Light-weight implementation options for curve-based cryptography: HECC is also ready for RFID

Junfeng Fan, Lejla Batina and Ingrid Verbauwhede
ESAT/SCD-COSIC, Katholieke Universiteit Leuven and IBBT
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{junfeng.fan, lejla.batina, ingrid.verbauwhede}@esat.kuleuven.be

Abstract

In this paper we describe a low footprint implementation of HyperElliptic Curve Cryptography (HECC) for RFID tags. This HECC processor supports divisor multiplication on a hyperelliptic curve defined over $\mathrm{GF}(2^{83})$. *We propose a Unified* $\mathrm{GF}(2^m)$ *Multiplier/Inverter (UMI) which is smaller than ALUs with separated multipliers and inverters. With the UMI divisor multiplications using affine coordinates can be efficiently supported. Since affine coordinates require less registers than projective coordinates, the size of register file is also reduced.*

We choose hyperelliptic curves defined with the $h(x) = x$ *and* $f(x) = x^5 + f_3x^3 + x^2 + f_0$. *The HECC processor, synthesized with 130 nm standard cell library, uses 14.5 kGates. It consumes 13.4* μ*W when running at 300 kHz. One divisor multiplication takes 450* ms*, which makes our solution a feasible option for light-weight applications.*

Keywords: Hyperelliptic Curve Cryptography, Modular multiplication, Modular inversion, RFID

1 Introduction

RFID tags are now widely used in logistics, medical applications, access control and so on. It is also believed that a large scale of utilization of RFID will take place in the near future. Security is one of the key concerns in many applications. By security we mean that an RFID tag should be unclonable and untraceable by attackers. Moreover, the security features built in the RFID tags should not limit the scalability of the system. Protocols [20, 17, 15] based on Public Key Cryptography (PKC) have been suggested. However, implementing PKCs with a sufficient security level on a passive RFID tag is a big challenge in terms of area, power and energy.

Several PKC implementations have been reported for passive RFID tags. For instance, Elliptic Curve Cryptosystem (ECC) was implemented for RFID tags [13, 16]. According to [1], a passive RFID tag should have power consumption less than 15 μW to guarantee a 1 meter operation range. Some ECC implementations can already fulfill the requirements. For example, ECC processor proposed by Lee $et\ al.$ [16], using 15.4 kGates with 130 nm technology, consumes 12.08 μW when running at 323 kHz, and one scalar multiplication takes only 243 ms. The ECC core proposed by Hein $et\ al.$ [13] consumes 10.08 μW when running at 106 kHz.

In this paper, we propose a compact implementation of Hyperelliptic Curve Cryptography (HECC) targeting RFID tags. HECC can be used for all the protocols that are based on ECC. To the best of our knowledge, there is no patents on HECC. We show that HECC can also fulfill the requirements with a comparable performance. We propose a Unified Multiplier and Inverter (UMI) for $\mathrm{GF}(2^m)$ arithmetic as well as a smaller register file. With 130 nm technology, the HECC core takes about 14.5 kGates. Running at 300 kHz, the power consumption is about 13.4 μW according to our simulation results. One divisor multiplication takes 136k clock cycles.

The rest of the paper is organized as follows. Section 2 gives the mathematical background of HECC and field arithmetic. Section 3 describes the methods that we propose to reduce the footprint of the HECC processor. The architecture of the proposed HECC processor is described in Sect. 4. In Sect. 5 we present the implementation results. We conclude the paper and give some future works in Sect. 6.

2 Mathematical Background

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 $g \geq 1$.

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

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

where $h(x) \in \mathrm{GF}(2^m)[x]$ 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 $(x, y) \in \overline{\mathrm{GF}}(2^m) \times \overline{\mathrm{GF}}(2^m)$ which simultaneously satisfy the equation (1) and the equations: $2v + h(u) = 0, h'(u)v - f'(u) = 0$. These points are called singular points. 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.* $D = \sum m_P P$ and its degree is $degD = \sum m_P$. 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 $J = Div_0/P$. Here P is the set of all principal divisors, where a divisor D is called principal if $D = div(f)$, for some element f of the function field of C ($div(f) = \sum_{P \in C} ord_P(f)P$). 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 usually used. Here, u is monic of degree 2, $deg(v) < deg(u)$ and $u|f - hv - v^2$ (so-called reduced divisors). For implementations of HECC, we need to implement the multiplication of elements of the Jacobian *i.e.* divisors with some scalar.

2.2 Field Arithmetic

An element α in $\mathrm{GF}(2^m)$ can be represented as a polynomial $A(x)$=$\sum_{i=0}^{m-1} a_i x^i$, where a_i is an element of $\mathrm{GF}(2)$. Addition of two elements in $\mathrm{GF}(2^m)$ is performed as polynomial addition in $\mathrm{GF}(2)$

$$\sum_{i=0}^{m-1} a_i x^i + \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{m-1} (a_i \oplus b_i) x^i,$$

where $\oplus$ is the XOR operation.

2.2.1 Multiplication

In the literature there are various algorithms and architectures [4, 21] proposed for modular multiplication in $\mathrm{GF}(2^m)$. Algorithm 1 shows a bit-serial multiplication algorithm.

Algorithm 1 MSB-first bit-serial multiplication algorithm in $\mathrm{GF}(2^m)$

Input: $A(x) = \sum_{i=0}^{m-1} a_i x^i$, $B(x) = \sum_{i=0}^{m-1} b_i x^i$, irreducible binary polynomial $P(x)$ with $\deg(P(x)) = m$.
Output: $A(x)B(x) \bmod P(x)$.

1: $C(x)(= \sum_{i=0}^{m} c_i x^i) \leftarrow 0$;
2: **for** $i = m - 1$ to 0 **do**
3: $\quad C(x) \leftarrow x(C(x) + c_m P(x) + b_i A(x))$;
4: **end for**

Return: C(x)/x.

2.2.2 Inversion

The multiplicative inverse of $\alpha \in \mathrm{GF}(2^m)$ is an element $\beta \in \mathrm{GF}(2^m)$ such that $\alpha\beta \equiv 1 \bmod P(x)$, where β is denoted as α^{-1}. Compared with the other modular operations, modular inversion is considered as a computationally expensive operation. Thus, projective coordinates are proposed to avoid inversions. However, one inversion is still required to convert projective coordinates to affine coordinates.

The most commonly used methods to perform the modular inversion are based on Fermat's little theorem [2], Extended Euclidean Algorithm [14] and Gaussian elimination [12]. EEA is widely used in practice. The schoolbook EEA-based inversion algorithm in $\mathrm{GF}(2^m)$ is considered inefficient due to the long polynomial division in each iteration. This problem was partially solved by replacing degree comparison with a counter [5]. Many variants of EEA [23, 11] have been proposed. Architecture for inverter and divider [6, 9] have also been reported using EEA-based algorithms.

2.3 HECC implementations

HECC has been implemented in both hardware [7, 19, 22, 3, 8] and software [18]. Most of the implementations are targeting high performance. For example, the implementation of HECC described in [22], using affine coordinates, uses three modular multipliers and two modular inverters. It uses 7785 slices on the Xilinx Virtex II FPGA(XC2V4000), and finishes one divisor multiplication of HECC over $\mathrm{GF}(2^{81})$ in 415 μs. Sakiyama reported a compact data-path for HECC targeting RFID tags [19]. Implemented with 130 nm standard cell library, the HECC processor takes 7652 gates and the estimated power consumption is 19 μW when it's running at 500 kHz. Note that the area and power consumption in [19, 3] do not include data memory.

In this paper, we try to minimize the area of both datapath and the data memory. We use affine instead of projective coordinates, thus the number of registers to store intermediate values is reduced. In order to improve the perfor-

mance, we also use a unified multiplier and inverter, which supports fast inversion.

3 Reducing the footprint

We use several techniques to reduce the footprint of the HECC processor. The register file occupies the biggest portion of the total area. Thus, reducing the size of register file is the key step towards a compact HECC processor. Besides, the modular arithmetic logic unit (MALU) is also of vital importance in terms of size and performance of the processor.

3.1 Reducing the register file size

The number of registers is determined by divisor multiplication algorithm. As described in Sect. 2, one can use affine coordinates and projective coordinates for divisor addition and doubling. Choosing projective or affine coordinates, when targeting a high performance implementation, is decided by the efficiency of multiplication and inversion. However, affine coordinates require less registers for storing intermediate results since they do not include Z coordinates. Moreover, using affine coordinates also reduces the number of intermediate results. Our investigation shows that 12 registers are sufficient for scalar multiplication with flexible base divisor.

Table 1 and 2 show the memory allocation of divisor doubling and divisor addition.

Table 1. Divisor doubling.

Input: {R4,R5,R6,R7} = D1(={u10,u11,v10,v11}).		
R3:= R4*R4;	R4:= R5*R5+f3;	R6:= R6*R6+f0;
R6:= 1/R6;	R6:= R6*R3;	R2:= R4*R6;
R0:= R2+R5;	R5:= R6*R6;	R1:= R6+R4 ;
R4:= R0*R0+R6;	R2:= R1*R2+f2;	R2:= R6*R5 +R2;
R7:= R7*R7+R2;	R6:= R1*R4 +R3;	
Return: {R4,R5,R6,R7} = 2*D1.		

3.2 Reducing the ALU size

There are several techniques to reduce the size of the data-path. First, fixing the underlying field can significantly reduce the complexity of modular reduction, resulting in a smaller multiplier. Second, using bit-serial multiplier or digit-serial multiplier with small digit size is also beneficial. Third, using a smaller multiplier, such as 16x16 bit instead of full length digit-serial multiplier, is also a good strategy in general to reduce the size and power [13].

In this paper, we use affine coordinates, thus a fast inverter is required. We propose a unified architecture for inversion and multiplication. The inversion algorithm is

Table 2. Divisor addition.

Input: {R4,R5,R6,R7} = D1(={u10,u11,v10,v11}), {R8,R9,R10,R11} = D0(={u00,u01,v00,v01}).		
R0:=R5+R9 ;	R1:=R0*R0 ;	R1:=R1*R4 ;
R2:=R5*R0 ;	R3:=R8+R4 ;	R2:=R2+R3 ;
R3:=R3*R2+R1 ;	R6:=R6+R10 ;	R1:=R2*R6 ;
R7:=R7+R11 ;	R6:=R7+R6 ;	R0:=R5+R9 ;
R7:=R0*R7 ;	R2:=R2+R0 ;	R6:=R2*R6+R1 ;
R6:=R7*R5+R6 ;	R6:=R7+R6 ;	R7:=R4*R7+R1 ;
R2:=R3*R6 ;	R2:=1/R2 ;	R6:=R6*R6 ;
R6:=R6*R2 ;	R2:=R3*R2 ;	R3:=R3*R2 ;
R4:=R4+R3 ;	R7:=R7*R2 ;	R0:=R9+R5 ;
R5:=R7+R5 ;	R7:=R7+R0 ;	R4:=R5*R7+R4 ;
R7:=R7+R0 ;	R1:=R9*R7+R8 ;	R4:=R4+R1 ;
R5:=R3*R3 ;	R3:=R8*R7 ;	R4:=R0*R5+R4;
R5:=R0+R5 ;	R7:=R9+R7 ;	R7:=R7+R5 ;
R0:=R5*R7+R4;	R0:=R0+R1 ;	R7:=R4*R7+R3 ;
R0:=R0*R6+R11 ;	R6:=R7*R6+R10 ;	R7:=R0+1 ;
Return: {R4,R5,R6,R7} = D1 + D0.		

Algorithm 2 Left-Shift EEA-Based Inversion Algorithm

Input: irreducible binary polynomial $P(x)$ with $\deg(P(x)) = m$, polynomial $A(x)$ with $\deg(A(x)) < m$.
Output: $A^{-1}(x) \bmod P(x)$.

1: $R(x) \leftarrow P(x)$, $S(x) \leftarrow A(x)$, $H(x) \leftarrow 0$, $J(x) \leftarrow x^{-m}$, $d \leftarrow 0$;
2: **for** $i = 0$ to $2m - 1$ **do**
3: **if** s_m=1 & $d \geqslant 0$ **then**
4: $[R(x), S(x)] \leftarrow [S(x), x(S(x) + R(x))]$;
$[H(x), J(x)] \leftarrow [J(x), x(H(x) + J(x)) \bmod P(x)]$;
$d = -d + 1$;
5: **else**
6: $[R(x), S(x)] \leftarrow [R(x), x(s_m R(x) + S(x))]$;
$[H(x), J(x)] \leftarrow [H(x), x(s_m H(x) + J(x)) \bmod P(x)]$;
$d = d + 1$;
7: **end if**
8: **end for**

Return: $H(x)$.

proposed in [9]. Note that here x^{-m} is precomputed and hardwired.

Figure 3.2 shows the data-path of our proposed digit-serial inverter and multiplier. Note that in order to further reduce the area, we choose a low Hamming-Weight irreducible polynomial, $P(x) = x^{83} + x^7 + x^4 + x^2 + 1$. Only 5 AND gates and 5 XOR gates are required to perform $(a_m P(x) + A(x))$. The architecture realizes both Alg. 1 and Alg. 2. The multiplier and the inverter share AND-XOR cells and two registers.

This data-path supports the following operations.

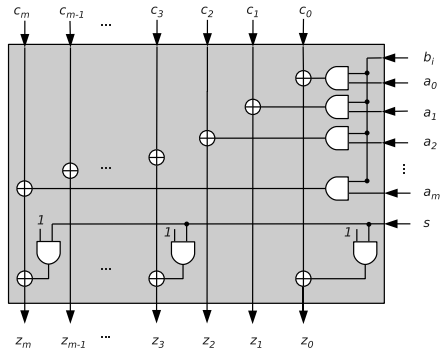
Modular Multiplication In the multiplication mode, the data-path realizes Alg. 1 by taking the following configuration:

- $s_i \leftarrow c_m$, $S(x) \leftarrow S\_J(x)$, $R(x) \leftarrow R(x)$.

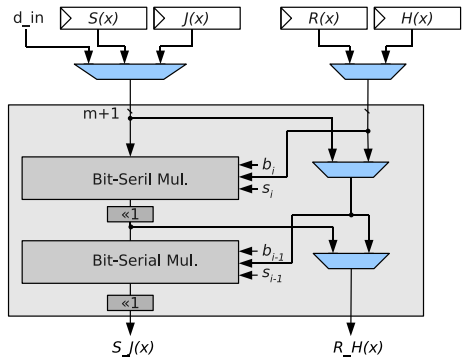
In this case, $R(x)$ and $J(x)$ are not updated, thus both can be used to store intermediate results. The final result of multiplication is stored in register $S(x)$.

Modular Inversion In the inversion mode, $R(x) - S(x)$ pair and $H(x) - S(x)$ pair are updated alternatively. For $R(x) - S(x)$ pair, there is no need of reduction, thus s is set to 0. For $H(x) - J(x)$ here, reduction may be needed depending on the value of h_m and s_m. Note that the choice of branch only depends on s_m and d, thus $R(x)$ and $S(x)$ can be updated first. The information about the branch that has been taken is recorded in a register, afterwards $H(x)$ and $J(x)$ are updated accordingly.

- $d > 0$, $s_m = 1$,
  $R(x) \leftarrow S(x)$. $S(x) \leftarrow x(R(x) + S(x))$.
  $H(x) \leftarrow J(x)$, $J(x) \leftarrow x(J(x) + H(x) + (h_m + j_m)P(x))$.



(a) Bit-serial modular multiplier



(b)Digit-serial modular multiplier/inverter ($d = 2$).

Figure 1. Unified Multiplier and Inverter (UMI).

- $d \leq 0$, $s_m = 1$,
  $R(x) \leftarrow R(x)$. $S(x) \leftarrow x(R(x) + S(x))$.
  $H(x) \leftarrow H(x)$, $J(x) \leftarrow x(J(x) + H(x) + (h_m + j_m)P(x))$.

- $s_m = 0$,
  $R(x) \leftarrow R(x)$. $S(x) \leftarrow xS(x)$.
  $H(x) \leftarrow H(x)$, $J(x) \leftarrow x(J(x) + j_m P(x))$.

When choosing the digit-size d, one multiplication in $\mathrm{GF}(2^m)$ takes $\lceil m/d \rceil$ cycles, while one inversion takes $\lceil 4m/d \rceil$ cycles.

4 HECC processor architecture

The HECC processor is shown in Fig. 2. It contains an Instruction ROM, a main controller, a unified modular multiplier/inverter, a Register File, and an input/output interface. The Instruction ROM contains the field operation sequences of divisor addition and doubling. The main controller calls different routines, i.e. load, store, divisor addition and divisor doubling, depending on the key bit.

We use ten 84-bit registers in the register file. During the scalar multiplication, 8 registers are used to store divisor D0 and D1. We need four registers to store temporary data, thus 12 registers in total. However, the register $R(x)$ and $J(x)$ in the unified multiplier and inverter are used only for inversion. Thus, we could use them as temporary storage. As a result, the register file contains only 10 registers.

5 Implementation Results

The processor is described with Gezel [10] language and synthesized with design compiler. We implemented the

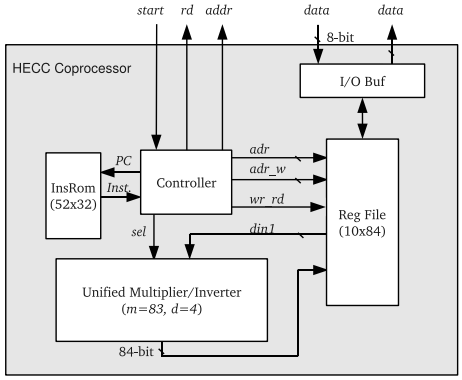


Figure 2. Block diagram of the proposed HECC processor.

HECC processor, as shown in Figure 2, with 130 nm standard cell library. Table 3 summarizes the area and power of the proposed design.

Our HECC implementation uses 14.5 kGates and finishes one divisor multiplication in 136838 clock cycles. The power consumption, estimated with power compiler, is around 13.4 μW when running at 300 kHz. The implementation of [19], using projective coordinates, requires 266133 clock cycles for one scalar multiplication. Note that the implementation in [19] is defined on a smaller field and the result does not include data memory. The power and energy consumption of our design is also significantly lower when achieving the same delay compared to that of [19]. The main speedup of our design comes from the fast inverter and the use of affine coordinates.

There are many ECC implementations proposed for RFID tags. Lee $et\ al.$ [16] explored the trade-off between area and power of ECC implementation using digit-serial multiplier with various digit size. From $d = 2$ to $d = 4$, the number of clock cycles for one scalar multiplication is halved, while the area increases only 10%. A trade-off can be made between power, performance and energy.

The architecture proposed in [13] uses a different approach. It utilizes 16x16 $\mathrm{GF}(2)$ multiplier and 32-bit accumulator. With shorter registers, the power consumption can be significantly reduced. On the other hand, it requires 296k clock cycles, twice as many as our HECC implementation, for one scalar multiplication, which makes the overall energy requirement significantly higher.

Our HECC processor can meet the requirements for passive RFID tags in terms of area, power and energy. However, our current implementation can hardly match the energy efficiency of some ECC implementations [16]. This is due to the fact that the computational complexity of HECC divisor scalar multiplication is higher than the point multiplication for ECC. Because divisor operations in HECC are much more complicated than point operations in ECC, almost double number of clock cycles is required for one scalar multiplication even if the multiplier has the same digit size.

Note that our current implementation is based on binary scalar multiplication method. No countermeasures for side-channel attacks are deployed. The ECC implementations [16, 13] use Montgomery scalar multiplication, which is believed to be secure against simple power analysis. Adding countermeasures normally causes area increase, performance degradation or both.

6 Conclusions

We describe a low-power HECC implementation for RFID tags. The reported HECC processor uses a unified modular multiplier/inverter to support fast scalar multiplication using affine coordinates. With a reduction on the size of register file, the area of the processor is largely reduced. The HECC implementation uses 14.5 kGates and finishes one scalar multiplication in 136838 clock cycles. The overall power consumption is estimated at 13.4 μW when running at 300 kHz. The results show that HECC can fully meet the requirements for passive RFID tags.

For the future study, we would like to include countermeasures against side-channel attacks. Also, in order to improve energy efficiency, a design trade-off between energy and area can be explored by adjusting the digit-size of the multiplier/inverter.

Acknowledgements

This work is supported by IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by FWO projects G.0300.07, by the European Comission through the IST Programme under Contract IST-2002-507932 ECRYPT NoE, and by the K.U. Leuven-BOF.

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, and S. Tsujii. Generalised fast algorithm for computing multiplicative inverses in GF(2^m). *Electronics Letters*, 25(10):664–665, 11 May 1989.

[3] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede. Public-Key Cryptography on the top of a needle. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 1831–1834, May 2007.

[4] T. Beth and D. Gollman. Algorithm engineering for public key algorithms. *Selected Areas in Communications, IEEE Journal on*, 7(4):458–466, May 1989.

[5] R. P. Brent and H. T. Kung. Systolic VLSI Arrays for Polynomial GCD Computation. *IEEE Trans. Computers*, 33(8):731–736, 1984.

[6] A. K. Daneshbeh and M. A. Hasan. A class of unidirectional bit serial systolic architectures for multiplicative inversion and division over GF(2^m). *Computers, IEEE Transactions on*, 54(3):370–380, March 2005.

Table 3. Performance comparison of HECC and ECC implementations targeting RFID tags.

Ref. Design	Platform	Finite Field	Area [kGates]	Perf. [#cycle]	Freq. [kHz]	Power [μW]	Energy* [μJ]	Comments
HECC This work	130 nm ASIC	$GF(2^{83})$	14.5	136838	300	13.4	6.03	Unified mult./inv. ($d = 4$)
HECC Sakiyama [19]	130 nm ASIC	$GF(2^{67})$	7.6†	266133	500	19†	10.0†	1 Mult. ($d = 8$)
ECC Lee *et al.* [16]	130 nm ASIC	$GF(2^{163})$	14.1‡	144842	590	21.55	5.29	($d = 2$)
			14.7‡	101183	411	15.75	3.88	($d = 3$)
			15.4‡	78544	323	12.08	2.94	($d = 4$)
ECC Hein *et al* [13]	180 nm ASIC	$GF(2^{163})$	11.9	296000	106	10.8	31.3$^\diamond$	16x16 mult.

† Modular Arithmetic Logic Unit only
‡ Including ECC core and an 8-bit controller for cryptographic protocols
* Energy for one scalar multiplication
$^\diamond$ Estimated by authors

[7] G. Elias, A. Miri, and T. H. Yeap. On efficient implementation of FPGA-based hyperelliptic curve cryptosystems. *Computers and Electrical Engineering*, 33(5-6):349–366, 2007.

[8] J. Fan, L. Batina, and I. Verbauwhede. HECC Goes Embedded: An Area-efficient Implementation of HECC. In *Proceedings of 15th Annual Workshop on Selected Areas in Cryptography (SAC 2008)*. Lecture Notes in Computer Science, R. Avanzi, L. Keliher and F. Sica (eds.), Springer-Verlag, 2008.

[9] J. Fan and I. Verbauwhede. Extended abstract: Unified digit-serial multiplier/inverter in finite field GF(2^m). In *IEEE International Workshop on Hardware-Oriented Security and Trust – HOST*, pages 72–75, June 2008.

[10] GEZEL. `http://rijndael.ece.vt.edu/gezel2/`.

[11] J-H Guo and C-L Wang. A novel digit-serial systolic array for modular multiplication. *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, 2:177–180 vol.2, 31 May-3 Jun 1998.

[12] M. A. Hasan and V. K. Bhargava. Bit-serial systolic divider and multiplier for finite fields GF(2^m). *Computers, IEEE Transactions on*, 41(8):972–980, Aug 1992.

[13] D. Hein, J. Wolkerstorfer, and N. Felber. ECC is ready for RFID A Proof in Silicon. In *Proceedings of 15th Annual Workshop on Selected Areas in Cryptography (SAC 2008)*. Lecture Notes in Computer Science, R. Avanzi, L. Keliher and F. Sica (eds.), Springer-Verlag, 2008.

[14] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1981.

[15] Y. Lee, L. Batina, and I. Verbauwhede. Untraceable RFID Authentication Protocols: Revision of EC-RAC. In *IEEE International Conference on RFID – RFID 2009*, Orlando, Florida, USA, April 2009.

[16] Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-curve-based security processor for RFID. *IEEE Trans. Comput.*, 57(11):1514–1527, 2008.

[17] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 31–53, London, UK, 1993. Springer-Verlag.

[18] J. Pelzl. Hyperelliptic Cryptosystems on Embedded Microprocessors. Master's thesis, Ruhr-Universitat Bochum, Sep, 2002.

[19] K. Sakiyama. *Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2007.

[20] Claus P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 239–252, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[21] L. Song and K. K. Parhi. Low-energy digit-serial/parallel finite field multipliers. *J. VLSI Signal Process. Syst.*, 19(2):149–166, 1998.

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

[23] Z. Yan, D. V. Sarwate, and Z. Liu. High-speed systolic architectures for finite field inversion. *Integration, VLSI Journal*, 38(3):383–398, 2005.