Hyperelliptic Curve Crypto-Coprocessor over Affine and Projective Coordinates

Howon Kim, Thomas Wollinger, Doo-Ho Choi, Dong-Guk Han, and Mun-Kyu Lee

This paper presents the design and implementation of a hyperelliptic curve cryptography (HECC) coprocessor over affine and projective coordinates, along with measurements of its performance, hardware complexity, and power consumption. We applied several design techniques, including parallelism, pipelining, and loop unrolling, in designing field arithmetic units, group operation units, and scalar multiplication units to improve the performance and power consumption.

Our affine and projective coordinate-based HECC processors execute in 0.436 ms and 0.531 ms, respectively, based on the underlying field GF(2^{89}). These results are about five times faster than those for previous hardware implementations and at least 13 times better in terms of area-time products. Further results suggest that neither case is superior to the other when considering the hardware complexity and performance. The characteristics of our proposed HECC coprocessor show that it is applicable to high-speed network applications as well as resource-constrained environments, such as PDAs, smart cards, and so on.

Keywords: Crypto-processor, hyperelliptic curve, elliptic curve cryptosystem.

Manuscript received Jan 15, 2007; revised Oct 9, 2007.

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD)" (The Regional Research Universities Program/Research Center for Logistics Information Technology)

Howon Kim (phone: 82-51-510-1010, email: howonkim@pusan.ac.kr) was with Information Security Research Division, ETRI, Daejeon, Rep. of Korea, and is now with the Department of Computer Engineering, Pusan National University, Pusan, Rep. of Korea.

Thomas Wollinger (email: twollinger@escrypt.com) is with the eScrypt GmbH, Bochum, Germany.

Doo-Ho Choi (email: dhchoi@etri.re.kr) and Dong-Guk Han (christa@etri.re.kr) are with SW & Content Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Mun-Kyu Lee (email: mklee@inha.ac.kr) is with the School of Computer Science and Engineering, Inha University, Seoul, Rep. of Korea.

I. Introduction

With the continuous evolution of information technology, security technology will become increasingly important for protecting critical information and enhancing the privacy of Internet users. An increasing number of applications are being connected wirelessly; therefore, they require protection due to the vulnerability of the wireless link. The core techniques used to maintain security in information technology are cryptographic primitives such as private-key and public-key crypto-algorithms. Private-key algorithms such as AES and triple-DES are usually used to encrypt data, whereas public-key algorithms such as RSA and elliptic curve cryptography (ECC) are used for secure key distribution and digital signatures [1].

Hyperelliptic curve cryptography (HECC), a public-key crypto-algorithm, has been studied with increasing interest during recent years, since it was introduced by Koblitz in 1988 [2]. Unlike the well-known RSA crypto-algorithm, HECC is based on the hardness of solving the discrete logarithm problem (DLP). The main advantages over RSA are that HECC requires shorter key and operand lengths than RSA. Hence, HECC is a promising cryptographic primitive for fast implementation and for resource-constrained environments. In this paper, we present a fast HECC coprocessor in affine and projective coordinates and show that it can be a candidate for resource-constrained and high-speed network applications.

The remainder of this paper is organized as follows. Section II provides a brief overview of the background of hyperelliptic curve cryptosystems and summarizes the previous works involving field programmable gate array (FPGA) implementations of HECC coprocessors. Section III presents the proposed HECC coprocessor architecture, and section IV,

outlines our methodology and the different design options. Results and analysis are provided in section V. Finally, we make some concluding remarks.

II. Background of HECC

In this section, we present an introduction to the theory of HECC over finite fields of arbitrary characteristics, restricting our attention to material that has cryptographic relevance.

1. Basics of HECC

Hyperelliptic curves are a special class of algebraic curves and a generalization of elliptic curves [2]. A hyperelliptic curve of genus $g = 1$ is an elliptic curve. A hyperelliptic curve C of $g \geq 1$ over F is the set of solutions $(x, y) \in F \times F$ to the equation $C: y^2 + h(x)y = f(x)$, where $h(x), f(x) \in F[x]$ with $\deg_x\{h(x)\} \leq g$ and $f(x)$ is monic with $\deg_x\{f(x)\}=2g+1$. There are no solutions $(x, y) \in F \times F$ that simultaneously satisfy the equation $y^2+h(x)y=f(x)$ and the partial derivative equations $2y+h(x)=0$ and $h'(x)y-f'(x)=0$. A divisor of C is a formal sum of the points P over C, expressed as $D = \sum_{P_i \in C} m_i P_i - mP_\infty, m_i \in Z$, where the degree of D, $\sum m_i$ is less than or equal to g, and P_∞ is a point at infinity.

The Jacobian $Jac(C)$ of C is an Abelian group related to C. Each element of $Jac(C)$ can be represented by a pair of polynomials $div(u(x), v(x))$ using Mumford's representation. Here, u is a monic polynomial of degree at most g, and we have $\deg(v)<\deg(u)$. Thus, we can easily handle $Jac(C)$ on software and hardware applications [3]. If the base field of the curve is a finite field with cardinality q, then the Jacobian of the curve is a finite Abelian group of order approximately q^g.

The following Hasse-Weil bound provides an interval for this order:

$$\left\lceil (\sqrt{q}-1)^{2g} \right\rceil \leq \left| Jac(C)(F_q) \right| \leq \left\lfloor (\sqrt{q}+1)^{2g} \right\rfloor .$$

The genus of a hyperelliptic curve should not be greater than three, because of possible attacks [4]. Furthermore, it is widely accepted that the group order for HECC should be at least $\approx 2^{160}$ for practical applications. Therefore, in the case of a genus-2 hyperelliptic curve, an 80-bit field size can be considered secure.

Because ECC is a special case of HECC (genus $g = 1$), the required field size for this case is 160 bits. Therefore, to implement ECC using hardware logic would require 160-bit registers, digital logic gates, and so on. However, the underlying hardware logic units for genus-2 hyperelliptic curves, particularly the field operation logic units, are smaller than those for ECC. This is one reason why there is currently a great deal of research on HECC.

2. HECC over Affine and Projective Coordinates

Cantor proposed an algorithm to add and double the divisors in Mumford's representation [5]. This algorithm requires two major steps. The first step involves finding a semi-reduced divisor $D'=div(u(x)', v(x)')$ such that $D' \sim D_1+D_2=div(u_1(x), v_1(x))+div(u_2(x), v_2(x))$ in the group $Jac(C)$. The second step involves reducing the semi-reduced divisor to an equivalent divisor, $D=div(u(x), v(x))$. This algorithm is known to be time consuming because it requires polynomial arithmetic. However, Harley noticed that the number of operations can be reduced by distinguishing among the possible cases based on the properties of the input divisors. He described an efficient algorithm using various techniques, such as Karatsuba multiplication, CRT, and Newton iteration methods. Harley's algorithm reduces the overall complexity of the group operations [6]. We refer to this algorithm as *explicit formulae*.

Our HECC coprocessor targets genus-2 curves using

Table 1. Inversion-free explicit formulae for adding a divisor on an HECC coprocessor of genus 2 over GF(2^n).

Input	$[U_{11}, U_{10}, V_{11}, V_{10}, Z_1]; [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$; $h = x$; and $f = x^2 + f_1 x + f_0$
Output	$[U_1', U_0', V_1', V_0', Z'] =$ $[U_{11}, U_{10}, V_{11}, V_{10}, Z_1] + [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$
Algorithm	precomputation: (cost: 5M) $Z = Z_1 Z_2; \tilde{U}_{21} = Z_1 U_{21}; \tilde{U}_{20} = Z_1 U_{20}; \tilde{V}_{21} = Z_1 V_{21}; \tilde{V}_{20} = Z_1 V_{20}$; compute resultant r of U_1, U_2 : (6M+1S) $z_1 = U_{11} Z_2 - \tilde{U}_{21}; z_2 = \tilde{U}_{20} - U_{10} Z_2; z_3 = U_{11} z_1 + z_2 Z_1$; $r = z_2 z_3 + z_1^2 U_{10}$; compute almost inverse of U_2 mod U_1 $inv_1 = z_1; inv_0 = z_3$; compute S : (8M) $w_0 = V_{10} Z_2 - \tilde{V}_{20}; w_1 = V_{11} Z_2 - \tilde{V}_{21}; w_2 = inv_0 w_0$; $w_3 = inv_1 w_1; s_1 = (inv_0 + Z_1 inv_1)(w_0 + w_1) - w_2 - w_3 (Z_1 + U_{11})$; $s_0 = w_2 - U_{10} w_3$; precomputation: (8M + 1S) $R = Zr; s_0 = s_0 Z; s_3 = s_1 Z; \tilde{R} = Rs_3; S_3 = s_3^2; S = s_0 s_1$; $\tilde{S} = s_3 s_1; S'' = s_0 s_3; R'' = \tilde{R}\tilde{S}$; compute l : (3M) $l_2 = \tilde{S}\tilde{U}_{21}; l_0 = S\tilde{U}_{20}; l_1 = (\tilde{S} + S)(U_{21} + U_{20}) - l_2 - l_0; l_2 = l_2 + S''$; compute U' : (6M + 2S) $U_0' = s_0^2 + s_1^2 z_1 (z_1 + \tilde{U}_{21}) + z_2 \tilde{S} + R(s_3 + rz_1); U_1' = Sz_1 + R^2$; precomputations: (4M) $l_2 = l_2 - U_1'; w_0 = U_0' l_2 - S_3 l_0; w_1 = U_1' l_2 + S_3 (U_0' - l_1)$; adjust: (3M) $Z' = \tilde{R} S_3; U_1' = \tilde{R} U_1'; U_0' = \tilde{R} U_0'$; compute V' : (2M) $V_0' = w_0 + R'' \tilde{V}_{20}; V_1' = w_1 + R''(\tilde{V}_{21} + Z)$;
Total cost	45M+4S

Table 2. Inversion-free explicit formulae for doubling a divisor on an HECC coprocessor of genus 2 over GF(2^n).

Input	$[U_1, U_0, V_1, V_0, Z]; h = x;$ and $f = x^5 + f_1 x + f_0$
Output	$[U_1', U_0', V_1', V_0', Z'] = 2[U_1, U_0, V_1, V_0, Z]$
Algorithm	compute the resultant and precomputations: (cost: 1M+3S) $Z_2 = Z^2; w_0 = V_1^2; w_1 = U_1^2; w_3 = U_1 Z;$ compute k : (3M) $k_0 = U_1 w_1 + Z(ZV_1 + w_0);$ compute $s = kinv \bmod u$: (7M) $w_4 = k_0 w_3; w_5 = w_1 Z; s_3 = (w_3 + Z)(k_0 + w_1) + w_4 + (1 + U_1) w_5;$ $s_1 = s_3 Z; s_0 = w_4 - Z U_0 w_5;$ precomputations : (6M+3S) $R = Z_2^2 U_0; \tilde{R} = R s_1; S_1 = s_1^2; S_0 = s_0^2; s_4 = s_3 s_1; s_5 = s_0 s_3;$ $S = s_5 Z; R'' = \tilde{R} s_4$ compute l : (3M) $l_2 = U_1 s_4; l_0 = U_0 s_5; l_1 = (s_4 + s_5)(U_1 + U_0) + l_2 + l_0;$ compute U' : (2M + 1S) $U_0'' = S_0 + R s_3 Z; U_1'' R^2;$ precomputations: (4M) $l_3 = l_2 + S + U_1''; w6 = U_0'' l_3 + S_1 l_0; w_7 = U_1'' l_3 + S_1 (U_0'' + l_1);$ adjust: (3M) $Z' = S_1 \tilde{R}; U_1' = \tilde{R} U_1''; U_0' = \tilde{R} U_0'';$ compute V' : (2M) $V_0' = w_6 + R'' V_0; V_1' = w_7 + R'' V_1 + Z';$
Total cost	31M+7S

underlying fields of characteristic two. For the remainder of this paper, we will only consider the group operations of genus-2 hyperelliptic curves (For further information, see [7] and [8].). We have used the explicit formulae for affine coordinates that were introduced in [9]. The explicit formulae for projective coordinates are shown in Tables 1 and 2. The affine coordinates require 1 inversion and 9 multiplications for group doubling and 1 inversion and 21 multiplications for group addition. For the case of projective coordinates, 45 and 31 multiplications are required for group addition and doubling, respectively. In projective coordinates, the inversion operation is not necessary for the group operations. However, if the output of a scalar multiplication is required in affine representation, we need one inversion and a few extra multiplications for the conversion to that coordinate system.

In projective coordinates based on the Mumford representation [3], we can define the quintuple $[U_1, U_0, V_1, V_0, Z]$ such that $\left[x^2 + (U_1/Z)x + (U_0/Z), (V_1/Z)x + (V_0/Z)\right] = \left[x^2 + u_1 x + u_0, v_1 x + v_0\right]$. Table 1 shows the explicit formulae for the case of projective coordinates with the explicit formulae optimized for the case of h_2=h_0=0. In this case, we have optimized the explicit formulae presented in [8] with the assumptions of h=x and f=x^5+f_1x+f_0, where $f_0, f_1 \in GF(2^n)$, and f_0+f_1 is not equal to 0. This assumption has also been applied to the case of affine coordinates in [9].

The cost of each stage of the explicit formulae is shown in Tables 1 and 2. Since the squaring operation in polynomial arithmetic can be implemented as rewired hardware, we can avoid the cost of squaring. After the completion of several stages of the explicit formulae, which consist of 45 multiplications and 4 squaring operations, the divisor addition $[U_1', U_0', V_1', V_0', Z']$ can be obtained.

Table 2 shows the divisor doubling equation. The result of doubling $2[U_1, U_0, V_1, V_0, Z]$ can be obtained after 31 multiplications and 7 squaring operations. As in the case of the addition operation, this equation is derived from [8] with the condition that h=x and f=x_5+f_1x+f_0. There are 14 fewer multiplications than with the divisor sum operation.

3. Previous Works on HECC Coprocessors

This subsection surveys the major preceding works on HECC coprocessor design. The first HECC coprocessor implemented on an FPGA was introduced in [10] It targeted a genus-2 HECC over GF(2^{113}). The scalar multiplication latency of this design is about 20 ms and the group operation is based on the Cantor algorithm. The first result of an HECC coprocessor using the explicit formulae was presented in [11]. This coprocessor is based on projective coordinates, and the latency of the divisor multiplication is approximately 2.03 ms. The most recent result based on affine coordinates was presented in [12]. The authors show that an HECC coprocessor can be used not only for high-performance applications but also for resource-restricted applications because of its high performance and low hardware requirements. The main difference between the work in [12] and our study is that our study is targeted for projective coordinates (in order to reduce the necessity for inversion operations) and that the circuit presented in [12] is designed for low power consumption.

III. Architecture of the HECC Coprocessor

The HECC coprocessor was implemented at the level of scalar multiplication and verified on an FPGA. The processor architecture includes the main control unit, arithmetic unit, and register file. The main control unit controls the scalar multiplication algorithm, arithmetic unit, and register file. Figure 1 shows the architecture of the HECC coprocessor. The arithmetic unit comprises an underlying field arithmetic unit and a group operation unit. The register file is used for storing the input/output divisor values and temporary data used in the HECC computation. Our HECC coprocessors were implemented on specific fields such as GF(2^{89}) and GF(2^{113}), and their arithmetic units were optimized with the minimal polynomials, $x^{89} + x^{38} + 1$ and $x^{113} + x^9 + 1$, respectively.

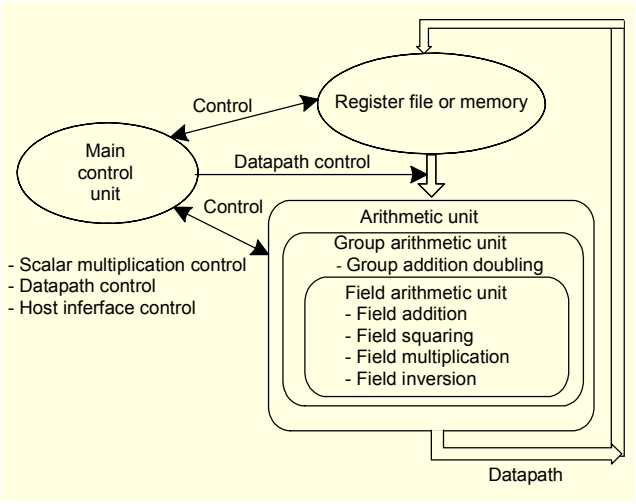

Fig. 1. Block diagram of the HECC coprocessor architecture.

The main control unit is responsible for controlling the scalar multiplication portion of the HECC coprocessor. We have implemented three scalar multiplication algorithms, including parallel binary, binary, and comb methods [12], [10], [13]. The arithmetic unit in the HECC coprocessor includes group operation units, such as group doubling and group addition. These are based on the explicit formulae [8], [12]. The group arithmetic units are responsible for field operations, such as field addition, squaring, multiplication, and inversion over a binary field. Since the HECC coprocessor is realized over a characteristic two field, the field addition operation is realized with modular two operations (that is, an exclusive-OR operation). Field squaring is realized with simple expansion and reduction based on an underlying minimal polynomial. The field multiplication and inversion operations are more complex, and we have designed these two field operation units to be fast, power efficient, and scalable.

1. Field Inversion

The field inversion unit is implemented with a modified almost inverse algorithm (MAIA), which is modified by applying a loop unrolling technique. By this technique, the MAIA with unrolling level 4 can complete its four internal iteration loops in just a single loop (see [12]). We have also implemented the extended Euclidean algorithm (EEA) on hardware by using the loop unrolling technique. As shown in Table 3, the operating frequency of the MAIA algorithm is 8.5% and 87.6% faster than that of the EEA algorithm with unrolling levels 1 and 4, respectively. Moreover, the hardware complexity of the MAIA algorithm is less than that of the EEA algorithm; hence, we have chosen the MAIA algorithm for the inversion operation of the HECC coprocessor.

Table 3. Features of the EEA and MAIA algorithms (Target platform: Xilinx XC2C1000bg575-6).

Field	Algorithm	Unrolling level	# of LUTs	Frequency (MHz)
$GF(2^{233})$	EEA	1	1,754	89.2
		4	4,161	51.8
	MAIA	1	1,421	96.8
		4	3,886	97.2

Table 4. Features of the inversion unit of the HECC coprocessors.

Types	Field size	Unrolling level	# of slices	Frequency (MHz)	Clock cycles	TTC* (μs)
MAIA with loop unrolling	$GF(2^{89})$	1	303	116.4	178	1.53
		2	547	65.0	120	1.85
		3	679	100.0	103	1.03
		4	733	97.6	97	0.99
	$GF(2^{113})$	1	387	95.1	226	2.38
		2	481	75.3	152	2.02
		3	856	82.9	131	1.58
		4	954	82.1	123	1.50

* TTC: time to completion

The loop unrolling technique has two major positive effects on the inversion unit: high performance and low power consumption. The detailed algorithm may be found in [12]. Table 4 shows the characteristics of an inversion unit targeted for implementation on an FPGA. As the unrolling level increases, the clock cycles required to complete the field inversion operation decrease while maintaining a reasonable operating frequency. For example, when moving from unrolling level 1 to level 4 in the case of $GF(2^{89})$, the clock cycles drop from 178 to 79 cycles; however, the operating frequency is reduced by only 19 MHz from 116 MHz to 97 MHz. Therefore, the latency of the inversion logic is reduced as the unrolling level increases at the cost of hardware complexity.

In Fig. 2, we have re-drawn the latency and maximal operating frequency characteristics of the inversion unit based on the data in Table 4. The TTC of the inversion unit over $GF(2^{113})$ decreases as the unrolling level increases. However, for the case of $GF(2^{89})$, the TTC of the inversion unit at unrolling level 2 is worse than that at unrolling level 1. If we unroll the inversion algorithm, then the number of clock cycles required to perform its operation certainly decreases. However, the latency of the inversion unit increases due to the data-path of the inversion unit, for which each cycle of operation becomes

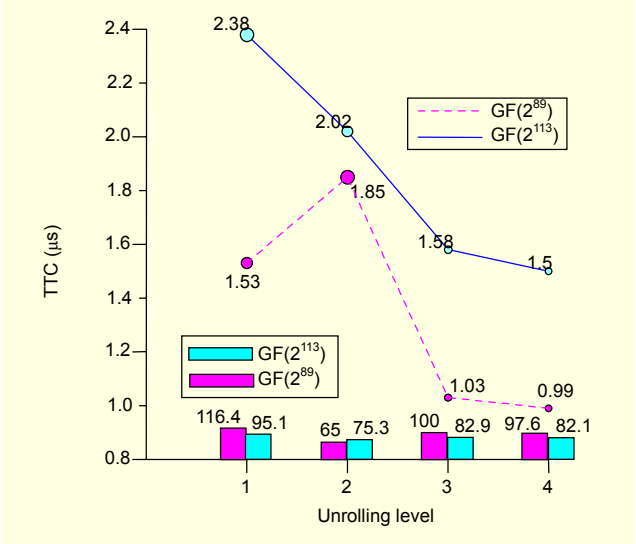

Fig. 2. Latency and operating frequency of the inversion units of the HECC coprocessors versus the unrolling level.

longer. In the case of inversion logic with unrolling level 2 over GF(2^{89}), increasing datapath delay has a much greater influence than the shortening of clock cycles. Thus, the overall performance of an inversion unit in which 2 loops are unrolled is worse than that in which loop unrolling is not applied. Based on these results, we can apply the loop unrolling technique to HECC coprocessor design in order to meet the desired trade-offs between performance and hardware complexity.

2. Field Multiplication

For field multiplication, we used the digit-serial multiplier introduced in [14] and [15]. This kind of multiplier allows trade-offs among the speed, area, and power consumption. Table 5 shows the implementation results for the least significant digit (LSD)-first multiplier. As seen in this table, the TTC of the multiplier decreases as the digit size increases; however, the hardware complexity (number of slices) also increases. We can choose the proper digit size by considering the target coprocessor's requirements. That is, if the coprocessor requires fast execution for field multiplication despite the high hardware complexity, a large-digit-size multiplier will be chosen.

In Fig. 3, we have re-drawn the latency plot of the multiplication unit based on the data in Table 5. The TTC of the multiplication units over the different fields decreases as the digit size increases. This result is straightforward and is as we had expected. However, when we consider the factor of the maximal operating frequency versus the digit size, we find that the case with digit sizes of 8 and 4 over GF(2^{89}) and GF(2^{113}), respectively, has the highest maximal operating frequency. We

Table 5. Features of the multiplication unit of the HECC coprocessors.

Types	Field size	Digit size (bits)	# of slices	Frequency (MHz)	Clock cycles	TTC (μs)
Digit-serial multiplier	GF (2^{89})	1	145	97.5	89	0.913
		4	239	106.8	23	0.215
		8	414	110.1	12	0.109
		16	645	87.4	6	0.069
		32	1,189	71.6	3	0.042
		45	1,616	63.7	2	0.031
		89	3,205	52.2	1	0.019
	GF (2^{113})	1	294	84.8	113	1.345
		4	366	106.6	29	0.282
		8	614	93.2	15	0.172
		16	931	86.4	8	0.104
		32	1,931	85.3	4	0.059
		38	2,288	83.3	3	0.048
		57	3,195	68.2	2	0.044
		113	7,278	63.1	1	0.032

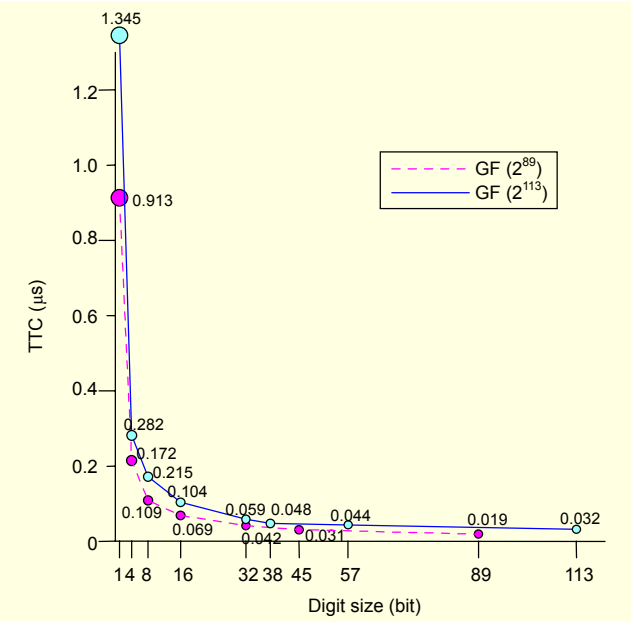

Fig. 3. Latency of the multiplication units of the HECC coprocessors versus the digit size.

can explain this if we only consider the datapath of the multiplication unit. The critical path of the multiplication unit in the case with digit sizes of 1 and 4 over GF(2^{89}) is not on the datapath but along the control logic of the multiplication unit. The same reason can explain why the case of a multiplication unit over GF(2^{113}) with a digit size of 1 has a lower operating

frequency than the cases with digit sizes of 4, 8, 16, and 32.

3. Architectural Characteristics of the HECC Coprocessor

The proposed HECC coprocessor includes several architectural features for attaining high performance. First, field operation units, such as the field multiplier and inversion logic units, are designed for high performance and low power. Second, field addition and squaring operations are executed during data movement between the register file and the field multiplier and inversion logic units. Third, the interconnect network between the register file and the field operation units is carefully designed for low hardware complexity and low latency. Since the HECC explicit formulae require several tens of multiplication operations to complete the scalar multiplication, the interconnect network (which is implemented with multiplexers or a bus) is complex. Thus, the issue of register allocation and multiplier assignment is important for high performance and low hardware complexity. We described these issues in [12] for designing an affine-coordinate-based HECC coprocessor, and the same methodology has been used to design the projective-coordinate-based HECC coprocessor presented in this work.

Table 6 shows the architectural characteristics of the HECC coprocessor over affine and projective coordinates. In the explicit formulae over affine coordinates, group doubling requires 1 field inversion, 9 field multiplications, and 6 field squarings. Group addition requires 1 field inversion, 21 field multiplications, and 3 field squarings [12]. Also, for the explicit formulae over projective coordinates, as seen in Tables 1 and 2, 31 multiplications and 7 squarings are necessary for group doubling and 45 multiplications and 4 squarings are necessary for group addition.

We implemented three different HECC coprocessors, which are referred to as types 1, 2, and 3. The type-1 HECC coprocessor is designed for high performance, and the type-2 and type-3 versions are designed for lower hardware complexity while maintaining reasonable performance. The type-1 coprocessor has two independent group operation units, consisting of group addition and group doubling logic. It performs group addition and doubling in parallel. The type-2 coprocessor has one group operation unit and shares the field arithmetic unit, register file, interconnect logic, and control logic. There are two major differences between types 2 and 3. Type 3 uses memory for storing data and a bus for the interconnect network, whereas type 2 uses registers for storing data and multiplexers for the interconnect network.

As shown in Table 6, there are more registers in the projective-coordinate-based HECC coprocessor than in the affine-coordinate-based coprocessor. This is because the explicit formulae for projective coordinates are more complex than those for affine coordinates.

Table 6. Architectural characteristics of HECC coprocessors.

Coord	Types	Logic	Inter-connect	Scalar mult. method	Storage for RF
Affine coord	Type 1	Addition: 2 mult. + 1 inv. Doubling: 1 mult. + 1 inv.	MUX	Parallel binary (R to L)	13 REGs
	Type 2	2 mult. + 1 inv. (shared)	MUX	Binary (L to R)	10 REGs
	Type 3	2 mult. + 1 inv. (shared)	MUX BUS	Binary (L to R)	Memory (14 entries)
Proj* coord	Type 1	Addition: 2 mult. Doubling: 1 mult.	MUX	Parallel binary (R to L)	25 REGs
	Type 2	2 mult. (shared)	MUX	Binary (L to R)	20 REGs
	Type 3	2 mult. + 1 inv. (shared)	MUX BUS	Binary (L to R)	Memory (25 entries)

* The core logic of the HECC coprocessor over projective coordinates does not include inversion logic, but we have added inversion logic as a peripheral block to convert the projective results to affine results for the projective-coordinate-based HECC coprocessors. Two inversion logic units are added for the type-1 coprocessor and one inversion logic unit each is added for the type-2 and type-3 HECC coprocessors over the projective coordinates.

IV. Design Methodology for the HECC Coprocessor

We followed the typical hardware coprocessor design methodology. First, we modeled the HECC coprocessor using the VHSIC hardware description language (VHDL), and implemented it on an FPGA chip after functional simulation and verification. The operating frequency derived from the FPGA Place & Route tool was confirmed by burning the design data onto an actual FPGA chip. To estimate the power consumption of the field operation units and the HECC coprocessors, we used a switching-activity-based power estimation technique. The power estimation technique is described in the following section.

V. Results and Analysis

In this section, we present the implementation results and the analysis of these results with respect to performance, hardware complexity, and power consumption[1].

1) Note that some of the results for the HECC coprocessor over affine coordinates are presented in [12]. In that paper, we only presented the performance results for HECC coprocessors in GF(2^{89}) over affine coordinates, which have parallel binary and binary-method-based scalar multiplications.

1. Performance of the HECC Coprocessors

As shown in Tables 7 and 8, we implemented 5 different HECC coprocessors for the cases of underlying fields GF(2^{89}) and GF(2^{113}). The group orders for GF(2^{89}) and GF(2^{113}) are 2^{178} and 2^{226}, respectively. In Table 7, which shows the results for our HECC coprocessors over affine coordinates, we note that type 1 is better than types 2 and 3 in terms of the area-time product (ATP). Furthermore, as seen in the last four rows of Tables 7 and 8, the comb scalar multiplication method is better than the parallel binary or binary scalar multiplication methods [13].

We can confirm that the comb method with a 4-bit window is faster than a simple binary scalar multiplication method. In Tables 7 and 8, the ATP term refers to the normalized ATP. That is, the ATP value of the type-3 HECC coprocessor over GF(2^{89}), which is the lowest, is used to normalize the performance of the other HECC coprocessors.

Type-1 HECC coprocessors implement the group addition and doubling operation units separately and, therefore, allow for faster scalar multiplication operations than the other two types (types 2 and 3). However, the hardware complexity of the type-1 coprocessor is higher.

Table 8 shows the implementation results for the HECC coprocessors over projective coordinates. Contrary to the results shown in Table 7, from the point of view of the ATP, the type-3 coprocessor exhibits better performance than the other two types (types 1 and 2), which use parallel binary or binary scalar multiplication methods. Furthermore, type 1 is better than type 2. When we compare Tables 7 and 8, we observe that the operating frequency of the projective-coordinate-based HECC coprocessors is lower than that of the affine-coordinate-based coprocessors. The reason for this is that the larger number of field multiplications in the explicit formulae in the projective coordinate case results in increased hardware complexity in the interconnect network as compared to that in the affine case. As in the affine case, the type-3 coprocessor with the comb method exhibits the best performance because of its fast scalar multiplication method and reduced hardware complexity (fast operating frequency). Also, the type-2 coprocessor with the binary method exhibits the worst performance.

Table 7. Performance of HECC coprocessors over affine coordinates.

Field size	Scalar mult.	Type	Size (slices)	Freq. (MHz)	Time (ms)	ATP
GF (2^{89})	Parallel binary	1	9,950	62.90	0.436	2.530
	Binary	2	7,096	50.08	0.791	3.277
	Binary	3	4,995	50.54	1.020	2.973
GF (2^{113})	Parallel binary	1	11,361	59.07	0.722	4.786
	Binary	2	8,934	42.43	1.459	7.605
	Binary	3	6,436	43.47	1.767	6.638
GF (2^{89})	Comb	2	7,255	45.98	0.290	1.226
	Comb	3	5,328	55.53	0.322	1.000
GF (2^{113})	Comb	2	8,998	45.83	0.458	2.407
	Comb	3	6,753	55.49	0.482	1.898

Table 8. Performance of HECC coprocessors over projective coordinates.

Field size	Scalar mult.	Type	Size (slices)	Freq. (MHz)	Time (ms)	ATP
GF (2^{89})	Parallel binary	1	12,133	36.51	0.531	4.285
	Binary	2	8,693	27.07	0.986	5.699
	Binary	3	5,605	48.10	0.684	2.549
GF (2^{113})	Parallel binary	1	15,850	31.55	0.924	9.735
	Binary	2	11,251	25.74	1.568	11.728
	Binary	3	7,105	42.77	1.128	5.329
GF (2^{89})	Comb	2	9,454	33.50	0.282	1.772
	Comb	3	6,421	50.55	0.234	1.000
GF (2^{113})	Comb	2	11,872	27.32	0.522	4.121
	Comb	3	7,821	46.98	0.369	1.917

2. Power Consumption Characteristics of the HECC Coprocessors

In this section, we compare the power consumption characteristics of the HECC coprocessors and their basic components such as multiplication and inversion logic. Though we could estimate the power consumption of the HECC coprocessors on FPGAs by using a proper tool such as Xilinx's XPower Analyzer [16], we decided to estimate the power consumption characteristics of the HECC coprocessors on the ASIC level because power consumption estimates at the ASIC level are known to be more accurate than those at the FPGA level [17]. At the FPGA level, we cannot neglect the power consumption factors introduced by pads and routing layers. Hence, we have chosen the power estimation method at the ASIC level to understand more accurately the power consumption characteristics of the hardware architecture.

To estimate the power consumption, we first extracted the forward switching activity interchange format (SAIF) file from the VHDL code for our HECC coprocessor by using a logic-level simulation tool. This forward SAIF annotation file

provides the input and output port information as well as hierarchical information for the design. Second, we compiled and simulated our HECC coprocessors using the forward-annotation file and various test vectors as inputs. During this step, we could obtain the most important information for logic-level power estimation—the switching activity of the circuit. Finally, we simulated and estimated the power consumption with our design after synthesis with this back-annotation information.

To make our estimation of the power consumption as accurate as possible, we used an accurately modeled 0.25-µm CMOS technology file provided by a chip fabrication service provider. Furthermore, we verified that this power estimation technique affords the estimated results within 10% to 20% of the power analysis results obtained with SPICE.

In our design, we focused on applying architectural level techniques for power consumption reduction to the primitive operation logic of the HECC coprocessor (multiplier and inversion logic). We also estimated and analyzed the power characteristics of different architectures for HECC coprocessors. Our approach was motivated by the fact that several researchers have demonstrated that architectural level design decisions can have a significant impact on the power consumption of general circuits [18]. In general, the power consumption of a certain circuit is proportional to its operating frequency, as shown in the following equation: $P_D=CV^2f$, where P_D is the level of dynamic power consumption, C is the capacitance of the circuit, V is the applied voltage, and f is the operating frequency. Hence, in this work, we have used the normalized power (the power consumption on a per-MHz basis) as a metric to understand the characteristics of the multiplication unit. The normalized power consumption values are shown in Table 9.

The 4-bit digit sized (D = 4) multiplier consumes the least power, and the 8- and 16-bit digit sized multipliers consume less power than the bit-serial (D = 1) multiplier. From the point of view of energy consumption, which is the time integral of the power consumption, the bit-serial multiplier is worse than even the largest multipliers (fully parallel multipliers), such as 89- and 113-bit multipliers. This is because most of the power consumption results from the switching activity of the circuit, and the bit-serial multiplier requires shift and add operations during every clock cycle. The digit-serial multiplier multiplies the multiplicand with the digit size D in parallel, as shown in the following equations [14].

Table 9. Power consumption of the multiplication block (2.5 V, 0.25-µm CMOS process, normalized power (mW/MHz), and normalized energy (mJ/MHz)).

Digit size	1	4	8	16	32	45	89
Normalized power in GF(2^{89})	0.134	0.042	0.064	0.129	0.322	1.21	1.087
Normalized energy	11.909	0.962	0.766	0.772	0.966	2.420	1.087
Digit size	1	4	8	16	32	57	113
Normalized power in GF(2^{113})	0.17	0.032	0.068	0.173	0.454	2.302	1.976
Normalized energy	19.211	0.917	1.026	1.383	1.816	4.604	1.976

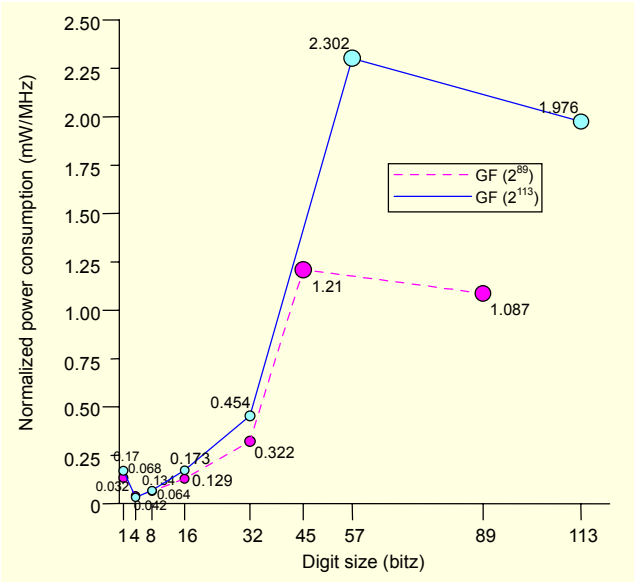

Fig. 4. Power consumption of the multiplication units of the HECC coprocessors versus the digit size (2.5 V, 0.25-µm CMOS process).

Assume a digit size of D bits. Let d denote the total number of digits with $d = \lceil m/D \rceil$. Let $A = \sum_{j=0}^{m-1} a_j \alpha^j$ and $B = \sum_{i=0}^{d-1} B_i \alpha^{Di}$, where

$$B_i = \begin{cases} \sum_{j=0}^{D-1} b_{Di+j} \alpha^j, & 0 \le i \le d-2 \\ \sum_{j=0}^{m-1-D(d-1)} b_{Di+j} \alpha^j, & i = d-1 \end{cases}.$$

Then, $C = A \bullet B \bmod p(x) = A \bullet \sum_{i=0}^{d-1} B_i \alpha^{Di} \bmod p(x)$.

From this equation, we can see that the total number of clock cycles required to perform the multiplication is d, and D bits are computed in parallel. The energy consumption levels of the 8-bit multiplier over GF(2^{89}) and that of the 4-bit multiplier over GF(2^{113}) are the lowest.

Based on this fact, we observe that the digit-serial multiplier is efficient in terms of its energy consumption. Moreover, we can select the digit size based on other factors such as the

Table 10. Power consumption of the inversion block (2.5 V, 0.25-μm CMOS process, normalized power (mW/MHz), and normalized energy (mJ/MHz)).

Unrolling level	1	2	3	4
Normalized power in GF(2^{89})	0.055	0.052	0.064	0.129
Clock cycles	178	120	103	97
Normalized energy	9.790	6.240	6.592	12.513
Unrolling level	1	2	3	4
Normalized Power in GF(2^{113})	0.063	0.064	0.086	0.082
Clock cycles	226	152	131	123
Normalized energy	14.238	9.728	11.266	10.086

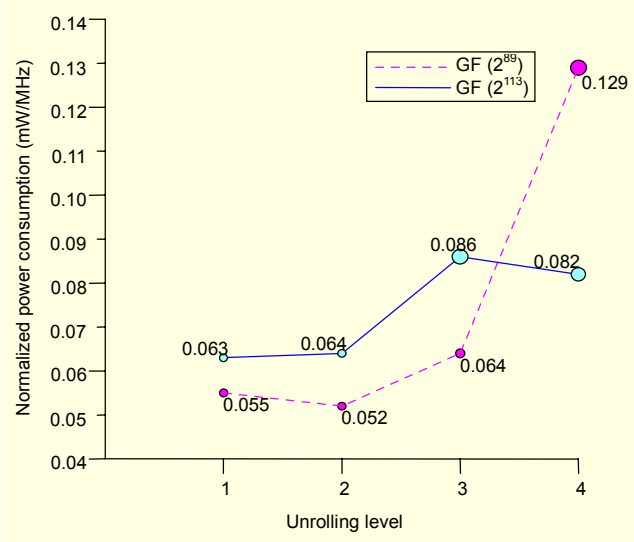

Fig. 5. Power consumption of the inversion block of the HECC coprocessors versus the unrolling level (2.5 V, 0.25-μm CMOS process).

performance and hardware size.

Table 10 shows the power consumption characteristics of the inversion logic. The inversion logic with unrolling level 1 consumes less power than others. However, from the viewpoint of the energy consumption, the inversion logic with unrolling level 1 consumes more energy than the others on GF(2^{113}). Therefore, the inversion logic with unrolling level 4 seems to be the best choice for GF(2^{113}), and level 3 seems to be the best for GF(2^{89}).

After considering the power consumption characteristics of the multipliers, we have chosen the 16-bit multiplier over GF(2^{89}) and the 32-bit multiplier over GF(2^{113}) for the HECC coprocessors. In addition, for inversion, we have chosen the logic with unrolling level 4. Table 11 shows the power consumption characteristics of the HECC coprocessors in

Table 11. Power consumption of the HECC coprocessors (2.5 V, 0.25-μm CMOS process).

Field size	Types*	Dynamic power @ 50 MHz (mW)	Leakage power (uW)	Normalized power (mW/MHz)	Clock cycles	Normalized energy (J/MHz)
GF (2^{89})	A1	37.56	6.8	0.751	27,410	20.5
	A2	27.25	4.99	0.545	39,630	21.6
	A3	19.80	5.55	0.396	51,550	20.4
GF (2^{113})	A1	43.33	9.55	0.867	42,640	36.9
	A2	31.52	6.31	0.631	61,890	39.0
	A3	24.23	7.01	0.485	76,830	37.2
GF (2^{89})	P1	120.73	10.1	2.415	19,390	46.8
	P2	92.84	6.59	1.857	26,690	49.5
	P3	98.50	24.36	1.970	32,900	64.8
GF (2^{113})	P1	159.75	13.10	3.195	29,140	93.1
	P2	116.58	8.37	2.332	40,350	94.0
	P3	117.59	30.78	2.352	48,240	113.4

* A: Affine coordinate type (A1 denotes affine coordinate type 1),
P: Projective coordinate type

affine and projective coordinates. The type-1 affine- and projective-coordinate-based HECC coprocessors with a field size of 89 bits consume 0.751 mW and 2.415 mW of normalized power, respectively. We note that the affine-coordinate-based HECC coprocessors consume less power than the projective-coordinate-based HECC coprocessors because the latter have the same number of multiplication and inversion logic units as affine coprocessors. Also, they require a larger interconnection network than the affine cases (see Table 6). As a result, they consume more energy than affine-coordinate-based coprocessors despite their faster operations.

Comparing the three types of HECC coprocessors for the affine case, we find that type 1 (A1) is faster than the others (A2 and A3) but consumes more power. This is because the A2 and A3 coprocessors share their underlying field arithmetic logic units to reduce both the power consumption and hardware complexity.

The A3 coprocessor consumes less power than the A2 coprocessor. This is because the A3 coprocessor uses memory for its storage elements and a bus for its interconnection network. The use of a bus makes its operation slower but reduces its power consumption. It is interesting to note that the A2 coprocessor consumes more energy than the A1 and A3 coprocessors. The energy consumption factor is due to its slower operation and the high power consumption characteristics of its interconnection network.

Unlike the affine-coordinate-based HECC coprocessors, the P2 (the type-2 projective-coordinate-based HECC) coprocessor

consumes less power than the P1 and P3 ones. In the case of projective-coordinate-based HECC coprocessors, the number of bus transaction operations is greater than that in the cases of the affine-coordinate-based coprocessors. Therefore, the P2 coprocessor consumes less power than the P3 coprocessor despite the high complexity of its interconnection network. In performance metrics, it is natural that the P1 coprocessor is faster than the others because of its parallel execution of scalar multiplication. In energy consumption, the P1 coprocessor is the least demanding. The P1 coprocessor's faster completion of scalar multiplication operations makes it consume less energy than the others. Thus, from these data and [19], the power consumption of the HECC coprocessors is less than 3.0 mW/MHz except in the case of P1 on GF(2^{113}). Hence, we can say that our HECC coprocessors can be used in resource-constrained environments such as in smart cards[2)].

3. Performance Comparisons

In this section, we compare our HECC coprocessors with those of previous works and ECC coprocessors. We have chosen to use the same methodology and the same FPGA chip model as in previous work to allow a fair comparison. The performance comparison results are shown in Table 12. To ensure a fair comparison, we have also omitted our results for comb-method-based coprocessors despite their higher performance compared with those using the binary method. The HECC and ECC crypto-coprocessors are compared use a binary or parallel binary method for their scalar multiplications, except for Elias' crypto-coprocessor.

Analyzing the ATP values for our designs, we observe that the type-3 HECC coprocessor over projective coordinates is the best; therefore, this value is normalized to 1. The second most efficient coprocessor in terms of the ATP is the type-1 coprocessor over affine coordinates, which is the best-performing (low TTC value) device among our coprocessors.

Our HECC coprocessors perform between 6 and 140 times better than previously reported HECC coprocessors. In terms of the ATP, our 226-bit type-3 coprocessor over projective coordinates performs 6 times better than Elias' device with a 226-bit key. Comparing the results for the affine- and projective-coordinate-based coprocessors, we can state that neither outperforms the other. This data suggests that from a practical viewpoint, projective coordinates have both positive and negative aspects. Our projective-coordinate-based HECC coprocessor uses field inversion logic to convert its computed scalar multiplication results into affine coordinates. Omitting the inversion logic in a projective-coordinate-based coprocessor is not practical because performing such a conversion using software results in considerably more overhead for a given system.

Table 12. Performance comparison with ECC and HECC coprocessors.

Type	Coord type	Scalar mult.	Key size	# of slices	Freq (MHz)	TTC (ms)	ATP
Cla03 [20]	Proj	Parallel binary	166 bits	22,000	-	10.0	57.39
				60,000	-	9.0	140.85
Elias [11]	Proj	NAF	226 bits	21,550	45.6	7.39	41.54
				25,271	45.3	2.03	13.38
Type 1	Affine	Parallel binary	178 bits	9,950	62.90	0.436	1.13
Type 2		Binary		7,096	50.08	0.791	1.47
Type 3		Binary		4,995	50.54	1.020	1.33
Type 1		Parallel binary	226 bits	11,361	59.07	0.722	2.14
Type 2		Binary		8,934	42.43	1.459	3.40
Type 3		Binary		6,436	43.47	1.767	2.97
Type 1	Proj	Parallel binary	178 bits	12,133	36.5	0.531	1.68
Type 2		Binary		8,693	27.1	0.986	2.24
Type 3		Binary		5,605	48.1	0.684	1.00
Type 1		Parallel binary	226 bits	15,850	31.5	0.924	3.82
Type 2		Binary		11,251	25.7	1.568	4.60
Type 3		Binary		7,105	42.8	1.128	2.09
ECC	Orlando et al. [21]		167 bits	1,501	76.7	0.210	-
	Gura et al. [22]		163 bits	11,845	66.4	0.143	0.44

Table 12 shows that our devices are about 5 times faster than previous hardware implementations and at least 13 times better in terms of the ATP (when compared with Elias' results). This higher performance can be the result of the hardware design features, such as loop unrolling, optimized scheduling, parallel execution of arithmetic operations, and trade-off design of the digit-serial multiplier. In Table 12, we can see that the ATP performance of the ECC coprocessors is 2.2 times better than that of the HECC coprocessors. Though HECC still requires an upgrade to outperform ECC, HECC can be considered a candidate public-key crypto-system for practical application environments.

VI. Conclusion

In this paper, we have implemented three different HECC coprocessors in affine and projective coordinates and analyzed

2) In [19], the authors state that the typical power consumption of a 13.56-MHz contactless smart card is about 30 mW when operated at around 10 MHz (3.0 mW/MHz). This is on the order of the power consumption of our HECC coprocessors.

their performance, hardware complexity, and power consumption. Among our HECC coprocessors, the type-1 HECC coprocessor over affine coordinates with a field size of GF(2^{89}) executes its scalar multiplication operation within 0.436 ms. This is about 5 times faster and 13 times better than previous results in terms of the area and time products. Also, the power consumptions of our HECC coprocessors are less than 30 mW, which is known as the typical power consumption value of a contactless smart card. Moreover, the implemented HECC coprocessors are scalable because their field operation units are scalable.

Based on computational and experimental results with FPGA implementations, we conclude that our HECC coprocessor achieves higher performance than those of previous works due to

- its efficient explicit formulae;
- its high-performance inversion logic;
- the high operating frequency of the multiplier design despite its large digit size;
- its reduced interconnect network latency achieved through the use of a buffer allocation mechanism and an efficient arithmetic unit design;
- its parallel execution of field and group operations;
- and its pipelined execution of field operations and data movement between register files.

Therefore, our HECC coprocessor is suitable for high-performance applications, such as public-key crypto-server systems. It is also suitable for resource-constrained environments, such as PDAs and smart cards, because of its high performance and moderate power consumption. Comparisons with an ECC coprocessor show that our HECC coprocessor performs comparably.

Secure implementation of cryptographic algorithms is another important issue in addition to the speed, area, and power consumption. Especially in small devices such as smart cards, it is necessary to have an implementation that is immune to side-channel attacks, such as timing analysis and power analysis attacks [23], [24]. While we have concentrated on efficiency in this paper, the implementation of secure HECC operations would be an interesting research direction. Finally, since the publication of [12], there have been some advances in the research on HEC algorithms, including Lange and Stevens's faster doubling formulas [25]. Hence, it would be another interesting possibility to upgrade our HECC coprocessor into one equipped with these latest algorithms.

References

[1] B. Schneier, *Applied Cryptography*, John Wiley & Sons, New York, 1996.

[2] N. Koblitz, "CM-Curves with Good Cryptographic Properties," *Advances in Cryptology-CRYPTO'91*, 1992, pp. 279-287.

[3] D. Mumford, "Tata Lectures on Theta II," *Progress in Mathematics,* vol. 43, Birkauser, Boston, 1984.

[4] N. Theriault, "Index Calculus Attack for Hyperelliptic Curves of Small Genus," *Proc. ASIACRYPT*, 2003, pp. 79-92.

[5] D.G. Cantor, "Computing in the Jacobian of a Hyperelliptic Curve," *Mathematics of Computation*, vol. 48, no. 177, 1987, pp. 95-101.

[6] P. Gaudry and R. Harley, "Counting Points on Hyperelliptic Curves over Finite Fields," *Proc. ANTS IV*, 2000, pp. 297-312.

[7] T. Wollinger, *Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems*, PhD. thesis, Ruhr-Universitaet Bochum, Germany, 2004.

[8] T. Lange, "Formulae for Arithmetic on Genus 2 Hyperelliptic Curves," *Applicable Algebra in Engineering Communication and Computing (AAECC)*, vol. 15, no. 5, 2005, pp. 295-328.

[9] J. Pelzl, T. Wollinger, and C. Paar. "High Performance Arithmetic for Special Hyperelliptic Curve Cryptosystems of Genus Two," *Proc. ITCC 2004*, 2004, pp. 513-517.

[10] N. Boston, T. Clancy, Y. Liow, and J. Webster, "Genus Two Hyperelliptic Curve Coprocessor," *Proc. CHES 2002*, 2003, pp. 383-397.

[11] G. Elias, A. Miri, and T.H. Yeap, "High-Performance, FPGA-Based Hyperelliptic Curve Cryptosystems," *Proc. 22nd Biennial Symposium on Communications*, 2004.

[12] H.W. Kim, T. Wollinger, Y.J. Choi, K. Chung, and C. Paar, "Hyperelliptic Curve Coprocessors on a FPGA," *Proc. WISA*, 2004, pp. 360-374.

[13] P.C. van Oorschot, A.J. Menezes, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Inc., Boca Raton, FL, 1996.

[14] L. Song and K.K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *Journal of VLSI Signal Processing Systems*, vol. 19, no. 2, 1998, pp. 149-166.

[15] C.H. Kim, C.P. Hong, and S. Kwon, "A Digit Serial Multiplier for Finite Field GF(2^m)," *IEEE Transactions on VLSI*, vol. 13, no. 4, 2005, pp. 476-483.

[16] http://www.xilinx.com, Xilinx XPower Analyzer.

[17] S. Hassoun and T. Sasao, *Logic Synthesis and Verification*, Kluwer International, Norwell, MA, 2001.

[18] S. Sheng, A. Chandrakasan, and R.W. Brodersen, "A Portable Multimedia Terminal," *IEEE Communication Magazine*, vol. 30, no. 12, 1992, pp. 64-75.

[19] C.P. Yu, C.S. Choy, H. Min, C.F. Chan, and K.P. Pun, "A Low Power Asynchronous Java Processor for Contactless Smart Card," *Proc. ASP-DAC*, 2004, pp. 553-554.

[20] T. Clancy, "FPGA-based Hyperelliptic Curve Cryptosystems," invited paper presented at *AMS Central Section Meeting*, April 2003.

[21] G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)," *CHES 2000*, 2000, pp. 41-56.

[22] H. Eberle, N. Gura, and S. Chang-Shantz, "A Cryptographic Processor for Arbitrary Elliptic Curves over GF(2^m)," *ASAP 2003*, 2003, pp. 444-454.

[23] C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," Proc. Advances in Cryptology-CRYPTO, 1996, pp. 104-113.

[24] C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Proc. Advances in Cryptology-CRYPTO*, 1999, pp. 388-397.

[25] T. Lange and M. Stevens, "Efficient Doubling on Genus Two Curves over Binary Fields," *Selected Areas in Cryptography*, LNCS 3357, 2004, pp.170-181.

Howon Kim received the BSEE degree from Kyungpook National University, Daegu, Rep. of Korea, in 1993, and the MS and PhD degrees in electronic and electrical engineering from Pohang University of Science and Technology (POSTECH), Pohang, Rep. of Korea, in 1995 and 1999, respectively. From July 2003 to June 2004, he studied with the COSY group at the Ruhr-University of Bochum, Germany. He was a senior member of technical staff at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Rep. of Korea. He is currently working as an assistant professor with the Department of Computer Engineering of Pusan National University, Busan, Rep. of Korea. His research interests include RFID technology, sensor networks, information security, and computer architecture. Currently, his main research focus is on mobile RFID technology and sensor networks, public key cryptosystems and their security issues. He is a member of the IEEE, IEEE Computer Society, and IACR.

Thomas Wollinger obtained his BS degree from the University of Dieburg in 1998. From 1999 to 2001, he had the opportunity with the support of a Fulbright grant and a corporate graduate fellowship to obtain his Master of Science at the WPI with emphasis on cryptology in Worcester (USA). In June 2003, he obtained his PhD with honors from the University of Bochum. His research interests are in improvements of hyperelliptic curve cryptosystems, fast software and hardware implementations of cryptographic algorithms, all aspects of embedded security, and side-channel attacks.

Doo-Ho Choi received his BS degree in mathematics from Sungkyunkwan University, Seoul, Rep. of Korea in 1994, and the MS and PhD degrees in mathematics from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea in 1996 and 2002, respectively. He has been a senior researcher in Electronics and Telecommunications Research Institute (ETRI) since Jan. 2002. His research interests include security technologies of RFID and sensor networks. He is an editor of the ITU-T X.nidsec-1.

Dong-Guk Han received his BS and MS degrees in mathematics from Korea University in 1999 and 2002, respectively. He received his PhD in information security engineering from Korea University in 2005. He was a post doctoral researcher with the Future University-Hakodate, Japan. After finishing the doctor course, he was an exchange student with the Department of Computer Science and Communication Engineering of Kyushu University from April 2004 to March 2005. He has been a senior researcher with ETRI since June 2006. He is a member of KIISC, IEEK, and IACR.

Mun-Kyu Lee received the BS and MS degrees in computer engineering from Seoul National University in 1996 and 1998, respectively, and the PhD degree in electrical engineering and computer science from Seoul National University in 2003. From 2003 to 2005, he was a senior engineer at ETRI. He is currently with the School of Computer Science and Engineering of Inha University, Rep. of Korea. His research interests are in the areas of information security and theory of computation.