The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

A Flexible, Prime-Field, Genus 2 Hyperelliptic-Curve Cryptography Processor with Low Power Consumption and Uniform Power Draw

Hamid-Reza Ahmadi, Ali Afzali-Kusha, Massoud Pedram, and Mahdi Mosaffa

This paper presents an energy-efficient (low power) prime-field hyperelliptic-curve cryptography (HECC) processor with uniform power draw. The HECC processor performs divisor scalar multiplication on the Jacobian of genus 2 hyperelliptic curves defined over prime fields for arbitrary field and curve parameters. It supports the most frequent case of divisor doubling and addition. The optimized implementation, which is synthesized in a 0.13 μm standard CMOS technology, performs an 81-bit divisor multiplication in 503 ms consuming only 6.55 μJ of energy (average power consumption is 12.76 μW.) We also present a technique to make the power consumption of the HECC processor more uniform and lower the peaks of its power consumption.

Keywords: Energy-efficient implementation, average power consumption, uniform power consumption, arithmetic processors, hyperelliptic curve cryptography (HECC) processor.

I. Introduction

Secure communication is an essential part of many current and anticipated applications of wireless sensor networks (WSNs) and radio-frequency identification (RFID) systems [1][2]. Devices used in these systems, including WSN motes, RFID tags, and contactless smart cards, usually have to operate with very low power consumption and high energy efficiency [1]-[3]. Therefore, since the security of digital communications is based on cryptography algorithms, there will be a need for energy-efficient (low average power) implementations of cryptography algorithms for these devices [1]-[3]. To meet the limitations of these devices, some researchers have proposed security protocols based on symmetric cryptography (for example, [4]) while others have used the so-called light-weight cryptography algorithms and protocols (for example, [5]-[6]). However, [7] gives reasons why standardized public-key cryptography (PKC) algorithms should be implemented and used in power/energy-limited applications and devices. This line of reasoning is also confirmed by the availability of commercial solutions equipped with standard PKC hardware blocks.

Considering similar reasoning to that of [7], many research groups (for example, [8]-[11]) have tried to implement standard PKC algorithms for the mentioned power/energy-limited devices. Among well-known PKC algorithms, the elliptic-curve cryptography (ECC) and the hyperelliptic-curve cryptography (HECC) are suitable candidates for energy-efficient implementations, since they provide higher security per bit of the operands [12]-[13]. While ECC has received more attention in recent years (for example, [8]-[9]), only few reports have been given for energy-efficient designs of HECC [10][11]. The

Manuscript received Apr. 04, 2014; revised Sept. 21, 2014; accepted Oct. 2, 2014.

Hamid-Reza Ahmadi (corresponding author, hrahmadi@ut.ac.ir) is with the Faculty of New Sciences and Technologies, University of Tehran, Tehran, Iran.

Ali Afzali-Kusha (afzali@ut.ac.ir) and Mahdi Mosaffa (m.mosaffa@ut.ac.ir) are with the School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran.

Massoud Pedram (pedram@usc.edu) is with the Department of Electrical Engineering, University of Southern California, Los Angeles, USA.

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

advantage of HECC is that it only requires half the operand length (or less) to provide the same security level as that of ECC with the disadvantage of being more complex than ECC [13]. Shorter operand length makes HECC more attractive than ECC for applications with constrained power/energy sources [13]. In this paper, we present an energy-efficient low-power HECC processor design.

The rest of this paper is organized as follows. Section II gives a simplified explanation of the mathematics of HECC and Section III summarizes the related works. Section IV describes the characteristics of our HECC processor, while Section V gives the details of the implemented algorithms. Section VI describes the architecture of the HECC processor and gives the exact register-mapping of the algorithms. Section VII discusses the results obtained for different implementations of the HECC processor and Section VIII describes a technique used to make the power consumption of the HECC processor more uniform. Finally, Section IX concludes the paper.

II. Mathematics of Hyperelliptic-Curve Cryptography

In this section, we present a simplified description of the underlying mathematics of hyperelliptic curves and HECC. Our explanations will be focused on genus 2 hyperelliptic curves defined over fields of odd characteristic (that is, prime fields) which are the base for our HECC processor design. More on the theory of hyperelliptic curves and HECC may be found in [13].

For our purpose, a hyperelliptic curve C of genus 2 defined over a prime field $GF(p)$, is the set of all points (x, y) with x and $y \in GF(p)$, satisfying

$$y^2 \equiv x^5 + f_3 x^3 + f_2 x^2 + f_1 x + f_0 \pmod{p} \tag{1}$$

where all $f_i \in GF(p)$ and the right hand side function should not have any multiple roots. We also note that, to obtain the above equation for the curves, some simplifying changes of variables have been made without altering the hyperelliptic curve properties [13]. By taking into account a point P_∞ on the curve C, called the *'point at infinity'*, an algebraic group can be defined. Members of this group, which is called the Jacobian of C, are finite sums of the points of the hyperelliptic curve [13].

For the curves of genus 2 defined above, each element of the Jacobian, denoted by J_C, is made from a finite sum of two points of the curve and called a *'reduced divisor'* [13]. Using the set of reduced divisors, then, a *'divisor addition'* operation can be defined on the Jacobian group as

$$\begin{aligned} &P_1 = (x_{P_1}, y_{P_1}),\ P_2 = (x_{P_2}, y_{P_2}) \in C \rightarrow D_1 \in J_C \\ &Q_1 = (x_{Q_1}, y_{Q_1}),\ Q_2 = (x_{Q_2}, y_{Q_2}) \in C \rightarrow D_2 \in J_C \\ &D_1, D_2 \in J_C \rightarrow \ \ D_1 \oplus D_2 = D_3 \in J_C \\ &D_3 \in J_C \rightarrow R_1 = (x_{R_1}, y_{R_1}),\ R_2 = (x_{R_2}, y_{R_2}) \in C \end{aligned} \tag{2}$$

To perform the above divisor addition, in the general case, first a function $y=S(x)$ of degree 3 in x is found which passes through the four points of D_1 and D_2. This function intersects curve C in two more points, which are $-R_1$ and $-R_2$. The two points of D_3 can easily be found by negating the results [13]. In practice, however, doing the calculations on the point coordinates is difficult, and therefore, a special representation (called the Mumford representation) of the divisors is used. This representation results in an efficient algorithm for adding the reduced divisors [13].

For genus 2 hyperelliptic curves, each reduced divisor on the Jacobian (made from two distinct points of the curve) can be represented with a pair of polynomials [13], defined in the Mumford representation as

$$\begin{aligned} &P_1 = (x_{P_1}, y_{P_1}),\ P_2 = (x_{P_2}, y_{P_2}) \in C \rightarrow D \in J_C \\ &D = div(\,u(x),\, v(x)\,) = [\,u(x),\, v(x)\,] \\ &u(x) = x^2 + u_1 x + u_0 = (x - x_{P_1})(x - x_{P_2}) \\ &v(x) = v_1 x + v_0,\ v(x_{P_1}) = y_{P_1},\ v(x_{P_2}) = y_{P_2} \end{aligned} \tag{3}$$

For the reduced divisors represented in the above form, an efficient general formula exists for the divisor addition operation [13]. This formula should be written in the explicit form for hardware implementation. The *'divisor doubling'* operation for adding a divisor to itself is similarly defined, and the corresponding explicit form is written. The details of our implemented divisor addition and doubling algorithms for the HECC processor will be given in Section V.

Having defined the divisor addition and also divisor doubling operations, next we define a *'scalar multiplication'* operation which multiplies a divisor by an integer using repeated additions (and/or doublings):

$$E = [k]\,D = \underbrace{D \oplus D \oplus D \oplus \ldots \oplus D}_{add\ \ k-1\ \ times} \tag{4}$$

Having the value of k and the divisor D, the resulting divisor E can be found in polynomial time by performing the scalar multiplication. On the other hand, finding the value of k for two known divisors D and E, requires exponential-time calculations. Therefore, cryptographic protocols can be designed based on the scalar multiplication over Jacobian of a hyperelliptic curve as defined above [13].

III. Related Work

In recent years, considerable attention has been given to

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

HECC as a means of enabling PKC in embedded and also resource-constrained applications. One line of research has focused on finding efficient explicit formulae for calculating the group operations on hyperelliptic curves and also improving the efficiency of calculations by reducing the number of operations. Examples include works presented in [16]-[19] which have covered both prime-field and binary-field curves. These works have used different coordinate systems (both affine and projective) to improve the efficiency of their formulae. Another area of research has considered hardware implementations of HECC mostly with the goal of improving the speed of calculations ([20]-[27]). All of these research activities used binary-field curves as a way to reduce the hardware area for possible implementations in embedded applications. Only [24]-[27] have provided power consumption results of their corresponding designs.

To the best of our knowledge, no report exists on energy-efficient hardware implementation of HECC using hyperelliptic curves defined over prime fields.

IV. Features of the Proposed Design

In this section, we describe the features of our HECC hardware processor design.

1. Hyperelliptic Curves of Genus 2

Hyperelliptic curves of genus 2 ($g = 2$) and genus 3 ($g = 3$) are considered more secure than $g > 3$ curves [13][14]. Between the genus 2 and the genus 3 curves, the former ones have lower complexity and lower calculation times [13] and also have been studied more. Therefore, we chose the genus 2 curves.

2. Curves Defined over Prime Fields

It is usually assumed that using the binary fields is the better choice when designing curve-based cryptography hardware. The main reasons are carry-free addition and simple squaring operations in the binary fields [12]. While these are attractive features for high-speed designs, they do not have the same importance in low-power designs (see, [9].) Since our main goal is a low-power design and also based on other reasons explained in [9], we chose the prime field over the binary field.

3. Flexibility in Field and Curve Parameters

To design a curve-based cryptography processor with low power and energy consumption, it is desired to keep the hardware as small as possible. One way to achieve this is to fix the parameters defining the field and the curve [12]. Fixing the parameters, however, has two drawbacks. One is that the resulting hardware will be optimized for one particular set of parameters, and therefore, cannot be used by other systems/users who may want to use other parameters. The second drawback is related to the case where a security breach occurs in the system making the used set of parameters obsolete. In such a condition, since the parameters of the hardware are fixed, the existing hardware cannot be re-used. In contrast to cryptography processors with fixed parameters, a crypto-hardware designed to support arbitrary values of field and curve parameters will lower the costs of manufacturing (since one hardware can be used for many systems/customers and may be re-used by changing the input parameters). Hence, we have chosen to implement the HECC processor for arbitrary values of parameters.

4. Implementation of Most Frequent Case Equations

As was explained in Section II, to achieve an efficient formula for adding/doubling the reduced divisors, the Mumford representation of the reduced divisors is used. In Eq. (3), we only showed the general case of the representation for reduced divisors of genus 2 curves. For the field sizes used in HECC (in the order of 2^{80} or more members), this general case will be encountered most frequently in the calculations and the probability of the occurrence of the special cases is too small (2^{-80} or less) [13]. Since the special cases of the reduced divisors rarely appear in practice, it is reasonable for a hardware implementation to only cover the general case [13]. The same approach was also taken in other implementations of HECC [19].

5. Affine Coordinate Representation in Montgomery Domain

Inspecting Eq. (3) suggests that the reduced divisors in genus 2 HECC can be stored using four variables which are the coefficients of $u(x)$ and $v(x)$. This is called the *affine* representation of the reduced divisors. In the affine form, every divisor addition/doubling requires one modulo inversion which is usually considered a time-consuming operation [12]. Other representations, such as *projective*, speed up the calculations by eliminating the inversions [12]. The resulting faster calculations lower the energy consumption by reducing the total calculation time, and therefore, many of the energy-efficient hardware designs have used inversion-free approaches [13]. It, however, should be noted that representations such as *projective*, which eliminate the inversion, usually add to the complexity of the explicit formula and need extra temporary variables (that is,

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

more storage space) as shown in [24]. The added complexity and extra storage requirement lead to higher power consumption, and may also retract the effect of lowering the calculation time on reducing the energy consumption. As another alternative, there are modulo inversion algorithms, such as Montgomery modulo inversion (MMI), which are not very time-consuming and can be used for energy-efficient implementations [9].

Based on the above discussion, and the results of [9] and [24], we decided to use the *affine* representation of divisors. In addition, we chose to perform all of the calculations on integers in Montgomery domain [13], since this allows us to use Montgomery multiplication (thus avoiding a full multiplication followed by a modulo reduction) [12] and the MMI algorithm [9].

6. Design for Low Power using Low Clock Frequency

Most of the cryptography hardware designed for use in RFID systems and WSNs, work at low clock frequencies (< 1 MHz) [8], which is an effective way of reducing the power consumption. The fact that the design will operate only with low clock frequencies should also be taken into consideration during the design. For example, the critical path of the circuit will not be important, and instead of fast adders, simple carry propagate adders can be used [9]. The same approach has been taken in this work.

7. Power Consumption as Uniform as Possible

A uniform power consumption trace for cryptography hardware units have the advantages of offering more security against simple power attacks (SPAs), resulting in longer battery life, and enabling more efficient power profiling and power management [28]. In this work, the HECC processor will be designed to have a power consumption trace as uniform as possible.

V. Implemented HECC Algorithm

As mentioned before, the topmost operation of the HECC processor is the scalar multiplication of Eq. (4). To obtain security against timing attacks, this operation is implemented using the *double-and-always-add* method [13]. The operations in the next level are the divisor addition and divisor doubling, whose details are given next.

1. Divisor Addition – Most Frequent Case

The divisor addition in the most frequent case can be performed by the equations shown in Fig. 1(a) [13]. Both the inputs and the result are in the form of Eq. (3). Also, the equations are based on the polynomial arithmetic in $GF(p)$. The equations shown in Fig. 1(a) can be written in the explicit form by expanding the polynomial operations using the coefficients of the polynomials and integer arithmetic operations in $GF(p)$. This results in the algorithm shown in Fig. 1(b) which consists of integer addition/subtraction, multiplication and one inversion [13]. All of the integer operations must be performed modulo p.

In the fourth line of the explicit formula, it is possible that the value of s_1' becomes zero. This special condition will generate a divisor of the form $[x+u_0, v_0]$ which cannot be used in most frequent case calculations [13]. Therefore, this condition will not be covered in our design.

$$
\begin{array}{l}
[u_1(x), v_1(x)] \oplus [u_2(x), v_2(x)] \\
result = [u(x), v(x)] \\
\hline
t \leftarrow (f - v_2^2)/u_2 \\
s \leftarrow \left((v_1 - v_2)/u_2\right) \bmod u_1 \\
l \leftarrow s\, u_2 \\
u \leftarrow \left(t - s(l + 2v_2)\right)/u_1 \\
u \leftarrow u \quad made \quad monic \\
v \leftarrow \left(-(l + v_2)\right) \bmod u
\end{array}
$$

(a)

$$
\begin{array}{l}
input: \quad div(u_1, v_1) = [x^2 + u_{11}x + u_{10}, v_{11}x + v_{10}] \\
input: \quad div(u_2, v_2) = [x^2 + u_{21}x + u_{20}, v_{21}x + v_{20}] \\
output: \quad [u, v] = [u_1, v_1] \oplus [u_2, v_2] = [x^2 + u_1 x + u_0, v_1 x + v_0] \\
\hline
z_1 \leftarrow u_{11} - u_{21}, \quad z_2 \leftarrow u_{20} - u_{10}, \quad z_3 \leftarrow u_{11} z_1 + z_2 \\
r \leftarrow z_2 z_3 + z_1^2 u_{10} \\
w_0 \leftarrow v_{10} - v_{20}, w_1 \leftarrow v_{11} - v_{21}, w_2 \leftarrow z_3 w_0, w_3 \leftarrow z_1 w_1 \\
s_1' \leftarrow (z_1 + z_3)(w_0 + w_1) - w_2 - w_3(1 + u_{11}), s_0' \leftarrow w_2 - w_3 u_{10} \\
w_1 \leftarrow (r s_1')^{-1} \\
w_2 \leftarrow r w_1, w_3 \leftarrow s_1'^2 w_1, w_4 \leftarrow r w_2, w_5 \leftarrow w_4^2 \\
s_0'' \leftarrow s_0' w_2 \\
l_2' \leftarrow u_{21} + s_0'', l_1' \leftarrow u_{21} s_0'' + u_{20}, l_0' \leftarrow u_{20} s_0'' \\
u_0 \leftarrow (s_0'' - u_{11})(s_0'' - z_1) - u_{10} + l_1' + 2v_{21} w_4 + (2u_{21} + z_1) w_5 \\
u_1 \leftarrow 2s_0'' - z_1 - w_5 \\
w_1 \leftarrow l_2' - u_1, w_2 \leftarrow u_1 w_1 + u_0 - l_1', v_1 \leftarrow w_2 w_3 - v_{21} \\
w_2 \leftarrow u_0 w_1 - l_0', v_0 \leftarrow w_2 w_3 - v_{20}
\end{array}
$$

(b)

Fig. 1. Divisor addition (most frequent case), (a) Polynomial arithmetic, (b) Explicit formula, [13].

2. Divisor Doubling – Most Frequent Case

The divisor doubling equations in the most frequent case are shown in Fig. 2(a) [13]. The explicit form of the equations

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

shown in Fig. 2(a) is also depicted in Fig. 2(b) [13]. Again, we have the special condition of s_1' becoming zero (sixth line of the explicit formula), which will not be covered in our design.

$$
\begin{array}{l}
[u(x), v(x)] \oplus [u(x), v(x)] \\
result \;\; = [\tilde{u}(x), \tilde{v}(x)] \\
\hline
t \leftarrow (f - v^2)/u \\
s \leftarrow t/(2v) \;\; \bmod \;\; u \\
l \leftarrow s\,u \\
\tilde{u} \leftarrow s^2 - (2vs - t)/u \\
\tilde{u} \leftarrow \tilde{u} \quad made \quad monic \\
\tilde{v} \leftarrow \big(-(l + v)\big) \;\; \bmod \;\; \tilde{u}
\end{array}
$$

(a)

$$
\begin{array}{l}
input: \quad div(u,v) = [x^2 + u_1 x + u_0, v_1 x + v_0] \\
output: \quad [\tilde{u}, \tilde{v}] = [2][u,v] = [x^2 + \tilde{u}_1 x + \tilde{u}_0, \tilde{v}_1 x + \tilde{v}_0] \\
\hline
w_0 \leftarrow v_1^2, \quad w_1 \leftarrow u_1^2, \quad w_2 \leftarrow 4w_0, \quad w_3 \leftarrow 2u_1 v_1 \\
r \leftarrow u_0 w_2 + 2v_0(2v_0 - w_3) \\
i_1' \leftarrow -2v_1, i_0' \leftarrow 2v_0 - w_3, w_3 \leftarrow f_3 + w_1, w_4 \leftarrow 2u_0 \\
t_1' \leftarrow 2w_1 + w_3 - w_4, \quad t_0' \leftarrow u_1(2w_4 - w_3) + f_2 - w_0 \\
w_0 \leftarrow t_0' i_0', w_1 \leftarrow t_1' i_1' \\
s_1' \leftarrow (i_0' + i_1')(t_0' + t_1') - w_0 - w_1(1 + u_1), s_0' \leftarrow w_0 - w_1 u_0 \\
w_1 \leftarrow (r s_1')^{-1} \\
w_2 \leftarrow r w_1, w_3 \leftarrow s_1'^2 w_1, w_4 \leftarrow r w_2, w_5 \leftarrow w_4^2 \\
s_0'' \leftarrow s_0' w_2 \\
l_2' \leftarrow u_1 + s_0'', l_1' \leftarrow u_1 s_0'' + u_0, l_0' \leftarrow u_0 s_0'' \\
\tilde{u}_0 \leftarrow s_0''^2 + 2v_1 w_4 + 2u_1 w_5 \quad , \quad \tilde{u}_1 \leftarrow 2s_0'' - w_5 \\
w_1 \leftarrow l_2' - \tilde{u}_1, w_2 \leftarrow \tilde{u}_1 w_1 + \tilde{u}_0 - l_1', \tilde{v}_1 \leftarrow w_2 w_3 - v_1 \\
w_2 \leftarrow \tilde{u}_0 w_1 - l_0', \tilde{v}_0 \leftarrow w_2 w_3 - v_0
\end{array}
$$

(b)

Fig. 2. Divisor doubling (most frequent case), (a) Polynomial arithmetic, (b) Explicit formula, [13].

To favor readability in the explicit formulae of Fig. 1(b) and Fig. 2(b), many temporary variables are used which are not necessary for the actual implementation. The implementation and the related issues will be described next.

VI. Architecture of the HECC Processor

In this section, we will describe the architecture of the HECC processor.

1. Arithmetic Building Blocks

We need to perform add/subtract operations modulo an arbitrary prime number p. This can be performed by two adders in one clock cycle. Also, to avoid a full multiplication followed by a modulo reduction, we will use the Montgomery multiplication. The Montgomery multiplication of two n-bit numbers can be performed in n clock cycles by a circuit consisting of two adders, two registers, and some glue logic [9]. For the HECC processor design, we can take two different approaches. In one approach, we use two separate add/subtract and multiply units ('separate' units), while in the other approach, we make use of the adders of the multiplier unit to also perform the add/subtract operations ('combined' units). The combined modulo add/subtract and Montgomery multiplier units are shown in Fig. 3 where the two registers (B and M) are used only for the Montgomery multiplication. We will implement both approaches and compare the power consumption results later.

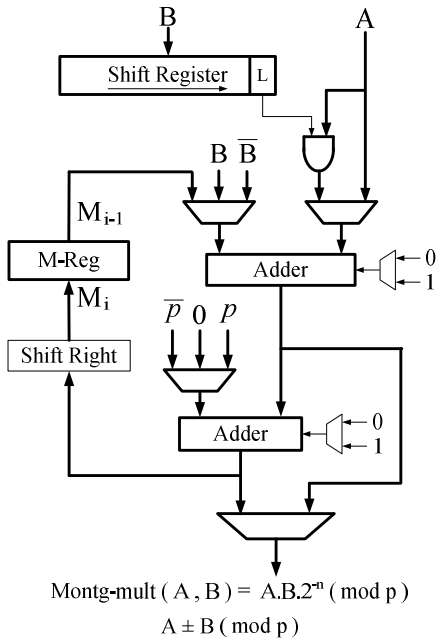

Fig. 3. Combined modulo-add/sub and Montgomery multiply circuit.

For the divisor addition of Fig. 1(b) and divisor doubling of Fig. 2(b), we need a modulo inversion operation due to the use of the *affine* coordinates for the divisors. The Montgomery modulo inverse (MMI) algorithm is a good candidate for energy-efficient implementations [9]. It can also be realized using the add/subtract and multiply units, without any need for other arithmetic units [9]. Therefore, we will use the MMI to implement the modulo inversion operation.

To better assess the effect of using separate and combined arithmetic units (explained above) on the power consumption, we will first use the inversion by exponentiation method instead of the MMI. The reason is that, contrary to the MMI, the inversion by exponentiation method can be implemented without any changes to the arithmetic units [9].

2. Temporary Storage

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

Since our HECC processor uses the *affine* coordinates, we need five variables at the top level of our design (see Fig. 4.) Four registers hold u and v coefficients of the partially multiplied divisor and one register is used for shifting of the scalar value (k). It is assumed that the original input divisor is fed to the HECC processor during the scalar multiplication.

In the next level, the hardware performs the algorithms shown in Fig. 1(b) and Fig. 2(b). We were able to limit the temporary storage of both algorithms to five extra registers. This was achieved by re-ordering and repeating some of the calculations and using the M-Reg of the multiplier unit (cf. Fig. 3) as a temporary register.

For the implementation of the inversion algorithm, we need more temporary registers. In the case of inversion by exponentiation, one more register is necessary while for the MMI, two more registers are needed. Therefore, the total number of registers (with a bit-length equal to the length of modulus p) in our design is 14 (13) for the case of MMI (inversion by exponentiation.)

2. Overall Architecture

Figure 4 shows the overall architecture of the proposed HECC processor. The field and curve parameters are input to the processor. The internal controller, named '*Divisor ADD/DBL Controller*', controls the hardware to perform the addition and doubling algorithms, as well as the inversion and multiplication operations.

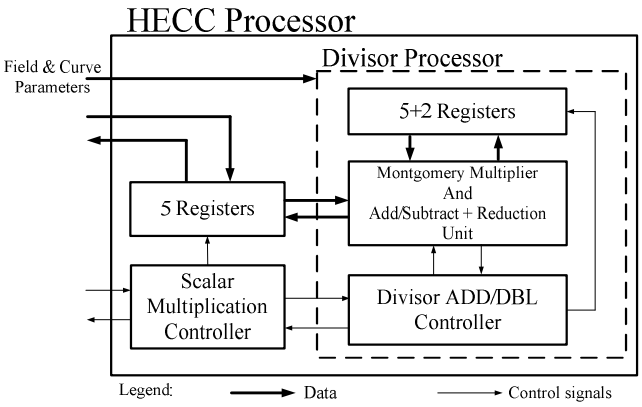

Fig. 4. Overall architecture of the HECC processor.

VII. Implementation Results

In this section, the results obtained from different implementations of the HECC processor are discussed. In the implementation process, we first coded the design in VHDL and synthesized it in a 0.13µm low-leakage standard CMOS process. Then we used the post-synthesis netlist for simulations to capture the activity of all of the circuit nodes (including glitches). The activity was then fed to a commercial power calculation tool to obtain the power consumption results. All of the simulations of our HECC processor were done for a bit-length of 81 bits, using a 1 MHz clock frequency. We used many random values for the scalar (k) and different divisors from different curves in the simulations.

1. Separate vs. Combined Arithmetic Units

In the first step, we implemented two versions of the HECC processor, one with separate add/subtract and multiply units and one with the combined unit which was shown in Fig. 3. Both used inversion by exponentiation. Note that this inversion method is very time consuming, and therefore, we only use it for the evaluation and comparison of the arithmetic units. The results of these implementations are given in Table 1.

Table 1. Results when using inversion by exponentiation.

81-bit Flexible Prime-Field genus 2 HECC processor 0.13µm low-leakage CMOS		
Implementation	Separate units	Combined unit
Area	100,518 µm^2 19739 GE*	89,382 µm^2 17552 GE*
Average Power @ **f = 1 MHz**	11.53 µW	13.49 µW
Total Clock Cycles**	2.037×10^6 double & always add	2.037×10^6 double & always add
Total Time** @ **f = 1 MHz**	2037 ms double & always add	2037 ms double & always add
Total Energy**	23.5 µJ	27.5 µJ
Maximum Frequency	34.8 MHz	33.8 MHz
Arithmetic Unit Power	10 µW (mult) 0.15 µW (add/sub)	12.08 µW
Multiplexers Power	0.74 µW	0.77 µW

* GE: Gate Equivalent – with 5.0922 µm^2 for NAND2X1 cell
** Values given for one complete divisor scalar multiplication

Table 1 shows that, as expected, using the inversion by exponentiation method causes a long calculation time and high energy consumption. For the comparison of arithmetic units, we see from Table 1 that for the '*combined unit*' implementation, the power consumption is higher. The reason is that combining the add/subtract and multiply units adds some multiplexers to the arithmetic unit consuming more power. Although for combining these two units, the structure of the multiplexers at the input ports of the arithmetic units should change, Table 1 shows that nearly the same amount of power is

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

consumed in these multiplexers. Based on this comparison, we will use separate add/subtract and multiply units for the rest of our HECC processor implementations.

2. Montgomery Modulo Inverse

The MMI algorithm, which is much faster than the inversion by exponentiation technique, can be realized with small modifications to the add/subtract unit and temporary registers. The price for the higher speed is a more complex controller and also one more temporary register, which result in higher area and power consumption. Table 2 compares the results of the MMI algorithm by those of the inversion by exponentiation using separate units.

Table 2. Results when using Montgomery modulo inverse algorithm.

81-bit Flexible Prime-Field genus 2 HECC processor 0.13μm low-leakage CMOS		
Inversion method	Montgomery	Exponentiation
Total Area	113,539 μm^2 22296 GE*	100,518 μm^2 19739 GE*
Average Power **@ f = 1 MHz**	13.46 μW	11.53 μW
Total Clock Cycles**	502,800 double & always add	2.037×10^6 double & always add
Total Time** **@ f = 1 MHz**	502.8 ms double & always add	2037 ms double & always add
Total Energy**	6.77 μJ	23.5 μJ
Maximum Frequency	34.8 MHz	34.8 MHz
Arithmetic Unit Power	8.48 μW (mult) 1.76 μW (add/sub)	10 μW (mult) 0.15 μW (add/sub)
Temp. Registers Power	1.26 μW	0.27 μW
Controller and Multiplexers Power	1.59 μW	0.74 μW

* GE: Gate Equivalent – with 5.0922 μm^2 for NAND2X1 cell
** Values given for one complete divisor scalar multiplication

The inversion by exponentiation technique repeatedly uses the multiplier unit to perform exponentiation while MMI mostly uses the add/subtract unit [9]. Also, the latter performs much more read/write from/to the temporary registers. Therefore, as can be seen in Table 2, the power consumption of the add/sub unit and the temporary registers is higher in the MMI-based implementation, while the multiplier unit has a lower power consumption. The more complex controller and multiplexers also give rise to more power consumption in the MMI case. As the results reveal, the MMI-based implementation has about 13% more area and 17% more power. The higher speed of MMI, however, leads to more than 71% reduction in the energy consumption.

It should also be noted that, in our implementation, the duration of one inversion operation using the MMI algorithm is almost seven times the duration of a multiplication. This ratio is lower than the values which justify the preference of the inversion-free coordinate types over the *affine* coordinates [13].

3. Comparison with Previous Work

As we mentioned in Section III, we are not aware of any other energy-efficient HECC implementation over the prime fields. Table 3 compares the performance of our HECC processor with those of other reported designs (all using binary fields). Although a detailed comparison is not possible due to the use of different technologies and design types, it can be observed that the prime-field HECC processor has lower power and energy levels than (or almost equal, in case of [27]) similar-sized binary-field HECC implementations. The last line of Table 3 compares our HECC processor with our previously reported ECC processor with the same security level. The shorter numbers used in the HECC (half the size used for ECC) makes it possible to lower the power by 58% and the energy by 48% compared to ECC.

Table 3. Comparison of our HECC processor with related works.

Design	CMOS Technology	Type	Total Energy (μJ)	Reported Average Power @ Clock Freq.	Calculation Time (ms)	Total Cycles
[24]	0.25μ	GF(2^{89}) g=2 HECC	20.4	396 μW @ 1 MHz	51.55	51,550
[25]	0.25μ	GF(2^{83}) g=2 HECC	Not reported	80 μW @ 1 MHz	Not reported	Not reported
[26]	0.13μ	GF(2^{83}) g=2 HECC	16.28	22 μW @ 500 KHz	740	370,000
[27]	0.13μ	GF(2^{83}) g=2 HECC	6.1	13.4 μW @ 300 KHz	456	136,838

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

This Work	0.13μ	GF(p) 81 bit g=2 HECC	6.77	13.46 μW @ 1 MHz	502.8	502,800
[11]	0.13μ	GF(p) 168 bit ECC	12.92	32.3 μW @ 1 MHz	400	400,000

VIII. Power Consumption Redistribution

In this section, we discuss the importance of uniform power consumption in cryptography circuits and describe our efforts at making the power consumption of our HECC processor more uniform by redistributing the power consumption.

1. Importance of Uniform Power Consumption

Paying attention to the temporal behavior of the power consumption and moving toward temporally uniform power consumption has the following benefits:

- The maximum value of the power consumption becomes closer to the average value preventing a waste of battery capacity. A lower maximum power value also enables the use of a smaller battery and increases the service time of the battery [29].
- More uniform power consumption eases the prediction of the remaining battery capacity and allows for more efficient power profiling and power management [29].
- More uniform power consumption makes it harder to extract information from the details of the power consumption diagram and gives more security against simple power attacks [28].

In the following subsection, we describe a technique which we applied to make the power consumption of the HECC processor more uniform.

2. Algorithm-based Partitioning of Power

Arithmetic building blocks and components such as registers consume different amounts of power during the execution of an algorithm. This originates from different input data values as well as the changes in the configuration of the blocks and the rate of their usage in the calculations. The changes in the configuration and usage rate of the blocks are determined by the algorithm. Any implemented arithmetic algorithm can be divided into separate parts such that during each part, the configuration of the blocks remains unchanged. If we obtain the average of the power consumption over each of these separated parts of the algorithm (that is, a partitioning of power), we can find which part of the algorithm consumes the largest amount of power. By focusing on the part of the algorithm with the highest average power, we can modify the algorithm and/or hardware to lower the power consumption in that part. This leads to a more uniform overall power consumption. This process may be repeated while modifications can be found to make the power consumption more uniform.

In our HECC processor implementation, such partitioning can be derived from the register-level operations. Both for the divisor addition and divisor doubling, there are two series of consecutive add/subtract and Montgomery multiplication operations with only one inversion in between. Since we have used separate add/subtract and multiply units, the configuration remains unchanged except for the inversion. Our implementation of the MMI [9] can itself be divided into three parts with different configurations. The result of the partitioning of power is shown in Table 4, for one doubling and one addition.

Table 4. Partitioning of the power based on the HECC algorithm.

Divisor Doubling			
#	Operations	Duration (μs)	Average Power (μW)
1	add/sub & Mont. mult	1248	12.54
2	inversion – convert [9]	86	9.92
3	inversion – calculation [9]	408	16.71
4	inversion – halving [9]	64	26.22
5	add/sub & Mont. mult	1328	12.67
Divisor Addition			
#	Operations	Duration (μs)	Average Power (μW)
1	add/sub & Mont. mult	974	12.61
2	inversion – convert	86	10.06
3	inversion – calculation	408	17.40
4	inversion – halving	64	26.97
5	add/sub & Mont. mult	1508	12.78

Table 4 shows that the highest value of the average power among the algorithm partitions is consumed during the third part of the inversion operation, and the next highest value is consumed during the second part of the inversion. Therefore, to obtain a more uniform power, we should try to lower the power consumption during the second and third parts of the inversion. It should be noted that, since the higher-power parts have short durations, lowering their power consumption will

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

have a small effect on the overall average power.

The power consumption values obtained for each hardware unit show that the main power consuming unit in the second and third parts of the inversion is the add/subtract unit. The unit uses two adders to perform an add/sub with reduction operation during the parts #1 and #5 of Table 4. However, during the inversion only one of the two adders is used while the other adder is operand-isolated. Our study shows that the multiplexers used in this unit are responsible for a relatively high portion of the power consumption of the unit during the inversion. To lower this power consumption, two different modifications to the HECC processor are considered. In the first modification, a single adder, as a new unit, is inserted into the HECC processor. This unit is only used during the inversion. By using a separate adder during the inversion, both of the adders of the main add/subtract unit would be always used and the operand-isolation multiplexers may be removed. In the second modification, instead of inserting a new unit, we omitted the second adder and the corresponding multiplexers from the original add/subtract unit, to reduce its power consumption during the inversion. However, with this change in the circuit, an add/subtract with the reduction operation will take two clock cycles, causing a negligible increase in the total calculation time.

The results are summarized in Table 5 which shows that both approaches lower the power consumption. The first approach (*Change* #1) lowers the power consumption in part #4 (and also #3) at the expense of a small area overhead. The second approach (*Change* #2) which is more effective in reducing the power during the inversion (parts #4 and #3), also gives a reduction in area. Thus, the second change is better. The results indicate that the changes also lower the total average power consumption of the HECC processor.

Table 5. Results of lowering power consumption during inversion.

Design	Original	Change #1	Change #2
Total Area	113,539 μm^2 22296 GE*	119,192 μm^2 23406 GE*	111,711 μm^2 21937 GE*
Average Power @ f = 1 MHz	13.46 μW	**13.17 μW**	**12.76 μW**
Total Energy**	6.77 μJ	**6.62 μJ**	**6.55 μJ**
Partitioned Average Power for Divisor Doubling (μW)			
Part #1	12.54	12.67	12.44
Part #2	9.92	9.95	9.93
Part #3	16.71	**15.08**	**13.92**
Part #4	26.22	**20.11**	**17.52**
Part #5	12.67	12.65	12.48

* GE: Gate Equivalent – with 5.0922 μm^2 for NAND2X1 cell
** Values given for one complete divisor scalar multiplication

The partitioned power diagrams of the '*Original*' and '*Change* #2' designs are compared in Fig. 5. It clearly demonstrates a more uniform power diagram for the optimized '*Change* #2' design. In this figure, the dashed lines show the power consumption averaged over much shorter durations, which would resemble the instantaneous power consumption. For the case of the '*Change* #2' design, the dashed lines are almost uniform, which makes distinguishing between different parts of the algorithm more difficult.

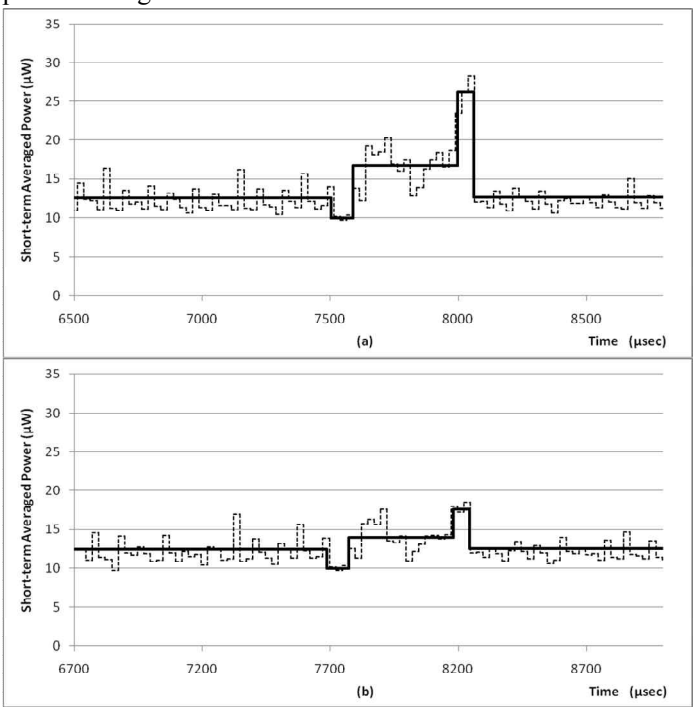

Fig. 5. Comparison of Short-term and Partitioned Power for (a) the 'Original' design and (b) the 'Change #2' design.

IX. Conclusion

In this work, we presented, for the first time, a prime-field energy-efficient hyperelliptic-curve cryptography (HECC) processor. Our HECC processor performed divisor scalar multiplication on the Jacobian of genus 2 hyperelliptic curves defined over prime fields. Only the most frequent cases of divisor addition and doubling were supported and the processor worked with any arbitrary field and curve parameter values. In terms of power and energy consumption, the HECC processor performed better than or almost equal to other similar designs. This shows that a prime-field HECC processor with flexibility in design parameters, can perform as good as binary-field designs. We also presented a technique to make the average

The article has been accepted for inclusion in a future issue of ETRI Journal, but has not been fully edited. Content may change prior to final publication.
http://dx.doi.org/10.4218/etrij.15.0114.0418

power consumption of the HECC processor more uniform and lower the peaks of its power consumption. This technique gave more security against simple power analysis attacks and eased the power supply requirements. The power and energy levels of the suggested processor made it appropriate for WSN and RFID-based systems.

References

[1] K. Finkenzeller, *RFID Handbook, 3rd ed.* John Wiley & Sons Inc., West Sussex, United Kingdom, 2010.

[2] D. Culler, D. Estrin, and M. Srivastava, "Overview of Sensor Networks," *Computer,* vol. 37, no. 8, pp. 41-49, Aug. 2004.

[3] W. Rankl and W. Effing, *Smart Card Handbook, 3rd ed.* John Wiley & Sons Inc., West Sussex, United Kingdom, 2003.

[4] A. Perrig et al., "SPINS: Security Protocols for Sensor Networks," *Wireless Networks,* vol. 8, issue 5, pp. 521-534, Sep. 2002.

[5] P.P. López, *"Lightweight Cryptography in Radio Frequency Identification (RFID) Systems,"* Ph.D. dissertation, Computer Science Department, Carlos III University of Madrid, Madrid, Spain, Oct. 2008.

[6] J.P. Kaps, G. Gaubatz, and B. Sunar, "Cryptography on a Speck of Dust," *Computer*, vol. 40, no. 2, pp. 38-44, Feb. 2007.

[7] M. Aigner, "Seven reasons for application of standardized crypto functionality on low cost tags," *in Proceedings of the 2007 EU RFID Forum*, Brussels, Belgium, March 2007.

[8] Y.K. Lee et al., "Elliptic-Curve-Based Security Processor for RFID," *IEEE Transactions on Computers,* vol. 57, no. 11, pp. 1514-1527, Nov. 2008.

[9] H.R. Ahmadi and A. Afzali-Kusha, "A Low-Power and Low-Energy Flexible GF(p) ECC Processor," *Journal of Zhejiang University - Science C,* vol. 11, no. 9, pp. 724-736, Sep 2010.

[10] K. Sakiyama, *"Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems,"* Ph.D. dissertation, Katholieke Universiteit Leuven, Leuven, Belgium, Dec. 2007.

[11] J. Fan, L. Batina, I. Verbauwhede, "Light-weight implementation options for curve-based cryptography: HECC is also ready for RFID," *in Proceedings of the International Conference for Internet Technology and Secured Transactions (ICITST) 2009,* London, UK, Nov. 2009.

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

[13] H. Cohen et al., *Handbook of Elliptic and Hyperelliptic Curve Cryptography,* Chapman and Hall/CRC, Florida, USA, 2006.

[14] P. Gaudry, *Hyperelliptic Curves and the HCDLP,* in I.F. Blake, G. Seroussi, and N.P. Smart, editors, *Advances in Elliptic Curve Cryptography,* Cambridge University Press, Cambridge, UK, 2005.

[15] N. Koblitz et al., *Algebraic Aspects of Cryptography,* Springer-Verlag, Berlin, Germany, 1998.

[16] T. Wollinger, J. Pelzl, and C. Paar, "Cantor versus Harley: Optimization and Analysis of Explicit Formulae for Hyperelliptic Curve Cryptosystems," *IEEE Transactions on Computers,* vol. 54, no. 7, pp. 861-872, Jul. 2005.

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

[18] T. Lange, and P.K. Mishra, "SCA Resistant Parallel Explicit Formula for Addition and Doubling of Divisors in the Jacobian of Hyperelliptic Curves of Genus 2," *in Proceedings of the 6th International Conference on Cryptology in India,* Bangalore, India, Dec. 2005.

[19] X. Fan, and G. Gong, "Efficient explicit formulae for genus 2 hyperelliptic curves over prime fields and their implementations," *in Proceedings of the 14th International Workshop on Selected Areas in Cryptography (SAC) 2007,* Ottawa, Canada, Aug. 2007.

[20] G. Elias, A. Miri, and T.H. Yeap, "FPGA Design of HECC Coprocessors," *in Proceedings of 2004 IEEE International Conference on Field-Programmable Technology (FPT) 2004,* Brisbane, Australia, Dec. 2004.

[21] J. Fan, L. Batina, and I. Verbauwhede, "HECC Goes Embedded: An Area-Efficient Implementation of HECC," *in Proceedings of the 15th International Workshop on Selected Areas in Cryptography (SAC) 2008,* New Brunswick, Canada, Aug. 2008.

[22] J. Pelzl et al., *"Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves,"* Cryptographic Hardware and Embedded Systems (CHES) 2003, in Lecture Notes in Computer Science, vol. 2779/2003, pp. 351-365, 2003.

[23] A. Hodjat et al., "A Hyperelliptic Curve Crypto Coprocessor for an 8051 Microcontroller," in Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS) 2005, Athens, Greece, Nov. 2005.

[24] H. Kim et al., "Hyperelliptic Curve Crypto-Coprocessor over Affine and Projective Coordinates," ETRI Journal, vol.30, no.3, pp.365-376, June 2008.

[25] K. Sakiyama et al., "Small-footprint ALU for public-key processors for pervasive security," *in Proceedings of the Workshop on RFID Security (RFIDSec) 2006,* Graz, Austria, Jul. 2006.

[26] L. Batina, K. Sakiyama, and I. Verbauwhede, *"Compact Public-Key Implementations for RFID and Sensor Nodes,"* Secure Integrated Circuits and Systems, in Series on Integrated Circuits and Systems, 2010(4), pp. 179-195, 2010.

[27] J. Fan, L. Batina, and I. Verbauwhede, "Light-weight Implementation Options for Curve-based Cryptography: HECC

http://dx.doi.org/10.4218/etrij.15.0114.0418

is Also Ready for RFID," in Proceedings of the 4th International Conference for Internet Technology and Secured Transactions, 2009. (ICITST 2009), London, UK, 2009, pp. 1-6.

[28] H.R. Ahmadi, A. Afzali-Kusha, and M. Pedram, "A Power-Optimized Low-Energy Elliptic-Curve Crypto-processor," *IEICE Electronics Express,* vol. 7, no. 23, pp. 1752-1759, Dec. 2010.

[29] P. Rong and M. Pedram, "An analytical model for predicting the remaining battery capacity of Lithium-ion batteries," *IEEE Transactions on VLSI systems,* vol. 14, no. 5, pp. 441-451, May 2006.

