in the control logic.

Fig. 2 shows the data flow for the *R5* processor. This alternative uses the multiplier product register, p, in the ALU as a temporary variable for the adding and doubling algorithms. The generator, x_1, and constant zero can be grouped with the x variables. The register p and $a'_6{}^{-1/2}$ can be grouped with the z's. Since p is both the multiplier product and used as a temporary, it must not be used as such during Galois field multiply operations. That poses no difficulty during point adding, whose flow is shown in the right half of the diagram in Fig. 2, but during point doubling, in the left half, an extra Galois field multiply is required, as represented by these formulas:

$$x_{2k} = \left(a'_6{}^{-1/2} \cdot x_k^2 + z_k^2\right)^2$$
$$z_{2k} = a'_6{}^{-1/2} \cdot \left(a'_6{}^{-1/2} \cdot x_k^2\right) \cdot z_k^2$$

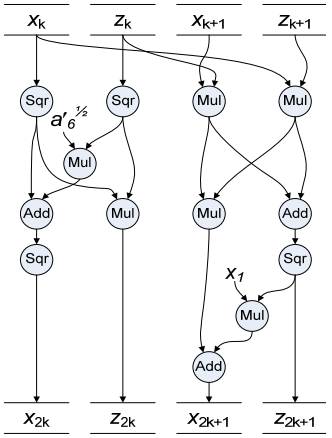

Fig. 1: Adding and Doubling Data Flows for *R6* Processor

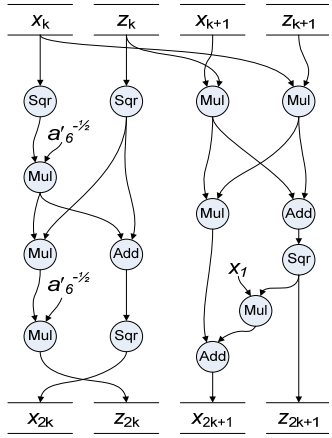

Fig. 2: Adding and Doubling Data Flows for *R5* Processor

Table 1: Point Adding Algorithms

$(x_B, z_B) \leftarrow (x_A, z_A) + (x_B, z_B)$

R6: Temporary register z_C	*R5*: Multiplier Product p
Generator x coordinate x_1	Generator x coordinate x_1
$x_B \leftarrow x_B \times z_A$	$x_B \leftarrow x_B \times z_A$
$z_B \leftarrow x_A \times z_B$	$z_B \leftarrow x_A \times z_B$
$z_C \leftarrow x_B + z_B$	$p \leftarrow x_B \times z_B$
$x_B \leftarrow x_B \times z_B$	$z_B \leftarrow x_B + z_B$
$z_B \leftarrow z_C^2$	$x_B \leftarrow 0 + p$
$z_C \leftarrow x_1 \times z_B$	$z_B \leftarrow z_B^2$
$x_B \leftarrow x_B + z_C$	$p \leftarrow x_1 \times z_B$
	$x_B \leftarrow x_B + p$

Table 2: Point Doubling Algorithms

$(x_A, z_A) \leftarrow 2(x_A, z_A)$

R6: Temporary register z_C	*R5*: Multiplier Product p
EC Constant $a'_6{}^{1/2}$	EC Constant $a'_6{}^{-1/2}$
$x_A \leftarrow x_A^2$	$x_A \leftarrow x_A^2$
$z_A \leftarrow z_A^2$	$z_A \leftarrow z_A^2$
$z_C \leftarrow a'_6{}^{1/2} \times z_A$	$x_A \leftarrow x_A \times a'_6{}^{-1/2}$
$z_A \leftarrow x_A \times z_A$	$p \leftarrow x_A \times z_A$
$x_A \leftarrow x_A + z_C$	$z_A \leftarrow x_A + z_A$
$x_A \leftarrow x_A^2$	$x_A \leftarrow 0 + p$
	$p \leftarrow x_A \times a'_6{}^{-1/2}$
	$x_A \leftarrow z_A^2$
	$z_A \leftarrow 0 + p$

Note that these produce different values for x_{2k} and z_{2k} than used in the *R6* processor or the Lopez-Dahab equations, but the ratio, x_{2k}/z_{2k}, is the same.

The algorithms for point adding and doubling for the *R5* processor are shown on the right sides of Table 1 and Table 2. The five m-bit registers are x_A, x_B, z_A, z_B and p, the ALU multiplier product. There are no other flip-flops in the *R5* processor except $O(\log m)$ flip-flops in the control logic.

B. *Arithmetic Logic Unit* (*ALU*)

The Galois field ALU can produce a^2, b^2, $a + b$, or $a \cdot b$. The ALU exposes the multiplier p register used by the *R5* version of the processor. Addition in $GF(2^m)$ is simply bit-wise XOR. Squaring requires about $1.5m$ or $0.5m$ XOR gates for reduction depending on whether the minimum reduction polynomial for m is a pentanomial or a trinomial. To save area, we use minimum reduction polynomials.

The multiplier is digit serial, MSB-first. Fig. 3 illustrates a six bit multiplier ($m = 6$) with three bit digits ($w = 3$). a and b are the multiplier inputs; y is the output. There are w multiplexers that select w bits from b. Each multiplexer selects one of $\lceil m/w \rceil$ inputs. If the high order digit of b is incomplete, it must be padded with zeroes. There is a register to count $\lceil m/w \rceil$ steps for selector s. That register has $\lceil \log_2 m/w \rceil$ bits. The multiplier has only one m-bit register, p.

C. *Key Control Logic*

The Montgomery ladder resists side-channel attacks when implemented so the key has no influence on the sequence of operations. In these designs, multiplexers select x_A vs. x_B and z_A vs. z_B for input and output from the ALU during point adding and doubling, minimizing the key's influence on the processor's power profile. The multiplexer selector signals are connected via XOR gates in the key control logic, in Fig. 4.

A counter of $\lceil \log_2 m \rceil$ bits is initialized at reset with the constant $m - 1$. During each loop to perform point adding and doubling, the counter output forms the selector signals of an m-to-1 multiplexer that selects a key bit. At the end of each iteration of the loop, the counter is decremented.

D. *Inversion Control Logic*

The Itoh-Tsujii inversion algorithm raises a number, a, to the power $2^m - 2$. This can be broken down into a sequence of squarings and multiplies. Let $h(n) = a^{2^n-1}$. So $h(1) = a$. And,

$$[h(m-1)]^2 = \left(a^{2^{m-1}-1}\right)^2 = a^{2^m-2} \equiv a^{-1} (\mathrm{mod}\ 2^m).$$

Therefore $[h(m-1)]^2$ is the inverse of $h(1)$, mod 2^m. If $h(n)$ is squared n times then multiplied by $h(n)$, this produces $h(2n)$:

$$[h(n)]^{2^n} \cdot h(n) = \left(a^{2^n-1}\right)^{2^n} \cdot a^{2^n-1} = a^{2^{2n}-1} = h(2n).$$

If $h(2n)$ is squared and multiplied by a, this gives $h(2n + 1)$:

$$[h(2n)]^2 \cdot a = \left(a^{2^{2n}-1}\right)^2 \cdot a^1 = a^{2^{2n+1}-1} = h(2n+1).$$

Initially, $h(1) = a$, so this provides a procedure to get $h(n)$ for any $n > 0$, by repeatedly shifting the binary form of n, to get $2n$ or by shifting n and adding one, to get $2n + 1$. The desired final value for n is $m - 1$.

Shifting and selectively adding one to form $m - 1$ while performing repeated squarings and multiplies produces a to the power $2^{m-1} - 1$, and squared again gives $2^m - 2$. The control logic for this is shown in Fig. 5. A shift register of $2\lceil \log_2 m \rceil$ bits is initialized with the constant $m - 1$ in the lower half. The loop begins with a left shift. If a one bit is shifted into the upper half, the accumulated result is squared and multiplied by a. Then the upper half is loaded into a counter of $\lceil \log_2 m \rceil$ bits

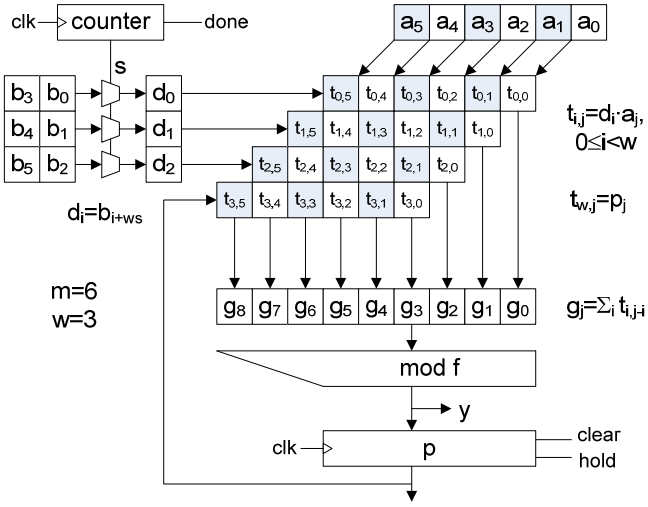

Fig. 3: MSB Digit-Serial Multiplier

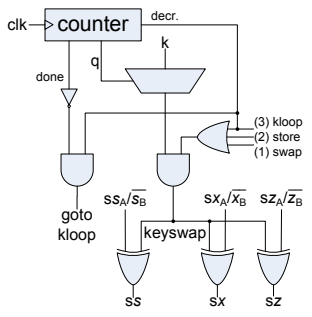

Fig. 4: Key Control Logic

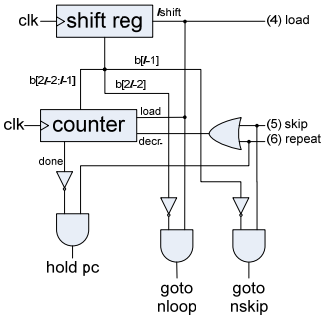

Fig. 5: Inverter Control Logic

to count squarings of a copy of the accumulated result, and multiplied. This is repeated for each bit of the constant $m - 1$.

E. High-Level Organization

The ALU inputs are supplied from the x and z buses. In the *R6* design, x is selected by multiplexer from x_A, x_B, $a'_6{}^{½}$ and x_1 while z is selected from among z_A, z_B and z_C. The ALU output is the s bus. In *R6*, registers x_A, x_B, z_A, z_B and z_C are equipped to hold, or load from s. The last register is the multiplier product, p. In the *R5* design, x is x_A, x_B, 0 or x_1 and z is z_A, z_B, $a'_6{}^{-½}$ or p. Registers x_A, x_B, z_A and z_B can hold, or load from s.

The high-level control logic consists of a five bit program counter that selects 12-bit microinstructions to select the ALU inputs, operation and output register and to select key and inversion control flow. Processor inputs are the key, k, and generator, $g = x_1$. The output is $e = x_A$. The processors perform the elliptic curve point multiplication, $E = k \cdot G$, where E and G are points with affine x coordinates e and g.

Initially, $0G$ and $1G$ are loaded into x_A/z_A and x_B/z_B, requiring $x_A \neq 0$, $z_A = 0$, and $x_B/z_B = g = x_1$ using only g and the ALU's adding and squaring operations. Firstly, g^2 is stored by selecting g from the x bus and the a^2 operation from the ALU to set $x_B = g^2$ and $z_B = g^2$. Then $z_A = x_B + z_B = g^2 + g^2 = 0$, using the ALU's $a + b$. And $z_B = g + z_A = g + 0 = g$. Such microcoding avoids special hardware for initializing registers.

F. Implementation

The processors were modeled and tested at the gate level in a C++ program which then generated Verilog code that was synthesized and simulated with Synopsys Design Compiler. The simulated processors were tested at the class level and as complete systems, in many cases using 11 bit examples from [7]. Full scale tests were carried out with the degree 163 elliptic curve NIST B-163 [19], using twenty-six vectors from COSIC. Verilog versions were verified for $1 \leq w \leq 16$.

The number of machine cycles required for point multiplication for the *R6* and *R5* designs and the final division are given in the following formulas for t_6, t_5 and t_d. HW is the Hamming weight function. The number of flip-flops required for *R6* and *R5* are given in the formulas for n_6 and n_5:

$$t_6 = 6lm + 13m + 11 + t_d, t_5 = 7lm + 17m + t_d + 5, l = \lceil m/w \rceil$$
$$t_d = (l+2)\lceil \log_2 m \rceil + (l+1)HW(m-1) + m - l - 1$$
$$n_6 = 6m + n_c, n_5 = 5m + n_c, n_c = 4\lceil \log_2 m \rceil + \lceil \log_2 l \rceil + 6$$

VI. Performance of the Proposed ECC Processors

A. Synthesis Results

Both versions of the processor, *R6* and *R5*, were synthesized for $113 \leq m \leq 251$, m prime, and $w \in \{1,2,4,8,16\}$ using Synopsys Design Compiler and 3 standard cell libraries: a LEDA library, excluding cells with large leakage current, for a TSMC 0.25 µm process and two ARM libraries, provided by MOSIS, for low-power 130 nm and high-v_t 90 nm IBM processes. Area, delay, dynamic power and leakage power simulation results were obtained using Synopsys tools. For dynamic power, activity was measured by simulating the synthesized circuits with hard-wired random a'_6 parameters and random g and k vectors. Each activity test was run for 16 complete encryption operations with different g and k vectors.

Fig. 6 through Fig. 13 graph area, time and energy vs. the degree, m, from 125 to 250. One NIST recommended curve has $m = 163$. Each graph includes plots for the two versions of the processor, *R6* and *R5*. In area, *R5* is smaller than *R6*. In time and energy, *R6* is generally smaller.

Fig. 6 gives area in gate equivalents, relative to the area of a two-input, single drive strength NAND gate for the technology. The figure compares bit-serial ($w = 1$) processors in 250, 130 and 90 nm. Processor size in gate equivalents increases as technology scale decreases. Fig. 6 also gives the gate equivalent area for D flip-flops for the bit-serial processors. This also increases as scale decreases simply because of the ratio of areas of D flip-flops and NAND gates in the cell libraries. Although gate equivalent area is often used to compare architectures, it is dependent on technology scale.

Fig. 7 through Fig. 13 graph *R6* and *R5* processor versions in five digit sizes: $w = 1$, 2, 4, 8 and 16.

Fig. 7 graphs time in machine cycles to complete the point multiply operation. This graph includes a very small curve labeled "÷" for the final division to convert the projective result to an affine value. This is a very small portion of the total times shown in the graph. Propagation delay varied from 7.47 to 12.34 ns for 250 nm, 8.81 to 14.85 ns for 130 nm and 9.68 to 16.20 ns for 90 nm, trending slower for the smaller, lower power processes. With a delay of 16.20 ns, the top of the graph, 400,000 cycles, represents 6.48 ms to complete an encryption operation, performance enough for an RFID tag.

Fig. 8 through Fig. 10 graph area results in $(mm)^2$ for the

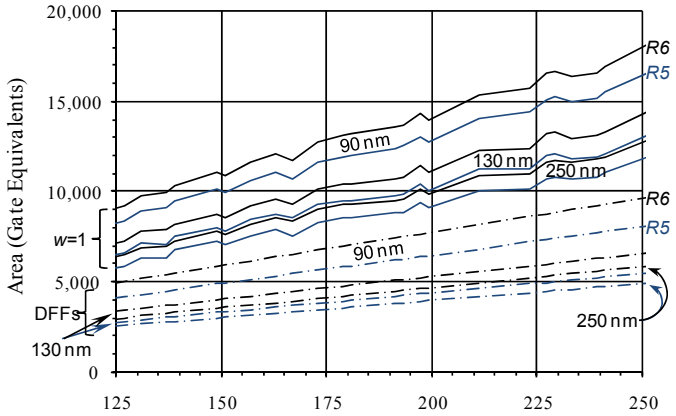

Fig. 6: w=1 Processor, DFFs, Area (Gate Eqv.) vs. Degree, m

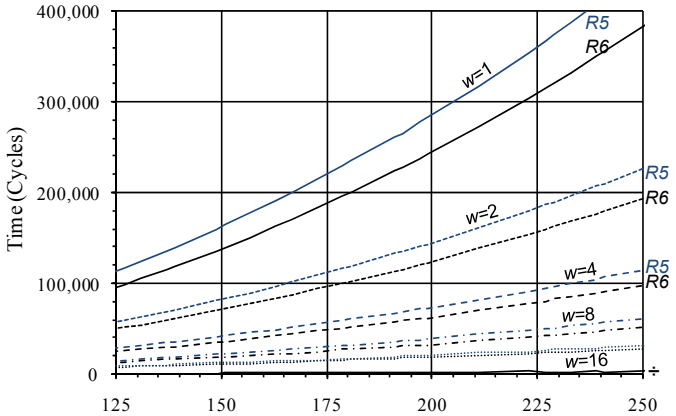

Fig. 7: Processor, Division, Time (Cycles) vs. Degree, m

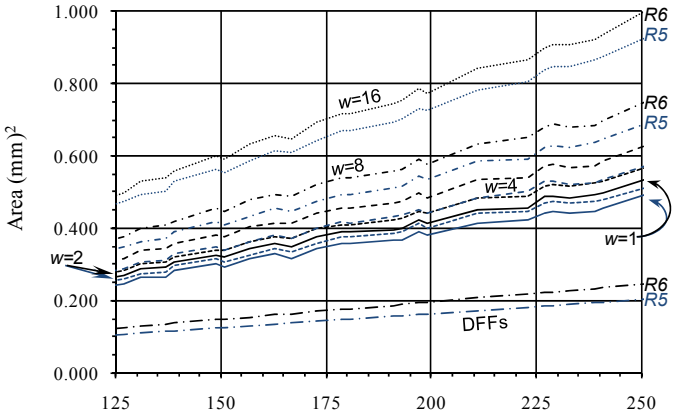

Fig. 8: 250 nm Processor, DFFs, Area (mm)^2 vs. Degree, *m*

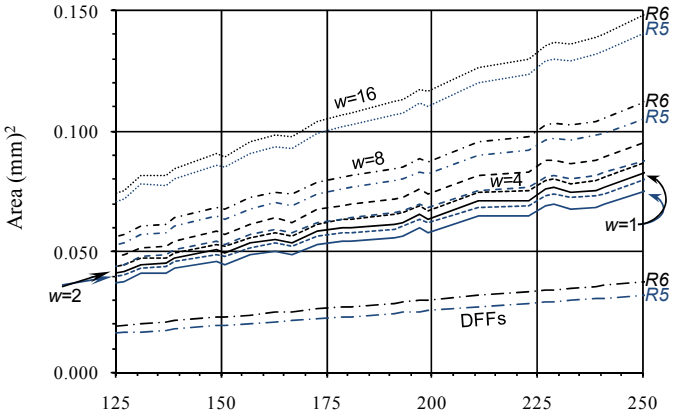

Fig. 9: 130 nm Processor, DFFs, Area (mm)^2 vs. Degree, *m*

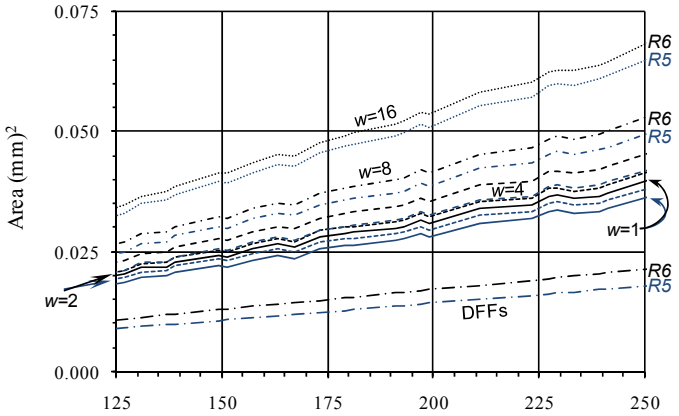

Fig. 10: 90 nm Processor, DFFs, Area (mm)^2 vs. Degree, *m*

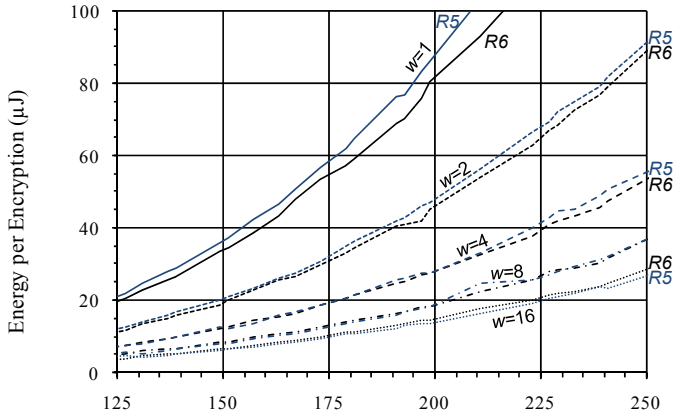

Fig. 11: 250 nm Energy per Encryption (µJ) vs. Degree, *m*

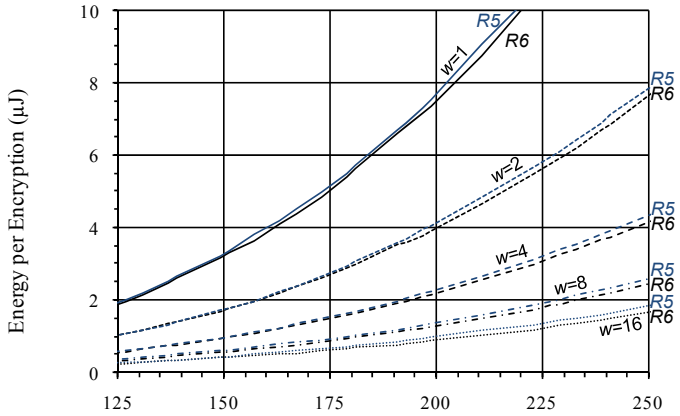

Fig. 12: 130 nm Energy per Encryption (µJ) vs. Degree, *m*

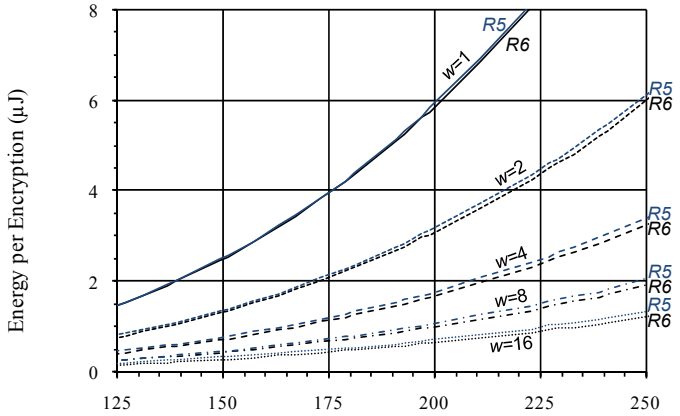

Fig. 13: 90 nm Energy per Encryption (µJ) vs. Degree, *m*

processors and the D flip-flops in the processors. Fig. 11 though Fig. 13 graph energy in µJ for one complete cryptographic operation, the point multiplication. The two sets of three graphs give the results for 250, 130 and 90 nm processes. Energy includes dynamic power and leakage power running at the maximum clock frequency limited by propagation delay. It is perhaps worth noting that the energy required to lift a drop of water (0.025 mL) one centimeter is 2.45 µJ.

B. Comparison with Other Works

Fig. 14 compares several reference works with our designs in terms of registers and multiply operations (if inversion costs two multiplies). No processor that requires as few multiply operations as the *R6* has as few flip-flops. No processor that has as few flip-flops as the *R5* needs as few Galois field multiply operations to complete a point multiplication.

Table 3 compares performance vs. our proposed designs synthesized in 130 nm. References [20] and [21] were in 350 nm; [17] used 250 nm; [16] and [18] used 130 nm.

The algorithm in [20] depends on the number of one bits in the key and is vulnerable to side-channel attacks. [20] and [21] operate in affine coordinates, requiring division for point addition and doubling. Each includes an extended Euclidean algorithm divider, which requires a large number of gates.

References [16] and [17] are compared for combinatorial area. They do not report memory area. [16], [17] and [18] produce projective coordinate results that require transmitting an

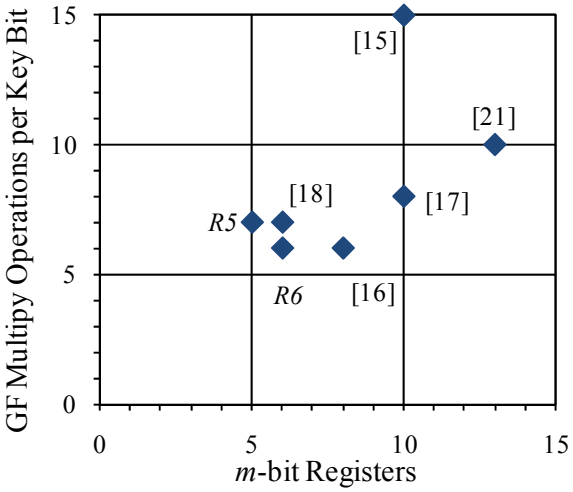

Fig. 14: Register, Multiply Comparison with Other Works

extra *m* bits to the tag reader. [18] has limited datapaths which may allow side-channel attacks depending on implementation.

Comparing energy, [18] reported that using a low leakage power, 130 nm UMC library for a 163-bit processor, in word sizes 1, 2 and 4, required 8.94, 5.29 and 2.94 µJ, respectively. In the low-power 130 nm IBM process, our *R6* processor uses 4.14, 2.22 and 1.22 µJ, and *R5* requires 4.19, 2.26 and 1.26 µJ.

VII. Conclusion

Efficient elliptic curve processors have been presented, which are especially useful for RFID applications. The designs use resources well while resisting side-channel attacks. Synthesis results were discussed and compared.

One of the proposed processors, the *R6*, requires only six Galois field multiplies per key bit. For large key size, *m*, each additional field multiply represents a 17% increase in delay. No processor in the literature that requires as few multiply operations requires as few flip-flops as the *R6* design. Whereas the *R6* requires six flip-flops per key bit, the *R5* design requires only five. For large key size, as flip-flop area reaches 50% of processor area in the *R5*, each additional *m*-bit register represents a 10% increase in circuit area. No processor in the literature that requires as few flip-flops as the *R5* needs as few Galois field multiply operations to complete an elliptic curve point multiplication.

Table 3: Area and Time Comparison with Other Works

			Area (Gate Equiv.)			Time (Cycles)		
Ref.	*m*	*w*	Ref.	*R6*	*R5*	Ref.	*R6*	*R5*
[20]	192	1	16,847	11,191	10,199	296,383	226,593	264,219
[21]	251	1	56,000	14,414	13,149	550,000	384,815	448,814
[16]*	131	1	6,718	4,295	4,193	210,600	106,007	123,686
[16]*	131	4	8,104	5,487	5,378	57,720	28,097	32,938
[16]*	163	1	8,214	5,261	5,097	353,710	163,355	190,570
[16]*	163	4	9,926	6,897	6,621	95,159	42,819	50,148
[17]*	131	1	8,582	4,295	4,193	159,250	106,007	123,686
[17]*	131	2	8,603	4,722	4,543	84,000	54,332	63,496
[17]*	163	1	10,122	5,261	5,097	241,500	163,355	190,570
[17]*	163	2	10,933	5,729	5,601	124,250	83,327	97,339
[18]	163	1	10,106	9,613	8,756	275,816	163,355	190,570
[18]	163	4	12,863	11,240	10,271	78,544	42,819	50,148

*Area comparison does not include memory devices.

VIII. References

[1] K. Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*, 2nd Ed., John Wiley & Sons Ltd., 2003.

[2] S. Garfinkel and B. Rosenberg (Eds.), *RFID Applications, Security, and Privacy*, Addison-Wesley, 2005.

[3] U.S. National Security Agency, "The Case for Elliptic Curve Cryptography," http://www.nsa.gov/business/programs/elliptic_curve.shtml

[4] R.J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

[5] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases," *Inf. Comput.*, 1988.

[6] L.C. Washington, *Elliptic Curves*: *Number Theory and Cryptography*, Chapman & Hall/CRC, 2003.

[7] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman & Hall/CRC, 2005.

[8] P.L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Math. of Computation*, Vol.48, 1987.

[9] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation," *Workshop on Cryptographic Hardware and Embedded Systems* (*CHES*), *LNCS* Vol. 1717, Springer-Verlag, 1999.

[10] I. Blake, G. Seroussi and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[11] K. Okeya and K. Sakurai, "Power Analysis Breaks Elliptic Curve Cryptosystems Even Secure Against the Timing Attack," *Progress in Cryptography*, *LNCS*, Vol. 1977, 2000.

[12] D. Naccache, N.P. Smart, and J. Stern, "Projective Coordinates Leak," *Advances in Cryptography*, *LNCS*, Vol. 3027, 2004.

[13] A.P. Fournaris and O. Koufopavlou, "Hardware Design Issues in Elliptic Curve Cryptography," *Wireless Security and Cryptography, Specifications and Implementations*, N. Sklavos and X. Zhang (Eds.), CRC Press, 2007.

[14] L. Batina, G.M. de Dormale, E. Oswald and J. Wolkerstorfer, "State of the Art in Hardware Implementations of Cryptographic Algorithms," *Information Society Technologies Publication IST-2002-507932*, 2006.

[15] P. Tuyls and L. Batina, "RFID-Tags for Anti-Counterfeiting," *Cryptographers' Track of RSA Conference* (CT-RSA), *LNCS*, Vol. 3860, Springer-Verlag, 2006.

[16] L. Batina, N. Mentens, K. Sakiyama, B. Preneed, and I. Verbauwhede, "Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks," *European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, *LNCS* Vol. 4357, 2006.

[17] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede, "Public Key Cryptography for RFID-Tags," *IEEE Int. Workshop on Pervasive Computing and Commun. Security*, 2007.

[18] Y.K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID," *IEEE Trans. on Comput.*, 2008.

[19] U.S. National Institute for Standards and Technology, "Recommended Elliptic Curves for Federal Government Use", 1999.

[20] J.H. Kim and D.H. Lee, "A Compact Finite Field Processor over $GF(2^m)$ for Elliptic Curve Cryptography, *IEEE Int. Symp. on Circuits and Systems,* 2002.

[21] C. Huang, J. Lai, J. Ren and Qianling Zhang, "Scalable Elliptic Curve Encryption Processor for Portable Application," *Int. Conf. on ASIC*, 2003.