Finding Optimum Parallel Coprocessor Design for Genus 2 Hyperelliptic Curve Cryptosystems

Guido Bertoni and Luca Breveglieri
Politecnico di Milano, Italy
{bertoni,breveglieri}@elet.polimi.it

Thomas Wollinger and Christof Paar
Communication Security Group (COSY)
Ruhr-Universitaet Bochum, Germany
{wollinger,cpaar}@crypto.rub.de

Abstract

Hardware accelerators are often used in cryptographic applications for speeding up the highly arithmetic-intensive public-key primitives, e.g. in high-end smart cards. One of these emerging and very promising public-key scheme is based on HyperElliptic Curve Cryptosystems (HECC). In the open literature only a few considerations deal with hardware implementation issues of HECC.

Our contribution appears to be the first one to propose architectures for the latest findings in efficient group arithmetic on HEC. The group operation of HECC allows parallelization at different levels: bit-level parallelization (via different digit-sizes in multipliers) and arithmetic operation-level parallelization (via replicated multipliers). We investigate the trade-offs between both parallelization options and identify speed and time-area optimized configurations. We found that a coprocessor using a single multiplier ($D = 8$) instead of two or more is best suited. This coprocessor is able to compute group addition and doubling in 479 and 334 clock cycles, respectively. Providing more resources it is possible to achieve 288 and 248 clock cycles, respectively.

Keywords: hyperelliptic curve, hardware architecture, coprocessor, parallelism, genus 2, embedded processor.

1 Introduction

All modern security protocols, such as IPSec, SSL and TLS use symmetric-key as well as public-key cryptographic algorithms. In order to be able to provide highly arithmetic-intensive public-key cryptographic primitives, hardware accelerators are often used. An example are high-end smart cards, where a cryptographic coprocessor takes over all the expensive (area and time) computations.

In practical applications the most used public-key algorithms are RSA and Elliptic Curve Cryptosystems (ECC). One emerging and very promising public-key scheme is the HyperElliptic Curve Cryptosystem (HECC). HECC has been analyzed and implemented only recently both in software [10, 11, 15, 17, 20, 21, 23–25, 27, 33] and in more hardware-oriented platforms such as FPGAs [4, 31, 32].

The work at hand presents, for the first time, an architecture for a HECC coprocessor considering the most recent explicit formulae to compute group operations. All of the previous work implementing HECC in hardware used the original Cantor algorithm, which is outdated. Furthermore, we present and evaluate different design options for the HECC coprocessor. In order to do so, we wrote software capable of scheduling the necessary operations, resulting in an optimal architecture with respect to area and speed. Parallelizing at the bit and arithmetic operation level we found that: 1) no more than three multiplier units are useful; 2) architectures implementing one inversion and one multiplication unit are the best choice; 3) and providing sufficient resources group addition and doubling can be performed in 288 and 248 clock cycles, respectively. Moreover, we explored the overlapping of two group operations and we analyzed the usage of registers.

The rest of the paper is organized as follows. Section 2 summarizes the contributions dealing with previous works. Section 3 gives a brief overview of the mathematical background of HECC. Section 4 presents the architecture of the HECC coprocessor and Section 5 the used methodology. Finally, we end this contribution with a discussion of our results (Section 6) and some conclusions (Section 7).

2 Previous Work

This section gives a short overview of the hardware implementations targeting HECC and of the previous research work to parallelize hardware ECC.

The first work discussing hardware architectures for the implementation of HECC appeared in [31, 32]. The authors describe efficient architectures to implement the necessary

field operations and polynomial arithmetic in hardware. All of the presented architectures are speed and area optimized. In [31], they also estimated that for a hypothetical clock frequency of 20 MHz, the scalar multiplication of HECC would take 21.4 ms using the window NAF method.

In [4] the authors presented the first complete hardware implementation of a hyperelliptic curve coprocessor. This implementation targets a genus-2 HEC over $\mathbb{F}_{2^{113}}$. The target platform is a Xilinx II FPGA. Point addition and point doubling with a clock frequency of 4 5MHz take 105μs and 90μs, respectively. The scalar multiplication could be computed in 10.1 ms.

Note that publications [4, 31, 32] adopt the Cantor algorithm to compute group operations. Today, there exist more efficient algorithms to compute group addition and group doubling, the so-called explicit formulae (for more details see Section 3.2).

In [16] the authors proposed a parallelization of the explicit group operation of HECC. They developed a general methodology for obtaining parallel algorithms. The methodology guarantees that the obtained parallel version requires a minimum number of rounds. They show that for the inversion free arithmetic [12] using 4, 8 and 12 multipliers in parallel, scalar multiplication can be carried out in 27, 14 and 10 parallel rounds, respectively. When using affine coordinates [11] and 8 multipliers it can be performed in 11 rounds, including an inversion round.

Note that for an effective implementation it is impractical to use so many multipliers in parallel as stated in [16]. The work at hand attempts to consider not only the minimum number of rounds (speed), but also the necessary devices (area) as well as practical applications.

A similar work as that presented here for HECC can be found in [3] for ECC, where a study of the trade-off between the number of operators and different coordinate systems is presented. In [2] two scalar multiplications are scheduled in parallel on the same architecture: the two operations are executed in different coordinate systems to improve the use of the operators. Note that the group operations of elliptic curves are much less complex than those of the hyperelliptic ones. In ECC the silicon area of the possible architecture is easily bounded since the critical path can be computed by hand, while in the case of HECC it is much more complex.

3 Mathematical Background

In this section we introduce briefly the theory of HECC, restricting attention to the material relevant for this work only. The interested reader is referred to [1, 9] for more background on HECC.

3.1 Definition of HECC

Let $\mathbb{F}$ be a finite field and let $\overline{\mathbb{F}}$ be the algebraic closure of $\mathbb{F}$. A hyperelliptic curve C of genus $g \geq 1$ over $\mathbb{F}$ is the set of the solutions $(x, y) \in \mathbb{F} \times \mathbb{F}$ to the following equation:

$$C : y^2 + h(x)y = f(x)$$

The polynomial $h(x) \in \mathbb{F}[x]$ is of degree at most g and $f(x) \in \mathbb{F}[x]$ is a monic polynomial of degree $2g + 1$. For odd characteristic it suffices to let $h(x) = 0$ and to have $f(x)$ squarefree. Such a curve C is said to be non-singular if there does not exist any pair $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ satisfying the equation of the curve C and the two partial differential equations $2y + h(x) = 0$ and $h'(x)y - f'(x) = 0$.

The so-called divisor D is defined as follows: $D = \sum m_i P_i$, to be a formal weighted sum of points P_i of the curve C (and the integers m_i are the wights), with the additional condition that $D^\sigma = \sum m_i P_i^\sigma$ is equal to D for all the automorphisms σ of $\overline{\mathbb{F}}$ over $\mathbb{F}$ (see [1] for details).

Divisors admit a reduced form. A reduced divisor can be represented as a pair of polynomials $u(x)$, $v(x)$ [18, page 3.17]. Reduced divisors can be added (group addition), e.g. $D_3 = D_1 + D_2$, or doubled (group doubling), e.g. $D_2 = 2D_1 = D_1 + D_1$, and hence the so-called scalar multiplication $kD = D + \cdots + D$ for k times is defined. The scalar multiplication kD is the basic operation of HECC, that we want to implement with a coprocessor.

3.2 Group Operations

The formulae given for the group operations (addition, doubling) of HEC by Cantor [6] can be rewritten in explicit form, thus resulting in more efficient arithmetic. The explicit formulae were first presented in [7]. Starting with this finding, a considerable effort of different research groups has been put into finding more efficient operations. The group operations of genus-2 curves have been studied most intensively ([7, 11–15, 17, 22, 29]), but also group operations on genus-3 curves ([10, 20, 21, 33]) and even genus-4 curves ([23]) have been considered.

In the work at hand, we target our HECC coprocessor for genus-2 curves using underlying fields of characteristic two. We used the up-to-date fastest explicit formulae, as presented in [22], where the authors introduced a group doubling requiring a single field inversion, 9 field multiplications and 6 field squarings. Group addition can be computed with 1 field inversions, 21 field multiplications and 3 field squarings.

3.3 Security of HECC

It is widely accepted that for most cryptographic applications based on EC or HEC, the necessary group is of order

at least $\approx 2^{160}$. Thus, for HECC over $\mathbb{F}_q$, we must have at least $g \cdot \log_2 |\mathbb{F}_q| \approx 160$. In particular, we will need a field order $|\mathbb{F}_q| \approx 2^{80}$ for genus-2 curves. Even the very recent attack found by Thériault [30] shows no progress in attacks against genus-2 HEC.

4 Architecture of the HECC coprocessor

To implement the coprocessor we chose a standard architecture, see Figure 1. It contains a register file to store temporary results and outputs. The size of each register was chosen to be the dimension of the field, namely 81 bits. The register file has two output ports to feed the operators and one input port to receive the result. This guarantees feasibility and ease of implementation. At any given clock cycle only one field operation can start. If the operation is unary, such as inversion, one bus remains idle.

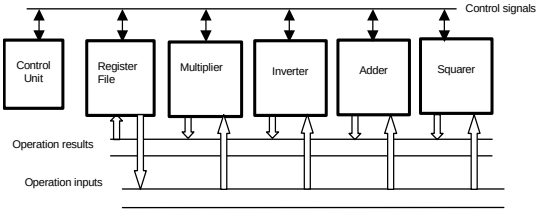

Figure 1. Crypto-processor architecture.

The following list is a summary of how we implemented the field arithmetic:

- Addition: The addition of two elements requires the modulo 2 addition of the coefficients of the field elements.

- Squaring: The squaring of a field element $A = \sum_{i=0}^{m-1} a_i x^i$ is ruled by the following equation: $A^2 \equiv \sum_{i=0}^{m-1} a_i x^{2i} \bmod F(x)$. Further details can be found in [19].

- Multiplication: We decided to use digit multipliers, introduced in [28] for fields GF(2^m). This kind of multiplier allows a trade-off between speed, area and power consumption. It works processing several multiplicand coefficients at the same time. The number of coefficients processed in parallel is the digit-size D. Given D, we denote by $d = \lceil m/D \rceil$ the total number of digits in a polynomial of degree $m-1$. Hence, $C \equiv AB \bmod F(x) = A \sum_{i=0}^{d-1} B_i x^{Di} \bmod F(x)$.

- Inversion: The inversion is computed using the algorithm proposed in [5]. It is based on a modification of Euclid's algorithm for computing the gcd of two polynomials. The asymptotic complexity is linear with the modulus both in time and area.

Table 1. Components of the coprocessor: area and time.

	Area		Latency [Clock cycles]
		Total number of gates	
Add	[m]XOR	[m]	1
Sqr [19]	[m+t+1]XOR	[m+t+1]	1
Mult [28]	$[D \cdot m]$AND & $[D \cdot m]$XOR	$[2Dm]$	$\lceil m/D \rceil$
Inv [5]	$[6 \cdot m + log_2 m]$AND & $[6 \cdot m + log_2 m]$ XOR	$[2(6m + log_2 m)]$	$2 \cdot m$

In Table 1 we give the area and latency for each arithmetic components we used. The given estimates assume 2-input gates and optimum field polynomials $F(x) = x^m + \sum_{i=0}^{t} f_i x^i$, where $m - t \geq D$.

5 Methodology

In this section we describe briefly our approach to find the best suited architecture for the HECC coprocessor.

1. Input: First we evaluated the most recent findings regarding the group operation of HECC. The given formulae were then prepared for the scheduler.

2. Scheduler: Our own software library, especially developed to schedule the HECC group operations, is the heart of our methodology. The scheduler is based on the method known as Operation Scheduling [8] and works accordingly to the As Soon As Possible (ASAP) policy. There is a list of operations that should be executed by the architecture. The scheduler takes one operation at a time and searches for the earliest time slot where the operation can be executed. It is constrained by the number of available resources and by the different times required to execute each operation.

   The same methodology is used by compilers for scheduling machine instructions. It should be noted that this methodology is heuristic and does not grant optimal results. To reach the optimal scheduling it is necessary to use other methods instead, see [8]. The scheduler has the following parameters:

   - HECC formulae.
   - Implementation method for addition, inversion, multiplication and squaring.
   - Number of multiplication units.
   - Different digit-sizes for the multiplier.
   - Properties of the bus.

- Memory access time.

3. Testing: The results of the scheduler were tested by applying test vectors. In order to do so we implemented HECC group operations with the NTL library [26].

4. Analysis: The results were analyzed and, if needed, the input architecture was changed, in order to find a better structure for the coprocessor.

Traditionally, when evaluating the performance of cryptographic implementations, emphasis was put first on the throughput of the implementation and, second, on the amount of hardware resources consumed to achieve the desired throughput. We will consider both the hardware requirements and the time constraints of the cryptographic application. Hence, we are going to use the area-time product and the optimal implementation will reach the highest throughput consuming the smallest area.

We designed an architecture for the HECC coprocessor using different design criteria. We varied the number of multipliers (one, two, three and four), as well as their digit-size D ($D = 2, 4, 8, 16$). Hence, we changed different architecture options, the processing power of the whole coprocessor and of each individual multiplier. Increasing the processing power yields a speed up of the group operation, but also causes a growth in area. Thus, there must exist an optimum architecture, where the area-time product is minimal.

Our goal is to implement a simple controller computing group operations with a fixed execution order. Hence, we look at a static schedule. The alternative would be to implement a finite state machine executing the schedule directly, controlling the availability of resources and deciding which operation should be executed. This solution is feasible but we consider it as too complex and expensive compared to a simple controller executing the operations in a fixed order.

Recent research in hyperelliptic curve cryptosystems provides a large number of explicit formulae to choose from for computing group operations. We chose the up-to-date fastest ones, as are given in [22].

6 Results

In this section the results of the scheduling methodology are presented. Our results are based in the following considerations: i) we choose a set of keys to schedule the operations; ii) we schedule group addition and doubling, accordingly to the values of the key; iii) we considered a long sequence of concatenated group operations. If one implements scalar multiplication using the double-and-add algorithm, the following consecutive group operations should be scheduled: addition after doubling, doubling after doubling and doubling after addition.

We examined all the cases in order to find out whether we could schedule the second group operation in a way to gain speed and achieve a higher hardware utilization. Our results show that addition is always scheduled in the same way. As for doubling, there are two different ways to execute it, depending on the previous operation. When doubling is executed after another doubling, we can gain speed; thus we decided to integrate two options for computing doubling. We have to allow negligible extra hardware for the controller to decide which option to choose.

6.1 Time Requirements of the Group Operations

Tables 2 and 3 show the clock cycles necessary for performing a group operations in the different system configurations. Table 3 shows two different latency figures because the time necessary to perform doubling depends on the operation executed before. The leftmost figure in each cell of Table 3 is the number of clock cycles necessary to compute doubling after doubling. The rightmost instead is the time latency of doubling after addition.

One can see that by increasing the digit-size and the number of multipliers, the time necessary to execute a group operation decreases. For group addition our results show that the performance does not increase in some cases, even when augmenting the resources of the system. For example, focusing on the speed of addition, using three rather than four digit-size multipliers with $D = 8$ (Table 2, third row) does not make any performance difference. The same behavior can be observed for 2, 3 and 4 multipliers with digit-size 16. This is due to the structure of the group operation, that shows no additional parallelism.

Focusing on doubling, there is almost no performance gain in moving from 3 to 4 multipliers, no matter of what kind (Table 3, rightmost two columns). In addition, there is no performance gain in providing 2, 3 or 4 multipliers of digit-size 8 or 16. Taking only performance into account, one concludes from Tables 2 and 3 that the design option using one inverter, two multipliers ($D = 16$), one adder and one squarer is preferable. The architecture can perform group doubling and addition in 289 and 248 clock cycles, respectively. As stated in Section 5, the ASAP scheduling policy does not grant an optimum solution; this is evident in the case of 3 or 4 multipliers with $D = 16$. The scheduling of doubling after addition is worse than the case with two multipliers.

6.2 Parallel Computation of Group Operations

In the case of genus-2 HEC each group operation produces as output one polynomial of degree two and one monic polynomial of degree three. Hence, the output consists of 4 coefficients, namely four field elements. They are neither produced at the same time nor are all necessary to start the

Table 2. Clock cycles per group addition.

Digit-size	Number of multipliers			
	1	2	3	4
2	1259	739	664	635
4	739	479	444	435
8	479	349	335	335
16	356	289	288	288

Table 3. Clock cycles per group doubling.

Digit-size	Number of multipliers			
	1	2	3	4
2	724 / 846	486 / 560	458 / 490	458 / 484
4	464 / 526	346 / 380	338 / 350	338 / 344
8	334 / 366	278 / 286	278 / 279	278 / 279
16	274 / 284	248 / 247	248 / 250	248 / 250

next group operation right away. This means that when one field element (one coefficient) is computed, it can be used by the next group operation. We measure the overlapping of two group operations as the difference of the time when the last field operation of the former group operation ends execution minus the time when the first of the latter starts.

Table 4 shows the overlapping for doubling after addition. Overlapping decreases as the speed and the number of multipliers increase. Similar behaviors have been observed in the other cases (doubling after doubling and doubling after addition). This decrease is due to the increase of the parallelism in the tail of the operation.

6.3 Register Allocation

Schedulers fall into two broad families: unconstrained or resource limited. We choose to upper bound the number of resources as for busses and arithmetic units, while that of registers is unbounded. Once the operations are scheduled, we count the number of live registers and compute the register allocation. This option is the simplest solution and gives good results. In fact, each register stores a field element of 81 bits. Simulations demonstrate that changing the other parameters of the system has a low impact on the number of required registers. The system needs 18 and 20 registers in the best and worst case, respectively.

If a designer wants to lower the number of required registers, he can trade the number of registers for additional latency. In order to do so, he should avoid overlapping and start a group operation only when the previous has finished. We noted that a single group operation uses from 8 to 10 registers, while the maximum register number is reached when two group operations overlap.

Table 4. Overlapping in clock cycles of doubling after addition.

Digit-size	Number of multipliers			
	1	2	3	4
2	333	172	169	140
4	173	92	89	80
8	93	48	48	48
16	50	30	33	33

6.4 Evaluation of Different Architecture Options

In this section we report and compare the different architectures in terms of latency and area. In order to do so, we listed the use of the different multipliers as a percentage of the total time of one group operation. Thus, the figures shown in Table 5 and 6 are computed as follows: $\frac{\#_{mul} \cdot t_{mul}}{t_{group}}$, where $\#_{mul}$ is the number of multiplications executed by one multiplier unit during one group operation, t_{mul} is the time needed for one multiplication and t_{group} is the total execution time of the group operation.

In an ideal scenario all the multipliers should be used uniformly. However, one can see that the fourth multiplier, and in some cases also the third multiplier, are used very infrequently (see Table 6, the columns corresponding to the 3^{rd} and 4^{th} multiplier). Hence, for most applications it will be unreasonable to provide this extra hardware units.

In Table 8 we show a comparison to find the optimal architecture. The optimal implementation will achieve the highest throughput consuming the smallest area (contrary to some traditional cryptographic implementations, where only best performance was evaluated). The analysis uses the normalized area-time product (with respect to the lowest area-time product). Table 8 shows that the architecture using one inversion, one multiplication ($D = 8$), one addition and one squaring achieves the best area-time product.

To evaluate the latency of a complete scalar multiplication kD, reported in Table 7, we examined it in an average case, where the integer k of 160 bits has half of its bits equal to 1 and the rest equal to 0. This means that 80 and 160 additions and doublings were performed, respectively. Half of the 160 doublings are computed after another doubling and half after an addition.

We supposed that the all different configurations of the system work always at the same frequency. This is a worst-case assumption. In fact, usually the multiplier unit dominates the frequency, and a smaller digit-size will yield to higher clock frequency and thus speed-up the system.

It should be noted that we omitted the register file area in the estimation. We decided to do so after noting that the required number of registers is almost the same in all the configurations, and that the area consumed by a register can vary depending on the implementation technology.

Table 5. Use of the multiplier as a percentage of the total time of group addition.

Digit-size	Number of multipliers			
	1	2	3	4
2	68.3 %	-	-	-
	61.0 %	55.4 %	-	-
	55.5 %	43.2 %	30.8 %	-
	51.6 %	45.1 %	25.8 %	12.9 %
4	59.6 %	-	-	-
	48.2 %	43.8 %	-	-
	42.5 %	33.1 %	23.6 %	-
	38.6 %	33.7 %	19.3 %	9.6 %
8	48.2 %	-	-	-
	34.6 %	31.5 %	-	-
	32.8 %	22.9 %	13.1 %	-
	32.8 %	19.7 %	13.1 %	3.2 %
16	35.3 %	-	-	-
	22.8 %	20.7 %	-	-
	22.9 %	16.6 %	4.1 %	-
	22.9 %	16.6 %	4.1 %	0 %

Table 6. Use of the multiplier as a percentage of the total time of group doubling.

Digit-size	Number of multipliers			
	1	2	3	4
2	50.9 %	-	-	-
	42.1 %	33.7 %	-	-
	35.8 %	35.8 %	8.9 %	-
	35.8 %	35.8 %	8.9 %	0 %
4	40.7 %	-	-	-
	30.3 %	24.2 %	-	-
	24.8 %	24.8 %	6.2 %	-
	24.8 %	24.8 %	6.2 %	0 %
8	29.6 %	-	-	-
	19.7 %	15.8 %	-	-
	15.8 %	15.8 %	3.9 %	-
	15.8 %	15.8 %	3.9 %	0 %
16	19.7 %	-	-	-
	12 %	9.6 %	-	-
	12 %	9.6 %	0 %	-
	12 %	9.6 %	0 %	0 %

Table 7. Latency estimation in clock cycles.

Digit-size	Number of multipliers			
	1	2	3	4
2	165987	113948	100345	99836
4	106527	80228	74625	74136
8	76797	63524	61844	61844
16	62969	56216	56139	56139

Table 8. Area-time product.

Digit-size	Number of multipliers			
	1	2	3	4
2	1.3552	1.1148	1.1442	1.3000
4	1.0422	1.0447	1.2133	1.4454
8	1	1.2385	1.6063	2.0067
16	1.2277	1.8241	2.5487	3.2758

7 Conclusions and Further Research

We proposed for the first time an architecture for a HECC coprocessor using the recently developed explicit formulae for the group operations. Different options for the architecture were evaluated. These options differ in the kind (various digit-sizes) and number of multipliers.

We found out that if resources are unbounded, the group addition and doubling operations of HECC execute in 288 and 248 clock cycles, respectively. However, we noted that using over three multipliers does not help significantly, because additional multiplication units have very low utilization rates. For a realistic scenario the architecture using one inverter, one multiplier (with $D = 8$), one adder and one squarer achieves the best area-time product. In addition, we tested the possibility to overlap group operations. In the case of doubling after addition, we could compute these operations for 333 clock cycles in parallel. Finally we also analyzed the usage of registers, resulting in the necessity of 19 registers of 81 bit each.

In the future one should address the following: 1) usage of different policies to schedule group operations; 2) implement the register file by means of a conventional 32-bit; 3) use inversion free formulae HEC group operations, which could reach higher degree of parallelism; 4) implementation using a FPGA, to examine the impact of the critical path of the operators on the throughput of the system. Hopefully our findings are of interest for the research community as well as for industry.

References

[1] A. J. Menezes and Y. H. Wu, and R. J. Zuccherato. An Elementary Introduction to Hyperelliptic Curves. Personal correspondence, November 1996.

[2] A. Antola, G. Bertoni, L. Breveglieri, and P. Maistri. Parallel Architectures for Elliptic Curve Crypto-Processors over Binary Extension Fields. In *IEEE Midwest symposium on circuit and system 03 — MWSCS03*, 2003.

[3] M. Bednara, M. Daldrup, J. Shokrollahi, J. Teich, and J. von zur Gathen. Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. In *The 9th Reconfigurable Architectures Workshop (RAW-02)*, 2002.

[4] N. Boston, T. Clancy, Y. Liow, and J. Webster. Genus Two Hyperelliptic Curve Coprocessor. In J. c. K. K. B. S. Kaliski and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2002*, volume LNCS 2523, pages 529–539. Springer-Verlag, 2002.

[5] H. Brunner, A. Curiger, and M. Hofstetter. On Computing Multiplicative Inverses in GF(2^m). *IEEE Transactions on Computers*, 42:1010–1015, August 1993.

[6] D. Cantor. Computing in Jacobian of a Hyperelliptic Curve. In *Mathematics of Computation*, volume 48(177), pages 95 – 101, January 1987.

[7] P. Gaudry and R. Harley. Counting Points on Hyperelliptic Curves over Finite Fields. In W. Bosma, editor, *ANTS IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 297 – 312, Berlin, 2000. Springer Verlag.

[8] R. Govindaraian. *Instruction scheduling*. CRC Press, the compiler design handbook edition, 2003.

[9] N. Koblitz. *Algebraic Aspects of Cryptography*. Springer-Verlag, Berlin, Germany, first edition, 1998.

[10] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsujii. Fast Genus Three Hyperelliptic CurveCryptosystems. In *The 2002 Symposium on Cryptography and Information Security, Japan — SCIS 2002*, Jan.29-Feb.1 2002.

[11] T. Lange. Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae. Cryptology ePrint Archive, Report 2002/121, 2002. http://eprint.iacr.org/.

[12] T. Lange. Inversion-Free Arithmetic on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/147, 2002. http:eprint.iacr.org.

[13] T. Lange. Weighted Coordinates on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/153, 2002. http:eprint.iacr.org.

[14] T. Lange. Formulae for Arithmetic on Genus 2 Hyperelliptic Curves, 2003. Available at http://www.ruhr-uni-bochum.de/itsc/tanja/preprints.html.

[15] K. Matsuo, J. Chao, and S. Tsujii. Fast Genus Two Hyperelliptic Curve Cryptosystems. In *ISEC2001-31, IEICE*, 2001.

[16] P. K. Mishra and P. Sarkar. Parallelizing Explicit Formula for Arithmetic in the Jacobian of Hyperelliptic Curves. In *Advances in Cryptology — Asiacrypt 2003*, volume LNCS. Springer-Verlag, 2003.

[17] Y. Miyamoto, H. Doi, K. Matsuo, J. Chao, and S. Tsuji. A Fast Addition Algorithm of Genus Two Hyperelliptic Curve. In *The 2002 Symposium on Cryptography and Information Security — SCIS 2002, IEICE Japan*, pages 497 – 502, 2002. in Japanese.

[18] D. Mumford. Tata lectures on theta II. In *Prog. Math.*, volume 43. Birkhäuser, 1984.

[19] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS 1965. Springer-Verlag, 2000.

[20] J. Pelzl. Hyperelliptic Cryptosystems on Embedded Microprocessor. Master's thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universitaet Bochum, Bochum, Germany, Setember 2002.

[21] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves. In Ç. K. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*. Springer-Verlag, 2003.

[22] J. Pelzl, T. Wollinger, and C. Paar. High performance arithmetic for hyperelliptic curve cryptosystems of genus two. Cryptology ePrint Archive, Report 2003/212, 2003. `http://eprint.iacr.org/`.

[23] J. Pelzl, T. Wollinger, and C. Paar. Low Cost Security: Explicit Formulae for Genus-4 Hyperelliptic Curves. In *Tenth Annual Workshop on Selected Areas in Cryptography — SAC 2003*. Springer-Verlag, 2003.

[24] Y. Sakai and K. Sakurai. On the Practical Performance of Hyperelliptic Curve Cryptosystems in Software Implementation. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E83-A NO.4, pages 692 – 703, April 2000. IEICE Trans.

[25] Y. Sakai, K. Sakurai, and H. Ishizuka. Secure Hyperelliptic Cryptosystems and their Performance. In *Public Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 164 – 181, Berlin, 1998. Springer-Verlag.

[26] V. Shoup. NTL: A libary for doing Number Theory (version 5.0c), 2001. http://www.shoup.net/ntl/index.html.

[27] N. Smart. On the performance of hyperelliptic cryptosystems. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume LNCS 1592, pages 165–175. Springer-Verlag, 1999.

[28] L. Song and K. K. Parhi. Low-energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing Systems*, 2(22):1–17, 1997.

[29] M. Takahashi. Improving Harley Algorithms for Jacobians of Genus 2 Hyperelliptic Curves. In *SCIS, IEICE Japan*, 2002. in Japanese.

[30] N. Thériault. Index calculus attack for hyperelliptic curves of small genus. In *Advances in Cryptology - ASIACRYPT '03*, Berlin, 2003. Springer Verlag. LNCS.

[31] T. Wollinger. Computer Architectures for Cryptosystems Based on Hyperelliptic Curves. Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, May 2001.

[32] T. Wollinger and C. Paar. Hardware Architectures proposed for Cryptosystems Based on Hyperelliptic Curves. In *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems - ICECS 2002*, volume III, pages 1159 – 1163, September 15-18 2002.

[33] T. Wollinger, J. Pelzl, V. Wittelsberger, C. Paar, G. Saldamli, and Ç. K. Koç. Elliptic & hyperelliptic curves on embedded μp. *ACM Transactions in Embedded Computing Systems (TECS)*, 2003. Special Issue on Embedded Systems and Security.