INTEGRATION, the VLSI journal 44 (2011) 280–289

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Design and design methods for unified multiplier and inverter and its application for HECC

Junfeng Fan [a,]*, Lejla Batina [a,b], Ingrid Verbauwhede [a]

[a] Katholieke Universiteit Leuven, ESAT/SCD-COSIC and IBBT, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
[b] Radboud University Nijmegen, Institute for Computing and Information Sciences (ICIS), Postbus 9010, 6500 GL Nijmegen, The Netherlands

ARTICLE INFO

Available online 22 April 2011

Keywords:
Modular multiplication
Inversion
Hyperelliptic curve cryptography

ABSTRACT

This paper describes two novel architectures for a unified multiplier and inverter (UMI) in GF(2^m): the UMI merges multiplier and inverter into one unified data-path. As such, the area of the data-path is reduced. We present two options for hyperelliptic curve cryptography (HECC) using UMIs: an FPGA-based high-performance implementation (Type-I) and an ASIC-based lightweight implementation (Type-II). The use of a UMI combined with affine coordinates brings a smaller data-path, smaller memory and faster scalar multiplication.

Both implementations use curves defined by $h(x)=x$ and $f(x)=x^5+f_3x^3+x^2+f_0$. The high throughput version uses 2316 slices and 2016 bits of block RAM on a Xilinx Virtex-II FPGA, and finishes one scalar multiplication in 311 μs. The lightweight version uses only 14.5 kGates, and one scalar multiplication takes 450 ms.

1. Introduction

Hyperelliptic curve cryptography (HECC) [22] is an important candidate for public-key cryptography [13]. Like elliptic curve cryptography (ECC) [27,21], it uses smaller parameter sizes than RSA [29] for equivalent security level. In constrained devices, HECC enables valuable optimizations in area and speed.

Implementing HECC on a resource-constrained platform is a challenge for both area and performance. Table 1 describes the computational complexity of divisor operations in different coordinate systems [5]. Here I, M and S denote modular inversion, multiplication and squaring, respectively. Over the past few years, HECC have been implemented in both software [28,30,3,4,6] and hardware [8,11,17,14]. Table 2 summarizes previous FPGA-based HECC implementations.

In 2001, Wollinger described the first hardware architecture for HECC implementations [36]. However, the architecture was only outlined. The first complete hardware implementation of HECC was presented in [8]. This work was improved by Clancy [11]. All of them use Cantor's algorithm [10] for divisor addition and doubling.

In 2002, Lange generalized the explicit formulae for HECC over finite fields with arbitrary characteristic [23]. The first hardware implementation of HECC using explicit formulae was described in [15]. In [14] an improved version is proposed. The first hardware implementation using the affine version of explicit formulae was described in [35], which presents the fastest FPGA-based HECC coprocessor up to date.

Also some ASIC implementations of HECC using projective coordinates were proposed. For example, Sakiyama proposed an HECC coprocessor [31] using 130 nm CMOS technology. The coprocessor is able to run at 500 MHz, and it can perform one scalar multiplication of HECC over GF(2^{83}) in 63 μs.

Previous HECC implementations often use multiple multipliers or inverters to speed up the scalar multiplication. Commonly, an architecture shown in Fig. 1 is used. The use of multiple multipliers in parallel demands a high-throughput memory and a complex data bus, which result in further area increase. In this paper, we explore the power of a unified multiplier and inverter (UMI) for area reduction and performance improvement. We consider the architecture shown in Fig. 2 more area-efficient. We show that the use of a UMI brings three main advantages. First of all, the fast inverter makes affine coordinates very efficient, thus the performance is improved. Secondly, it reduces the area of the data-path. Thirdly, using only one data-path simplifies the data-bus and reduces the size of memory.

Our contributions: The contributions of this paper are threefold. Firstly, we propose two novel architectures for a unified

---

* Corresponding author.
E-mail addresses: fanjunfeng@gmail.com, junfeng.fan@esat.kuleuven.be (J. Fan), lejla.batina@esat.kuleuven.be (L. Batina), ingrid.verbauwhede@esat.kuleuven.be (I. Verbauwhede).

0167-9260/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2011.04.001

multiplier and inverter. We show that merging an inverter into a multiplier results in a substantial area reduction. Secondly, we propose a digit-serial UMI architecture and describe a method to adjust the digit-size. We use this method to explore the best area-time trade-off for HECC. Thirdly, using the proposed UMIs, we implement two HECC processors: a high-throughput design that outperforms all previous FPGA-based HECC implementations and a lightweight design that is suitable for passive RFID tags.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to the previous work, including mathematical background of HECC and the field arithmetic. Sections 4 and 5 describe the FPGA-based, high throughput architecture and the ASIC-based, lightweight architecture for HECC coprocessors, respectively. We conclude the paper in Section 6.

2. Previous work

2.1. Hyperelliptic curve cryptography

Hyperelliptic curves are a special class of algebraic curves; they can be viewed as a generalization of elliptic curves. Namely, a hyperelliptic curve of genus $g=1$ is an elliptic curve, while in general, hyperelliptic curves can be of any genus $g \geq 1$. However, not all geni are used for cryptography.

Let $\overline{GF(2^m)}$ be an algebraic closure of the field $GF(2^m)$. Here we consider a hyperelliptic curve C of genus $g=2$ over $GF(2^m)$, which is given by an equation of the form:

$$C: y^2+h(x)y=f(x) \quad \text{in } GF(2^m)[x,y], \tag{1}$$

where $h(x) \in GF(2^m)[x]$ is a polynomial of degree at most g ($\deg(h) \leq g$) and $f(x)$ is a monic polynomial of degree $2g+1$ ($\deg(f)=2g+1$). Also, there are no solutions $(x,y) \in \overline{GF(2^m)} \times \overline{GF(2^m)}$ which simultaneously satisfy Eq. (1) and the equations: $h(x)=0, h'(x)y+f'(x)=0$. For the genus 2, in the general case the following equation is used $y^2+(h_2x^2+h_1x+h_0)y=x^5+f_4x^4+f_3x^3+f_2x^2+f_1x+f_0$.

A divisor D is a formal sum of points on the hyperelliptic curve C, i.e., $D=\sum m_P P$, where P is a point on C, m_P is an integer and $m_P=0$ for almost all P. The degree of D is defined as $\deg D=\sum m_P$. Let Div denote the group of all divisors on C and Div_0 the subgroup of Div of all divisors with degree zero. The Jacobian J of the curve C is defined as quotient group $J=\text{Div}_0/R$, where R is the set of all principal divisors. A divisor D is called principal if $D=\text{div}(f)$ for some element f of the function field of C ($\text{div}(f)=\sum_{P \in C} ord_P(f)P$). The discrete logarithm problem in the Jacobian is the basis of security for HECC. In practice, the Mumford representation according to which each divisor is represented as a pair of polynomials $[u,v]$ is commonly used. Here, u is monic and $[u,v]$ satisfy $\deg(u) \leq 2, \deg(v) < \deg(u)$ and $u | f-hv-v^2$ (so-called reduced divisors).

The main operation in any hyperelliptic curve based primitive is scalar multiplication, i.e., mD where m is an integer and D is a reduced divisor in the Jacobian of C. The first algorithm for arithmetic in the Jacobian is due to Cantor [10]. However, until "explicit formulae" were introduced, HECC was not considered a suitable alternative to elliptic curve based cryptosystems. For geni 2 and 3, there was some substantial work on the formulae and algorithms for computing the group law on the Jacobian have been optimized. The algorithms we use for divisor operations are due to Lange and Stevens [25].

2.2. Field arithmetic

An element α in $GF(2^m)$ is represented as a polynomial $A(x)=\sum_{i=0}^{m-1} a_i x^i$, where $a_i \in GF(2)$. For the sake of simplicity, we use capital letters to denote polynomials, and small letters with subscript to denote their coefficients. For example, A stands for $A(x)$, and a_0 is the least significant bit of A.

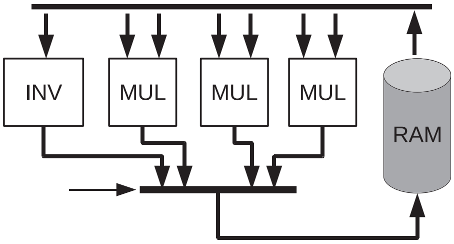

Fig. 1. Conventional architecture.

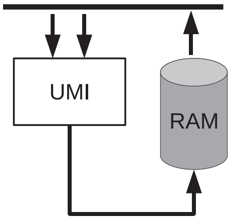

Fig. 2. Proposed architecture.

Table 1
Modular operations required by divisor operations.

	Coordinates	Divisor addition	Divisor doubling	Coordinates conversion
HECC	Affine	I+22M+3S	I+20M+6S	–
	Inversion-free	49M+4S	38M+7S	I+4M
	Lange–Stevens	I+21M+3S	I+5M+6S[a]	–

*This table is not exhaustive. State-of-the-art formulae can be found in [5,12].

[a] Note this fast doubling formulae only work for curves with $\deg(h)=1$.

Table 2
HECC implementations on FPGA.

Ref.	Year	Fields	Algorithm	Coordinates	Notes
[36]	2001	–	Cantor's	Affine	Architecture is only outlined
[8,11]	2002	$GF(2^{113})$			Two multipliers, one inverter, one ring GCD, one ring norm
[15]	2004	$GF(2^{113})$	Explicit formulae	Inversion-free	Twelve multipliers, one inverter
[35]	2004	$GF(2^{81})$		Affine	Three multipliers, two inverters
[32]	2006	$GF(2^{83})$		Projective	Three multipliers
[14]	2007	$GF(2^{113})$		Projective/mixed	Twelve multipliers, one inverter

Algorithm 1. MSB-first multiplication [7].

```
Input: Polynomial $A$, $B$ and $P$.
Output: $R = AB \bmod P$.
1: $C^m \leftarrow 0$;
2: for $i = m-1$ to 0 do
3:     $C^i \leftarrow x(C^{i+1} + b_i A) \bmod P$;
4: end for
Return: $R \leftarrow C^0/x$.
```

Algorithm 2. Left-shift inversion [20].

```
Input: Polynomial $A$ and $P$.
Output: $R = A^{-1} \bmod P$.
1:  $R^0 \leftarrow P$, $S^0 \leftarrow A$,
    $H^0 \leftarrow 0$, $J^0 \leftarrow x^{-m}$,
    $d^0 \leftarrow 0$;
2:  for $i = 1$ to $2m$ do
3:      $c \leftarrow (s_m^{i-1}) \& (d^{i-1} \geq 0)$;
4:      if $c = 1$ then
5:          $\{R^i, H^i\} \leftarrow \{S^{i-1}, J^{i-1}\}$;
6:      else
7:          $\{R^i, H^i\} \leftarrow \{R^{i-1}, H^{i-1}\}$;
8:      end if
9:      if $c = 1$ then
10:         $S^i \leftarrow x(R^{i-1} + S^{i-1})$;
            $J^i \leftarrow x(H^{i-1} + J^{i-1}) \bmod P$ ;
            $d^i \leftarrow -d^{i-1} + 1$;
11:     else
12:         $S^i \leftarrow x(s_m^{i-1} R^{i-1} + S^{i-1})$;
            $J^i \leftarrow x(s_m^{i-1} H^{i-1} + J^{i-1}) \bmod P$;
            $d^i \leftarrow d^{i-1} + 1$;
13:     end if
14: end for
Return: $R \leftarrow H^{2m}$.
```

Algorithm 3. LSB-first multiplication [7].

```
Input: Polynomial $A$, $B$ and $P$.
Output: $R = AB \bmod P$.
1:  $C^0 \leftarrow 0$, $T^0 \leftarrow A$;
2:  for $i = 0$ to $m-1$ do
3:      $C^{i+1} \leftarrow C^i + b_i T^i$;
4:      $T^{i+1} \leftarrow xT^i \bmod P$;
    end for
Return: $R \leftarrow C^m$.
```

Algorithm 4. Right-shift inversion [37].

```
Input: Polynomial $A$ and $P$.
Output: $R = A^{-1} \bmod P$.
1:  $R^0 \leftarrow P$, $S^0 \leftarrow xA$,
    $H^0 \leftarrow 0$, $J^0 \leftarrow x^m$,
    $d^0 \leftarrow 2$, $sign^0 \leftarrow 1$;
2:  for $i = 1$ to $2m-1$ do
3:      $c_1 \leftarrow s_m^{i-1}$;
        $c_2 \leftarrow c_1 \& sign^{i-1}$;
        $sign^i \leftarrow sign^{i-1} ? \bar{c}_1 : d_0^{i-1}$ ;
4:      if $c_2 = 1$ then
5:          $\{R^i, H^i\} \leftarrow \{S^{i-1}, J^{i-1}/x\}$;
6:      else
7:          $\{R^i, H^i\} \leftarrow \{R^{i-1}, H^{i-1}/x\}$;
8:      end if
9:      if $c_1 = 1$ then
10:         $S^i \leftarrow x(R^{i-1} + S^{i-1})$;
            $J^i \leftarrow H^{i-1} + J^{i-1}$;
11:     else
12:         $S^i \leftarrow xS^{i-1}$;
            $J^i \leftarrow J^{i-1}$;
13:     end if
        $d^i \leftarrow sign^i ? 2d^{i-1} : d^{i-1}/2$;
14: end for
Return: $R \leftarrow H^{2m-1}$.
```

2.2.1. Multiplication

In the literature there are various algorithms and architectures proposed for modular multiplication in GF(2^m) [7,34]. The bit-serial algorithms can be classified into two categories, the most significant bit (MSB) first algorithms and the least significant bit (LSB) first algorithms. Algorithms 2 and 4 show an MSB-first and an LSB-first multiplication algorithm, respectively. Here we use C^i to denote the value of C after ith iteration, and b_i the ith coefficient of B.

The MSB-first multiplication scans B from the MSB side. In each iteration, $b_i A$ is added to C, which is then shifted to the left and reduced. The LSB-first multiplication scans B from the LSB side. In each iteration, T is updated to xT, and b_{iT} accumulated in C. LSB-first multipliers update T and C in parallel, thus they can achieve shorter critical path than MSB-first multipliers [7]. On the other hand, it requires an extra register to keep T.

2.2.2. Inversion

Modular inversion is considered as a computationally expensive operation. The most commonly used inversion algorithms are based on Fermat's little theorem [2], extended Euclidean algorithm (EEA) [20] and Gaussian elimination [18]. EEA is widely used to perform inversion in practice.

The schoolbook EEA-based inversion algorithm in GF(2^m) is considered inefficient due to the long polynomial division in each iteration. This problem was partially solved by replacing the degree comparison with a counter [9]. Algorithms 2 and 4 show two variants of EEA, namely, left-shift inversion and right-shift inversion [37], respectively. Here we use S^i to denote the value of S after ith iteration, and s_m^{i-1} the MSB of S^{i-1}. The complement of c_1 is represented as $\bar{c}_1$.

From an implementation perspective, the right-shift inversion algorithm is preferred for a high-performance inverter. The right-shift inversion algorithm has no modular operations. As a result, a short critical path delay can be easily achieved. The counter d is realized with the ring counter [37]. A ring counter d has only one 1-bit. The value of the counter is defined as $(-1)^{sign} \cdot \delta$, where δ is the number of 0 at the right side of 1 in the register d. An n-bit ring counter can count up to $n-1$, thus it is larger than an equivalent counter using ripple-carry adder. On the other hand, it has a shorter critical path delay since it only has shift operations. The left-shift inversion algorithm uses a ripple-carry adder, and it fits better in area-constrained devices.

3. Unified multiplier and inverter

The main observation of this paper is that multiplier and inverter can be efficiently merged, which brings a significant reduction in area. For example, Step 3 in Algorithm 3 and Step 10 in Algorithm 4 can be generalized to the following operation:

$$T \leftarrow x(G + eQ).$$

Table 3
Unified multiplier and inverter: Type-I vs. Type-II.

	Optimization priority	Algorithm selection	Counter d	I/M	Target applications
Type-I	Short critical path delay	Alg. 3 + Alg. 4	Ring	2	High throughput
Type-II	Low footprint	Alg. 1 + Alg. 2	Carry-ripple	4	Lightweight

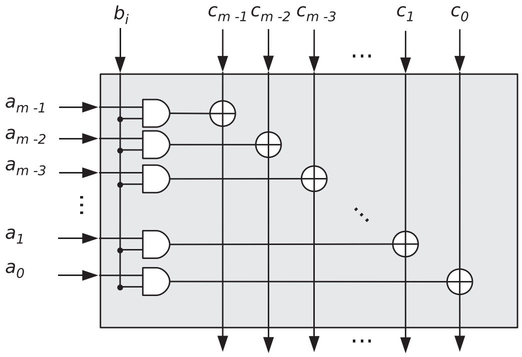

Fig. 3. AND–XOR cell building block.

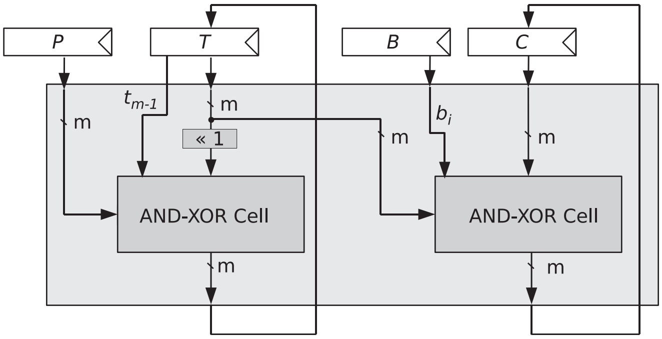

Fig. 4. LSB-first bit-serial modular multiplier.

Another example is Algorithm 1 and Step 12 in Algorithm 2. They can be generalized to

$T \leftarrow x(G+eQ) \bmod P.$

Indeed, a modification of the architecture of a bit-serial multiplier makes it also an inverter.

In the following sections we describe two UMI architectures. Table 3 summarizes the design rationale of these two types of UMI. Let I/M denote the inversion to multiplication ratio in terms of delay. Type-I UMI is optimized for low critical path delay. It realizes the LSB-first multiplication and the Right-shift EEA algorithm. Here one inversion is equivalent to 2 multiplications. Type-I UMI is used to build a high-performance HECC processor. Type-II UMI is targeting ultra-constrained devices. It realizes the MSB-first multiplication and the Left-shift EEA algorithm. Here one inversion is equivalent to 4 multiplications. Type-II UMI is used to build a low footprint HECC processor.

4. High-throughput UMI and HECC processor

In this section we present the architecture of Type-I UMI and a high-performance HECC processor. We first describe the architecture of the UMI, then we discuss a method to select the I/M ratio. We also compare the performance of the design with previous implementations in the end of this section.

4.1. Type-I UMI architecture

Fig. 3 describes the AND–XOR cell that realizes (b_iA+C). Fig. 4 shows the architecture of an LSB-first multiplier. Here $(t_{m-1}P+(T \ll 1))$ and (b_iT+C) are performed on the left and the right cell, respectively. The critical path delay is $T_{\mathrm{AND}}+T_{\mathrm{XOR}}$, where T_{AND} and T_{XOR} denote the delay of a 2-input AND and XOR gate, respectively. B is shifted to the right by one bit in each clock cycle. Hence one multiplication in GF(2^m) takes m clock cycles on this multiplier.

Fig. 5 shows the data-path of a bit-serial inverter using the AND–XOR cells. It realizes Algorithm 4. The critical path, from $sign^{i-1}$ to d^i, has a delay of $2T_{\mathrm{MUX}}$, where T_{MUX} denotes the delay of a 2-input multiplexer.

Fig. 6 shows the data-path of the proposed unified inverter and multiplier. The data-path realizes both Algorithms 3 and 4. Table 4 describes how to configure the UMI to perform inversion or multiplication.

The goal of this data-path merging is to maximize the hardware sharing and at the same time to minimize the overhead on critical path delay.

- *Hardware sharing*: Three registers (R, S and H) and one AND–XOR cell are shared.
- *Critical path*: The critical path delay is the same as a standalone inverter, i.e., $2T_{\mathrm{MUX}}$.
- *Function selection*: The selection of a working mode (i.e., multiplication or inversion) is performed on the existing registers at the first cycle. It is also shown in Table 4.
- *Throughput*: The UMI achieves a throughput of $1/(2m-1)$ inversions or $1/m$ multiplications per cycle.

The critical path delay of UMI is longer than the one of a multiplier. In other words, merging an inverter into a multiplier slows down the multiplication. However, for divisor additions in HECC, performing one inversion saves 28 multiplications (see Table 1). Indeed, having a fast inverter at the cost of slower multiplication may still speed up the divisor addition and doubling. This issue is discussed in the following section.

4.1.1. Digit-serial UMI

While the use of UMI achieves an area reduction of the ALU, it also slows down multiplications. For applications where many more multiplications than inversions are required, maximizing the throughput of an inverter at the cost of a slower multiplier is not always desirable. Therefore, we propose a flexible architecture which enables an arbitrary I/M ratio. Fig. 7 shows a design that replaces two bit-serial UMI with multipliers. We use w_I and w_M to denote the actual digit-size of the inverter and multiplier, respectively. The UMI in Fig. 7 uses two UMIs ($w_I=2$) and two multipliers ($w_M=4$). When $m=83$, one inversion takes $\lceil 2m-1/w_I \rceil = 83$ clock cycles, while one multiplication takes $\lceil m/w_M \rceil = 21$ clock cycles. The I/M ratio is approximately $2w_M/w_I$.

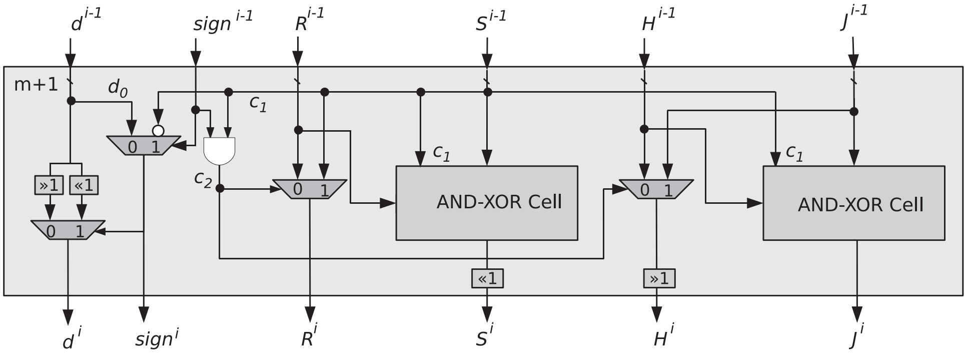

Fig. 5. Right-shift bit-serial inverter.

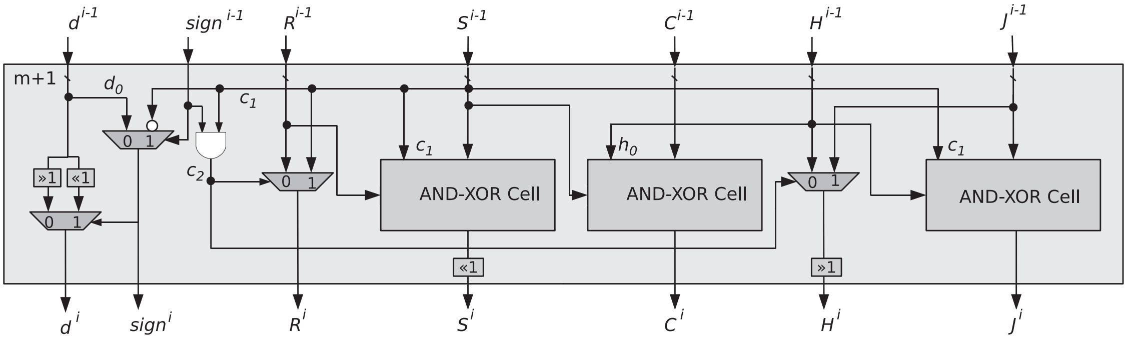

Fig. 6. Type-I bit-serial UMI.

Table 4
Configurations and operations of UMI-I.

Registers	Multiplication		Inversion		
	$i=0$	$0<i<m+1$	$i=0$	$0<i<2\,m$	
d	0	–	2	$d^{i-1} \ll 1$	if $sign^i = 1$
				$d^{i-1} \gg 1$	if $sign^i = 0$
$sign$	0	–	1	$\neg s_m^{i-1}$	if $sign^{i-1} = 1$
				d_0^{i-1}	if $sign^{i-1} = 0$
R	P	R^{i-1}	P	S^{i-1}	if $(s_m^{i-1}\ \&\ sign^{i-1}) = 1$
				R^{i-1}	if $(s_m^{i-1}\ \&\ sign^{i-1}) = 0$
S	xA	$(S^{i-1}+s_m^{i-1}R^{i-1}) \ll 1$	xA	$(S^{i-1}+s_m^{i-1}R^{i-1}) \ll 1$	
C	0	$h_0(S^{i-1} \gg 1)+C^{i-1}$	–	–	
H	B	$H^{i-1} \gg 1$	0	$J^{i-1} \gg 1$	if $(s_m^{i-1}\ \&\ sign^{i-1}) = 1$
				$H^{i-1} \gg 1$	if $(s_m^{i-1}\ \&\ sign^{i-1}) = 0$
J	–	–	x^m	$J^{i-1}+s_m^{i-1}H^{i-1}$	
Return	C^m		H^{2m-1}		

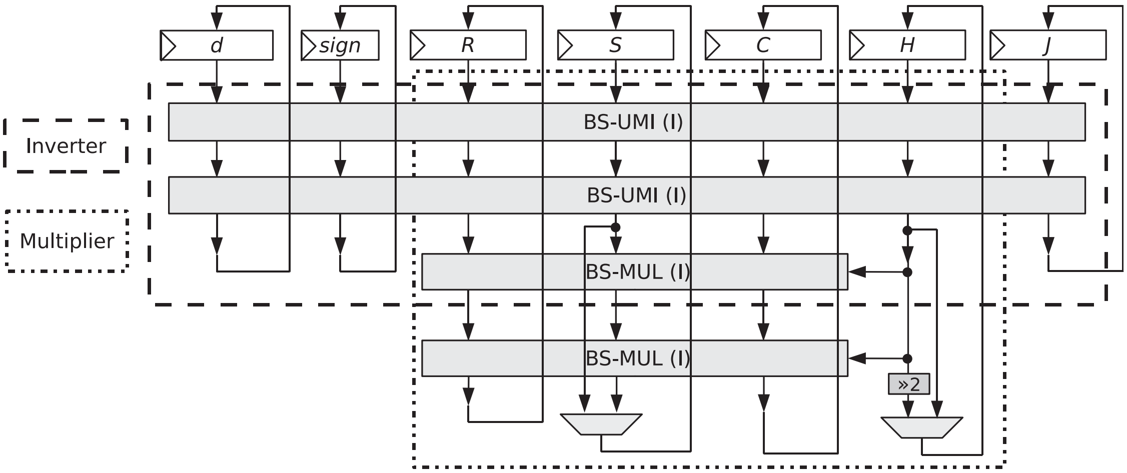

Fig. 7. Type-I digit-serial UMI with I/M $\approx 2w_M/w_I$ ($w_I=2, w_M=4$).

The w_I/w_M ratio should be decided by the applications and the design constraints of the circuit. The next section describes an HECC processor built with the UMI.

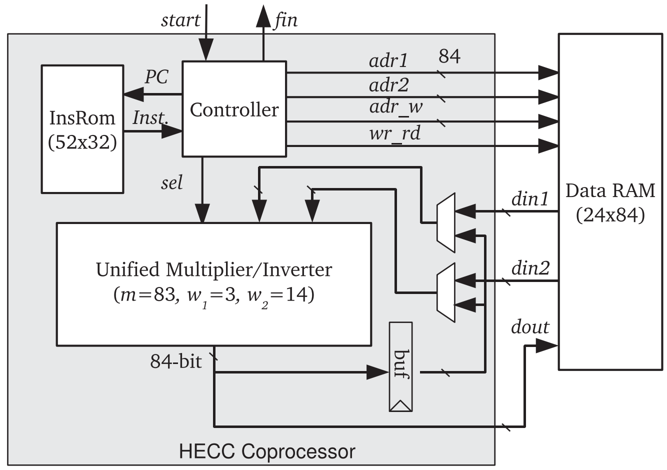

Fig. 8. Block diagram of the Type-I HECC processor.

4.2. Type-I HECC processor

The HECC coprocessor is shown in Fig. 8. It contains an instruction ROM, a main controller and the Type-I UMI. The instruction ROM contains the field operation sequences of divisor addition and doubling. As only a single data-path is used, the coprocessor does not require high-bandwidth register files. Instead, a data RAM is used to keep the curve parameters, base divisor and intermediate data. On FPGAs, block RAMs are used.

For divisor addition and doubling, we use the explicit formulae proposed by Lange and Stevens [25]. One divisor addition takes 1I+21M+3S, while one divisor doubling takes 1I+5M+6S. We give in the Appendix the explicit formulae in the form of register operations.

The selection of w_M and w_I is decided by the following constraints: speed and area. We choose w_M=14 such that the area meets our constraints, i.e., the overall area should be smaller than the smallest known implementation ([32] in Table 2). We then adjust w_I and measure the performance of the UMI on a Xilinx XC2V4000 FPGA. The following equations are used to estimate the delay of one divisor addition (DA), one divisor doubling (DD) and one scalar multiplication

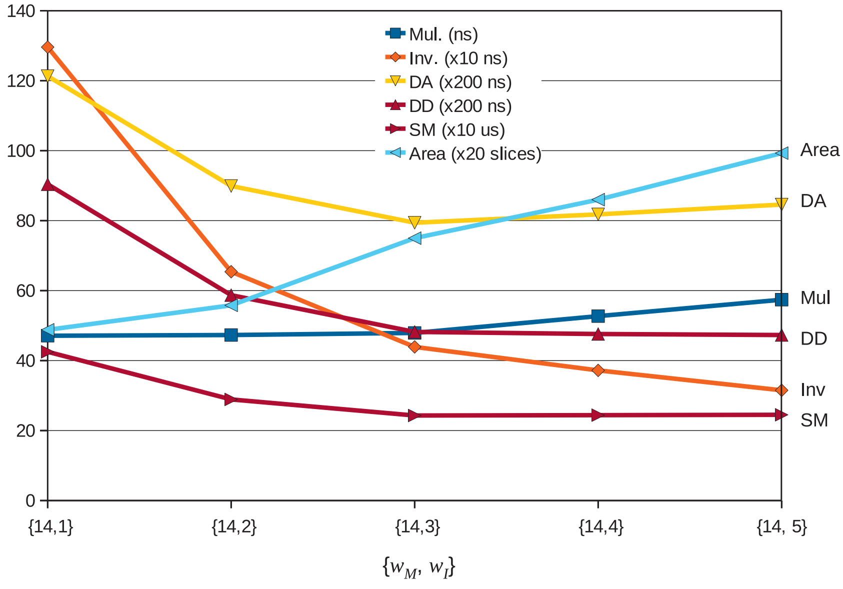

Fig. 9. Area of the UMI and delay for DA, DD and SM.

Table 5
Performance comparison of FPGA-based HECC implementations in GF(2^m).

Ref. Design	FPGA	Freq. (MHz)	Area (Slices)	RAM (bits)	Finite field	Perf. (μs)	Comments
Clancy [11]	Xilinx Virtex-II	N/A	23,000	0	GF(2^{83})*	10,000	Two multipliers, one inverter, using NAF[a]
Elias et al. [14]	Xilinx Virtex-II (XC2V8000)	45.3	25,271	0	GF(2^{113})	2030	Twelve multipliers, one inverter, using NAF
Sakiyama et al. [32]	Xilinx Virtex-II Pro (XC2VP30)	100	6586	8064	GF(2^{83})*	420	Three multipliers, using NAF
		100	4749	5376	GF(2^{83})*	549	Two multipliers, using NAF
			2446	2688	GF(2^{83})*	989	One multiplier, using NAF
Wollinger [35]	Xilinx Virtex-II (XC2V4000)	56.7	7785	0	GF(2^{81})	415[b]	Three multipliers, two inverter
		47.0	5604	0	GF(2^{81})	724[b]	Two multipliers, one inverter
		54.0	3955	1536	GF(2^{81})	831[b]	Two multipliers, one inverter
This work	Xilinx Virtex-II (XC2V4000)	125	2316	2016	GF(2^{83})	311	Type-I UMI, using NAF

Designs with * support fields defined with an arbitrary polynomial P.

[a] Non-adjacent form.

[b] Using binary method for scalar divisor multiplication.

(SM), respectively. Here T_I and T_M denote the delay of one inversion and multiplication, respectively. Note that squaring is also performed with the UMI, thus $T_S = T_M$.

$$T_{DA} = T_I + 24T_M, \quad T_{DD} = T_I + 11T_M, \quad T_{SM} = 166T_{DD} + 83T_{DA}.$$

As shown in Fig. 9, when w_I increases, T_M goes up. However, the delay of one inversion goes down. T_{DA} reaches its low point when $w_I=3$, while T_{DD} stays almost unchanged when w_I goes from 3 to 5. The delay of one scalar multiplication also reaches its low point at $w_I=3$. Note that the area increases almost linearly when w_I grows. When $w_I > 3$, there is no gain in speed while area goes up. Thus, we choose $w_I=3$ and $w_M=14$ as the best performance-area trade-off for this architecture. One multiplication and one inversion in GF(2^{83}) take 47.9 and 439 ns, respectively.

4.2.1. Results and comparison

We implemented the architecture from Fig. 8 on a Xilinx Virtex-II (XC2V4000) FPGA. The coprocessor is described with the Gezel [33] language and synthesized with Xilinx ISE8.1. It uses 2316 slices and 6 block RAMs. A clock frequency of 125 MHz can be reached. Table 5 gives a comparison in the area and performance with previous FPGA-based implementations of HECC in GF(2^m).

Among all the previous implementations, the design from Sakiyama et al. and Wollinger are of special interests to compare with. They both use explicit formulae, and the designs are much smaller than other implementations. The HECC coprocessor presented in [32] uses projective coordinates and a superscalar architecture. Different number of digit-serial ($w=12$) multipliers are used for different configurations. Our coprocessor, using one unified multiplier/inverter, is faster than the one of [32] using three multipliers.

The architectures proposed in [35], however, uses affine coordinates of the explicit formulae. Three different architectures ranging from high speed to low hardware cost are proposed. The high speed version uses three multipliers and two inverters, and it takes 415 µs to finish one scalar multiplication. To the best of our knowledge, this is also the fastest HECC implementation on FPGA up to date. The low-area version uses 3955 slices. However, it requires 831 µs for one scalar multiplication.

Compared to all the previous implementations, our HECC processor achieves a higher performance at a lower area cost. The area reduction is attributed to the use of compact ALU and the reduction of the memory throughput. The ALU in [35] contains two multipliers and one inverter, which in total use 2427 slices. The ALU used in this paper requires only 1500 slices. The performance gain is mainly due to the efficient inverter. When running at 56.7 MHz, the inverter in [35] requires 1570 ns on average for one inversion in GF(2^{81}), while our high-throughput UMI finishes one inversion in GF(2^{83}) in 439 ns. Although we use only one multiplier, which is also slower than the one in [35], the divisor addition and doubling are faster.

5. Lightweight UMI and HECC processor for RFID

In this section, we describe an HECC processor targeting extremely constrained devices such as passive RFID tags. In such applications, area and power consumption are of higher priority than performance. According to [1], a passive RFID tag should have power consumption less than 15 µW to guarantee 1 m operation range. Some ECC implementations [26,19] can already fulfill the requirements. We show that HECC can also fulfill the requirements with a comparable performance.

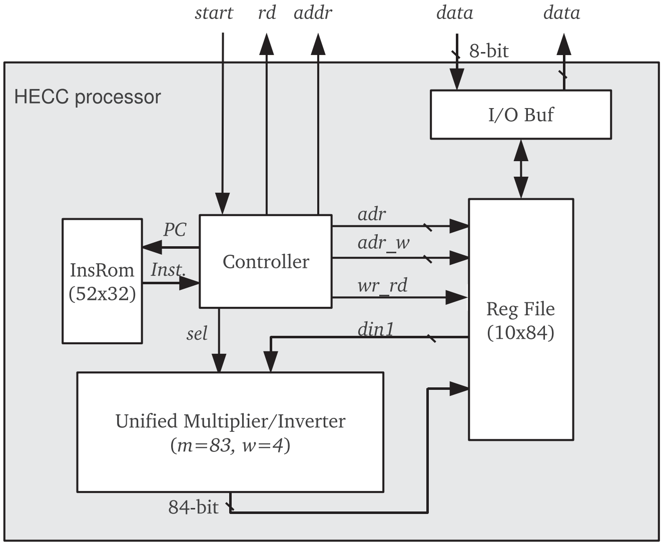

Fig. 11. Block diagram of the Type-II HECC processor.

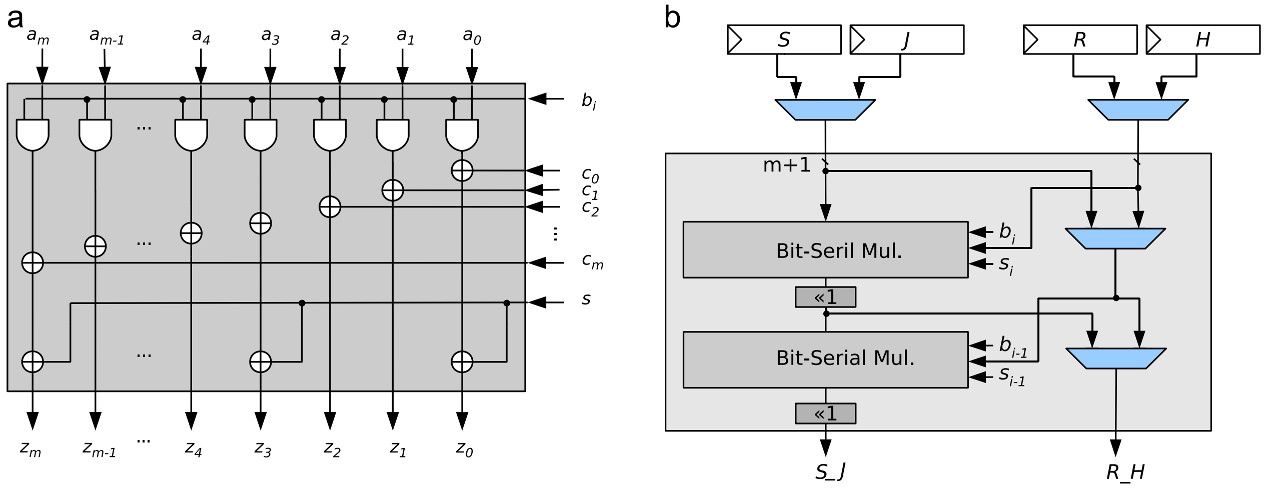

Fig. 10. Type-II UMI. (a) Bit-serial modular multiplier building block and (b) Digit-serial UMI ($w=2$).

Table 6
Performance comparison of HECC and ECC implementations targeting RFID tags.

Ref. Design	ASIC Tech.	Finite field	Area (kGates)	Perf. (#cycle)	Freq. (kHz)	Power (μW)	Energy[a] (μJ)	Comments
HECC (this work)	130 nm	$GF(2^{83})$	14.5	136,838	300	13.4	6.03	Type-II UMI ($d=4$)
HECC (Sakiyama [31])	130 nm	$GF(2^{67})$	7.6[b]	266,133	500	19[b]	10.0[b]	One multiplier ($d=8$)
ECC (Lee et al. [26])	130 nm	$GF(2^{163})$	14.1[c]	144,842	590	21.55	5.29	($d=2$)
			14.7[c]	101,183	411	15.75	3.88	($d=3$)
			15.4[c]	78,544	323	12.08	2.94	($d=4$)
ECC (Hein et al. [19])	180 nm	$GF(2^{163})$	11.9	296,000	106	10.8	31.3[d]	16 × 16 multipliers

[a] Energy for one scalar multiplication.
[b] Modular arithmetic logic unit only.
[c] Including ECC core and an 8-bit controller for cryptographic protocols.
[d] Estimated by authors.

5.1. Type-II UMI

Type-II UMI realizes Algorithms 1 and 2. In this architecture, the bit-serial multiplier is reused for the inversion. The counter d, implemented as a ring counter in Type-I UMI, is replaced with a ripple-carry adder. Fig. 10 shows the data-path of the proposed digit-serial inverter and multiplier.

- *Multiplication*: The data-path performs $C^{i+1} \leftarrow ((C^i + b_i A) \ll 1) \bmod P$. In this case, only two registers (S and H) are used, thus R and J can be used as storage.
- *Inversion*: In this mode, $\{R,S\}$ pair and $\{H,S\}$ pair are updated alternatively. The bit-serial multiplier performs one of the following operations:
    - ○ $S^i \leftarrow (R^{i-1} + S^{i-1}) \ll 1$
    - ○ $S^i \leftarrow (s_m^{i-1} R^{i-1} + S^{i-1}) \ll 1$
    - ○ $J^i \leftarrow ((H^{i-1} + J^{i-1}) \ll 1) \bmod P$
    - ○ $J^i \leftarrow ((s_m^{i-1} H^{i-1} + J^{i-1}) \ll 1) \bmod P$.

  Note that R and S are updated first, then H and J are updated accordingly in the next cycle.

Assuming the digit-size is w, one multiplication in $GF(2^m)$ takes $\lceil m/w \rceil$ cycles, while one inversion takes $\lceil 4m/w \rceil$ cycles.

5.2. Type-II HECC processor: low-footprint

The HECC processor is shown in Fig. 11. It contains an instruction ROM, a main controller, a Type-II UMI, a register file, and an input/output interface. It differs from the Type-I HECC processor in both the UMI architecture and storage. The Type-I HECC processor uses a dual-port RAM, while the Type-II HECC processor uses a single-port register file.

Besides the multiplier and inverter, the register file makes a big portion of the area. Reducing area of the register file is the key step towards a compact implementation. An HECC processor using affine coordinates requires fewer registers to store intermediate results since no Z coordinates are used. Moreover, it also reduces the number of intermediate results. Lange and Mishra studied the register allocation for parallel multipliers [24]. Our investigation shows that 12 registers are sufficient for scalar multiplication with flexible base divisor D. Note that the Type-II UMI has four registers, among which two can be used for storage when it is not working as an inverter. Thus, we only need 10 registers in the register file. The complete register allocation for divisor doubling and divisor addition is given in the Appendix.

5.3. Results and comparison

We synthesized the Type-II HECC processor with 130 nm standard cell library. Table 6 summarizes the area and power of the proposed design.

Our HECC implementation uses 14.5 kGates and finishes one scalar multiplication in 136,838 clock cycles. The power consumption, estimated with power compiler, is 13.4 μW when running at 300 kHz. The implementation of [31], using projective coordinates, requires 266,133 clock cycles for one scalar multiplication. Note that it is defined on a smaller field and the result does not include data storage. The power and energy consumption of our design is 65% lower while it achieves the same throughput.

There are several ECC implementations proposed for RFID tags. Lee et al. [26] use digit-serial multipliers, while Hein et al. use a 16 × 16 GF(2) multiplier and 32-bit accumulator. Comparing the implementation in [26], using a 16 × 16-bit multiplier requires less area, lower power consumption. On the other hand, it requires 296k clock cycles, twice as many as Lee's ECC processor (and our HECC processor), for one scalar multiplication, and its energy consumption is about six times higher.

Our HECC processor can meet the requirements for passive RFID tags in terms of area, power and energy. However, ECC implementations are still leading in terms of the energy efficiency. This is due to the fact that the computational complexity of HECC scalar multiplication is higher than ECC.

6. Conclusions

We explore the efficiency of a unified multiplier and inverter datapath in HECC implementations. Two types of UMI are proposed. Type-I UMI, which realizes the LSB-first multiplication and right-shift EEA algorithms, achieves a short critical path delay. Using the Type-I UMI results in a high performance HECC processor on FPGA. The Type-II UMI, which realizes the MSB-first multiplication and the left-shift EEA algorithms, achieves a low footprint. Using the Type-II UMI results in a lightweight HECC processor for constrained devices. The use of UMI brings a substantial improvement in terms of area and performance of HECC implementations.

For applications like RFIDs, physical security is very important. Known HECC implementations are either based on a binary scalar multiplication method or NAF. They might be vulnerable to side-channel attacks. In ECC implementations, Montgomery ladder is widely used for protection against simple power analysis. A recent work by Gaudry and Lubicz [16] has shown that scalar multiplication of divisors can also use the Montgomery ladder. As a future work, we will combine our architecture with the algorithm proposed by Gaudry and Lubicz.

Acknowledgments

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by FWO project G.0300.07, by the Research Council K.U. Leuven: GOA TENSE (GOA/11/007), and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

Appendix A. Register allocation for divisor doubling and addition

Register allocation for divisor doubling and divisor addition is given in Tables A1 and A2.

Table A1
Divisor doubling.

Input: {R4,R5,R6,R7}= D1(={u10,u11,v10,v11}).

1. R3:=R4*R4;
2. R4:=R5*R5+f3;
3. R6:=R6*R6+f0;
4. R6:=1/R6;
5. R6:=R6*R3;
6. R2:=R4*R6;
7. R0:=R2+R5;
8. R5:=R6*R6;
9. R1:=R6+R4 ;
10. R4:=R0*R0+R6;
11. R2:=R1*R2+f2;
12. R2:=R6*R5+R2;
13. R7:=R7*R7+R2;
14. R6:=R1*R4+R3;

Return: {R4,R5,R6,R7}=2*D1.

Table A2
Divisor addition.

Input: {R4,R5,R6,R7}=D1(={u10,u11,v10,v11}),
{R8,R9,R10,R11}=D0(={u00,u01,v00,v01}).

1. R0:=R5+R9;
2. R1:=R0*R0;
3. R1:=R1*R4;
4. R2:=R5*R0;
5. R3:=R8+R4;
6. R2:=R2+R3;
7. R3:=R3*R2+R1;
8. R6:=R6+R10;
9. R1:=R2*R6;
10. R7:=R7+R11;
11. R6:=R7+R6;
12. R0:=R5+R9;
13. R7:=R0*R7;
14. R2:=R2+R0;
15. R6:=R2*R6+R1;
16. R6:=R7*R5+R6;
17. R6:=R7+R6;
18. R7:=R4*R7+R1;
19. R2:=R3*R6;
20. R2:=1/R2;
21. R6:=R6*R6;
22. R6:=R6*R2;
23. R2:=R3*R2;
24. R3:=R3*R2;
25. R4:=R4+R3;
26. R7:=R7*R2;
27. R0:=R9+R5;
28. R5:=R7+R5;
29. R7:=R7+R0;
30. R4:=R5*R7+R4;
31. R7:=R7+R0;
32. R1:=R9*R7+R8;
33. R4:=R4+R1;
34. R5:=R3*R3;
35. R3:=R8*R7;
36. R4:=R0*R5+R4;
37. R5:=R0+R5;
38. R7:=R9+R7;
39. R7:=R7+R5;
40. R0:=R5*R7+R4;
41. R0:=R0+R1;
42. R7:=R4*R7+R3;
43. R0:=R0*R6+R11;
44. R6:=R7*R6+R10;
45. R7:=R0+1;

Return: {R4,R5,R6,R7}=D1+D0.

References

[1] ISO/IEC 18000-1:2004, Information technology—radio frequency identification for item management. Part 3: parameters for air interface communications at 13.56 MHz.
[2] Y. Asano, T. Itoh, S. Tsujii, Generalised fast algorithm for computing multiplicative inverses in GF(2^m), Electronics Letters 25 (10) (1989) 664–665.
[3] R. Avanzi, Aspects of hyperelliptic curves over large prime fields in software implementations, CHES, Lecture Notes in Computer Science, vol. 3156, Springer, 2004, pp. 148–162.
[4] R. Avanzi, N. Thériault, Z. Wang, Rethinking low genus hyperelliptic Jacobian arithmetic over binary fields: interplay of field arithmetic and explicit formula, Journal of Mathematical Cryptology 2 (2008) 227–255.
[5] R.M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, F. Vercauteren, Handbook of Elliptic and Hyperelliptic Curve Cryptography, CRC Press, 2005.
[6] D.J. Bernstein, T. Lange, eBACS: ECRYPT benchmarking of cryptographic systems, 2010 ⟨http://bench.cr.yp.to/⟩.
[7] T. Beth, D. Gollman, Algorithm engineering for public key algorithms, IEEE Journal on Selected Areas in Communications 7 (4) (1989) 458–466.
[8] N. Boston, T. Clancy, Y. Liow, J. Webster, Genus two hyperelliptic curve coprocessor, CHES 2002, Lecture Notes in Computer Science, vol. 2523, Springer, 2003, pp. 400–414.
[9] R.P. Brent, H.T. Kung, Systolic VLSI arrays for polynomial GCD computation, IEEE Transactions on Computers 33 (8) (1984) 731–736.
[10] D.G. Cantor, Computing in the Jacobian of a hyperelliptic curve, Mathematics of Computation 48 (1987) 95–101.
[11] T. Clancy, FPGA-based hyperelliptic curve cryptosystems, Invited paper presented at AMS Central Section Meeting, April 2003.
[12] Explicit-formulas database ⟨http://www.hyperelliptic.org/EFD⟩.
[13] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976) 644–654.
[14] G. Elias, A. Miri, T.H. Yeap, On efficient implementation of FPGA-based hyperelliptic curve cryptosystems, Computers and Electrical Engineering 33 (5–6) (2007) 349–366.
[15] G. Elias, A. Miri, T.H. Yeap, High-performance FPGA-based hyperelliptic curve cryptosystems, in: The Proceeding of the 22nd Biennial Symposium on Communications, May 2004.
[16] P. Gaudry, D. Lubicz, The arithmetic of characteristic 2 Kummer surfaces and of elliptic Kummer lines, Finite Fields and Their Applications 15 (2) (2009) 246–260.
[17] H. Kim, T.J. Wollinger, Y. Choi, K. Chung, C. Paar, Hyperelliptic curve coprocessors on a FPGA, WISA 2004, Lecture Notes in Computer Science, vol. 3325, Springer, 2004, pp. 360–374.
[18] M.A. Hasan, V.K. Bhargava, Bit-serial systolic divider and multiplier for finite fields GF(2^m), IEEE Transactions on Computers 41 (8) (1992) 972–980.
[19] D. Hein, J. Wolkerstorfer, N. Felber, ECC is ready for RFID a proof in silicon, SAC 2008, Lecture Notes in Computer Science, vol. 5381, Springer, 2008, pp. 401–413.
[20] D.E. Knuth, The Art of Computer Programming, vol. 2, Addison-Wesley, 1981.
[21] N. Koblitz, Elliptic curve cryptosystem, Mathematics of Computation 48 (1987) 203–209.
[22] N. Koblitz, Hyperelliptic cryptosystems, Journal of Cryptology 1 (3) (1989) 129–150.
[23] T. Lange, Formulae for arithmetic on genus 2 hyperelliptic curves, Journal of AAECC 15 (5) (2005) 295–328.
[24] T. Lange, P.K. Mishra, SCA resistant parallel explicit formula for addition and doubling of divisors in the Jacobian of hyperelliptic curves of genus 2, INDOCRYPT, Lecture Notes in Computer Science, vol. 3797, 2005, pp. 403–416.
[25] T. Lange, M. Stevens, Efficient doubling on genus two curves over binary fields, SAC 2004, Lecture Notes in Computer Science, vol. 3357, Springer, 2004, pp. 170–181.
[26] Y.K. Lee, K. Sakiyama, L. Batina, I. Verbauwhede, Elliptic-curve-based security processor for RFID, IEEE Transactions on Computers 57 (11) (2008) 1514–1527.
[27] V. Miller, Uses of elliptic curves in cryptography, in: H.C. Williams (Ed.), CRYPTO'85, Lecture Notes in Computer Science, vol. 218, Springer-Verlag, 1985, pp. 417–426.
[28] J. Pelzl, Hyperelliptic cryptosystems on embedded microprocessors, Master's Thesis, Ruhr-Universitat Bochum, September 2002.
[29] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
[30] Y. Sakai, K. Sakurai, Design of hyperelliptic cryptosystems in small characteristic and a software implementation over F_{2^n}, ASIACRYPT, Lecture Notes in Computer Science, vol. 1514, Springer, 1998, pp. 80–94.
[31] K. Sakiyama, Secure design methodology and implementation for embedded public-key cryptosystems, Ph.D. Thesis, Katholieke Universiteit Leuven, Belgium, 2007.
[32] K. Sakiyama, L. Batina, B. Preneel, I. Verbauwhede, Superscalar coprocessor for high-speed curve-based cryptography, CHES 2006, Lecture Notes in Computer Science, vol. 4249, Springer, 2006, pp. 415–429.
[33] P. Schaumont, I. Verbauwhede, A component-based design environment for esl design, IEEE Design & Test of Computers 23 (5) (2006) 338–347.
[34] L. Song, K.K. Parhi, Low-energy digit-serial/parallel finite field multipliers, Journal of VLSI Signal Processing Systems 19 (2) (1998) 149–166.
[35] T. Wollinger, Software and hardware implementation of hyperelliptic curve cryptosystems, Ph.D. Thesis, Ruhr-University Bochum, Germany, 2004.
[36] T. Wollinger, Computer architectures for cryptosystems based on hyperelliptic curves, Master's Thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, May 2001.
[37] Z. Yan, D.V. Sarwate, Z. Liu, High-speed systolic architectures for finite field inversion, Integration, VLSI Journal 38 (3) (2005) 383–398.

Junfeng Fan received the BS and MS degrees in electrical engineering from Zhejiang University, China, in 2003 and 2006, respectively. Since 2006, he has been a Ph.D. student in the Electrical Engineering Department (ESAT), Katholieke Universiteit Leuven (KU Leuven), Belgium. His research interests include computer arithmetics, with special focus on efficient implementations for Public Key Cryptography (PKC). He is also interested in physical security of embedded systems and secure design methodologies.

Lejla Batina is an assistant professor at Radboud University Nijmegen in The Netherlands and a postdoctoral researcher at the research group COSIC in the Electrical Engineering Department of the Katholieke Universiteit Leuven, Belgium.

She received her M.Sc. degree in Mathematics from the University of Zagreb, Croatia in 1995 and Ph.D. degree in engineering from the K.U. Leuven in 2005. She also studied and worked as a research assistant at the Technical University of Eindhoven, The Netherlands from 1999 to 2001. Her research interests include efficient arithmetic for cryptographic algorithms, secure implementations of cryptographic algorithms, side-channel security, lightweight cryptography e.g. crypto for RFIDs, sensor net works, etc.

Ingrid Verbauwhede received the electrical engineering degree and Ph.D. degree from the Katholieke Universiteit Leuven (KU Leuven), Belgium, in 1991. From 1992 to 1994, she was a postdoctoral researcher and visiting lecturer at the Electrical Engineering and Computer Sciences Department, University of California, Berkeley. From 1994 to 1998, she worked for TCSI and ATMEL in Berkeley, California. In 1998, she joined the faculty of University of California, Los Angeles (UCLA). She is currently a professor at the KU Leuven and an adjunct professor at UCLA. At KU Leuven, she is a codirector of the Computer Security and Industrial Cryptography (COSIC) Laboratory. Her research interests include circuits, processor architectures, and design methodologies for real-time embedded systems for security, cryptography, digital signal processing, and wireless communications. This includes the influence of new technologies and new circuit solutions on the design of next-generation systems on chip. She was the program chair of the Ninth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 07), the 19th IEEE International Conference on Application specific Systems, Architectures and Processors (ASAP 08), and the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED 02). She was also the general chair of ISLPED 2003. She was a member of the executive committee of the 42nd and 43rd Design Automation Conference (DAC) as the design community chair. She is a senior member of the IEEE.