Optimal Tower Fields for Hyperelliptic Curve Cryptosystems

Selçuk Baktır, Jan Pelzl, Thomas Wollinger, Berk Sunar, and Christof Paar

***Abstract*— Cryptographic primitives have increasingly emerged into embedded systems such as mobile phones, smart cards, and personal digital assistants. Elliptic Curve Cryptosystems (ECC) and Hyperelliptic curve cryptosystems (HECC) are the cryptosystems of choice for asymmetric data encryption in environments where processor power and storage are limited [1]. We introduce the first cryptographic implementation of Optimal Tower Fields (OTF) [2], [3] for HECC. Furthermore, we introduce the first implementation of HECC over an extension field of odd characteristic on an embedded processor. With our implementation, a scalar multiplication for a 160 bit group order can be performed in $44ms$ on the ARM processor which is 57% faster than the best previously known implementation on the same processor. Our implementations also target a general purpose processor.**

Keywords: optimal tower field, hyperelliptic curve cryptosystem, efficient implementation, embedded system, cryptographic application.

I. Introduction

Asymmetric cryptosystems based on hyperelliptic curve cryptosystem (HECC) are perfectly suited for devices where computational power and disposal memory are constrained [1]. Consequently, a lot of work has been done on improving hyperelliptic curve cryptosystems (see [4], [5] for a summary).

It is widely accepted that for HECC one needs a group order of size at least $\approx 2^{160}$. Thus, for HECC over $\mathbb{F}_q$ we will need at least $g \cdot \log_2 q \approx 2^{160}$, where g is the genus of the curve. In particular, for a curve of genus two and three, we will need a field $\mathbb{F}_q$ with $p \approx 2^{80}$ and $p \approx 2^{56}$, respectively. Therefore, the field arithmetic has to be performed using 80-bit and 56-bit long operands. The overall performance of HECC is directly related to its underlying field by means of the field arithmetic. Note that the necessary bit lengths for the operands are still large compared to the word size of embedded processors.

Optimal Extension Fields (OEF) [6], Processor Adequate Finite Fields (PAFF) [7], and Optimal Tower Fields (OTF) [2], [3] present techniques to adjust the computational effort to the processor. OTFs seem to be particularly suited for processor adapted implementations. The use of pseudo-Mersenne primes as field characteristic and the selection of binomials as field extension polynomials allow for very efficient field arithmetic. Moreover, the inversion operation has a low complexity and can be performed as fast as a single field multiplication depending on the particular OTF used.

S. Baktır and B. Sunar are with the Worcester Polytechnic Institute, Electrical and Computer Engineering Department, 100 Institute Rd., Worcester, MA 01609. E-mail: {selcuk, sunar}@wpi.edu.

J. Pelzl, T. Wollinger, and C. Paar are with the Department of Electrical Engineering and Information Sciences, Communication Security Group(COSY), Ruhr-Universität Bochum, Universitätsstrasse 150, 44780 Bochum, Germany. E-mail: {pelzl, wollinger, cpaar}@crypto.ruhr-uni-bochum.de.

Our Main Contributions

This contribution provides for the first time an implementation of a cryptographic application using OTFs. The resulting performance indicates clearly the advantages and the practical relevance of the use of OTFs for modern cryptosystems on different processors.

- The work at hand provides the first implementation of HECC over an extension field of odd characteristic on an embedded processor.
- We analyze our implementations on a general purpose processor (Pentium 4) and an embedded processor (ARM7TDMI) and put them into prospective with previous work.
- We investigated the performance of a low power implementation, using the ARM Thumb mode.
- We examined the impact of assembly driven optimizations of the implementations on the ARM processor as well as on the Pentium 4 processor.

II. Previous Work

A. Software Implementation of HECC

Hyperelliptic curve cryptosystems were first suggested for cryptographic use in 1988 [8] and it took almost 10 years until they were implemented. Relevant software implementations on general purpose machines can be found in [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [5]. The first two contributions listed implemented Cantor's algorithm with polynomial arithmetic, whereas the others used explicit formulae. For a detailed overview of HECC implementations the reader is referred to [1], [5].

Aware of several practical advantages, the research community recently implemented HECC on embedded processors using characteristic two fields. The authors in [20], [17], [18] provided results of relevant implementations on the ARM microprocessor for genus-2, genus-3, and genus-4 curves, respectively.

We are not aware of any publications considering an implementation of HECC over an extension field of odd characteristic on an embedded processor. Due to their high complexity, GF(p) and GF(p^n) implementations of HECC have not been accomplished on embedded devices. With the use of OTF, this contribution is the first to address an implementation of HECC over fields of odd characteristic on an embedded microprocessor.

III. Mathematical Background

A. OTF

Optimal Tower Fields were introduced and suggested for use in elliptic curve cryptography in 2003 [2]. In this section we briefly introduce OTFs. For the theory of optimal tower fields and further details of efficient implementation options of OTF arithmetic, the interested reader is referred to [2], [3].

A field obtained by repeatedly extending a ground field with a series of irreducible polynomials of the same degree is called a tower field. *Optimal Tower Fields* are a special class of tower fields defined as follows

Definition 1: An Optimal Tower Field (OTF) is a finite field $GF(q^{t^k})$ such that

1) q is a pseudo-Mersenne prime,
2) $GF(q^{t^k})$ is constructed by an ensemble of binomials $P_i(x) = x^t - \alpha_{i-1}$ irreducible over $GF(q^{t^{i-1}})$ with $P_i(\alpha_i) = 0$, for $0 < i \leq k$.

OEF arithmetic can be utilized for doing OTF arithmetic operations. Moreover, the special structure of OTFs allows particularly very fast inversion by utilizing the OTF recursive direct inversion technique [2], [3]. By using OTF inversion, an inversion in $GF(q^{t^k})$ can be reduced to a single inversion in the ground field $GF(q)$ in addition to some multiplications and additions in the subfields of $GF(q^{t^k})$.

In our implementation, we applied OTF inversion to OTFs of the form $GF(q^{2^2})$ and $GF(q^{2^3})$. We performed the single inversion in $GF(q)$ by using either table lookup or the binary extended Euclidean algorithm (EEA). In [2] the theoretical complexity of an inversion using the OTF recursive direct inversion technique is shown to be in the order of a single multiplication, which is also verified by our implementation results.

B. HECC

We only present a brief introduction to the theory of hyperelliptic curves and refer the reader to [21], [22] for more details.

Let $\mathbb{F}$ be a finite field and $\overline{\mathbb{F}}$ be the algebraic closure of $\mathbb{F}$. A hyperelliptic curve C of genus $g \geq 1$ over the field $\mathbb{F}$ is defined as the following equation:

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

The solutions $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ are points which satisfy the equation C and the partial derivative equations $2y + h(x) = 0$ and $h'(x)y - f'(x) = 0$.

The Jacobian of C over $\mathbb{F}$, denoted by $\mathbb{J}_C(\mathbb{F})$, is a divisor $D = \sum m_i P_i$ that is defined over $\mathbb{F}$ if $D^\sigma = \sum m_i P_i^\sigma$ is equal to D for all automorphisms σ of $\overline{\mathbb{F}}$ over $\mathbb{F}$ [23]. Each element of the Jacobian can be represented uniquely by a reduced divisor [24], [25]. This divisor can be represented as a pair of polynomials $u(x)$ and $v(x)$, where the coefficients of $u(x)$ and $v(x)$ are elements of $\mathbb{F}$ [26].

The algorithms used for adding and doubling divisors on $\mathbb{J}_C(\mathbb{F})$ were introduced by Cantor [25]. These group operations are performed in two steps: composition and reduction. Cantor's group operation can be written explicitly, resulting in more efficient arithmetic [27]. Since the publication of the explicit formulae a major effort of the research community has been the optimization of all the formulae. A chronological overview of the improvements, the computational complexity, and the corresponding references can be found in [4], [5].

In this contribution the implementation is based on the explicit formulae for genus 2 and 3 presented in [20], [16] and [17], [15], respectively.

IV. Methodology

1) Selection of tower fields: we chose the underlying fields (security levels of 160, 192, and 240 bits) for genus-2 and genus-3 curves. We focused on fields with characteristic smaller than 32-bits for efficiency. Table IV (Appendix) provides an overview of all the fields used for implementation.
2) Programming of the field arithmetic.
3) Implementation of HECC in NTL [28]: The main purpose of this implementation was to get test vectors to check our final implementation. Microsoft's *Visual C++ Compiler 6.0* and *Developer Studio 6* were used for compilation and debugging on the Pentium 4 processor.
4) Implementation of HECC on Pentium: Using the field operations we could program the group addition, group doubling, and finally the scalar multiplication for the Pentium processor.
5) Adjusting the code for the ARM microprocessor: We used our Pentium implementation, ported it onto ARM7TDMI@80MHz (ARMulator) and timed it. We used *ARM Developer Suite 1.2* as programming and debugging platform. The *ARM Compiler 1.2* was used to generate executable images for the microprocessor.
6) Analysis of the efficiency of a low power implementation on ARM (Thumb mode).
7) Optimization of our implementation for the ARM and Pentium: The bottleneck of our implementation is the field multiplication routine. Therefore, we implemented this routine for one designated field in assembly to accelerate the HECC.

V. Results

A. Field Arithmetic

The core routines of our HECC implementation are the field operations, namely multiplication, squaring, inversion and addition. The efficiency of these operations are in a large extent dependent on the particular field structure. We timed the field multiplication and inversion on the ARM microprocessor for fields of different structures (see Table I).

TABLE I. Timings of the Field Library on the ARM80MHz

Fieldsize	Field	Polynomial	Multiplication in μs	Inversion in μs		
				LUT	Euclid	Fermat
2^{56}	$GF(p^4)$ with $p = 2^{14} - 3$	$p(x) = x^4 - 2$	6.4	6.4	12.8	-
2^{64}	$GF(p^8)$ with $p = 2^8 + 1$	$p(x) = x^8 - 3$	9.7	16.0	-	-
2^{64}	$GF(p^4)$ with $p = 2^{16} + 1$	$p(x) = x^4 - 3$	6.4	-	9.6	9.6
2^{80}	$GF(p^8)$ with $p = 2^{10} - 3$	$p(x) = x^8 - 2$	12.8	16.0	-	-
2^{80}	$GF(p^4)$ with $p = 2^{20} - 3$	$p(x) = x^4 - 2$	6.0	-	19.2	25.6
2^{96}	$GF(p^8)$ with $p = 2^{12} - 3$	$p(x) = x^8 - 2$	12.8	16.0	-	-
2^{96}	$GF(p^4)$ with $p = 2^{24} - 3$	$p(x) = x^4 - 2$	6.0	-	19.2	28.8
2^{112}	$GF(p^8)$ with $p = 2^{14} - 3$	$p(x) = x^8 - 2$	12.8	16.0	-	-
2^{116}	$GF(p^4)$ with $p = 2^{29} - 3$	$p(x) = x^4 - 2$	6.0	-	22.4	28.8

1) Field Inversion: For the field inversion, we used the OTF recursive direct inversion technique. We investigated three different ways to compute the single ground field $GF(p)$ inverse required for performing an OTF inversion: precomputation and lookup table (LUT), binary extended Euclidean algorithm (EEA) and Fermat's method. For small values of p, this inversion can be done very fast by LUTs. EEA and Fermat's method can be used for large p values, where inversion by LUT is not practical.

According to our timings, the binary EEA turns out to be the faster than Fermat's method for ground field $GF(p)$ inversion, therefore it will be used in our implementation for p larger than 2^{14}.

2) Field Multiplication: Like the other field arithmetic operations, the timings of the field multiplication depend on the field characteristics and the field generating binomials. Naturally, a binomial of higher degree (e.g. 8 in our case) leads to a larger number of processor word multiplications than with lower degree (e.g. 4 in our case). Table I shows that multiplications over fields with higher degree binomials are more than twice as slow. Hence, for the implementation of HECC only fields of low degree binomials will be used.

Note that an OTF inversion takes between a factor of 1.25 and 4.75 more time than a field multiplication. Therefore, HECC with affine coordinates using OTFs is preferable.

B. Scalar Multiplication of HECC

The main operation in HECC is the scalar multiplication of a divisor and is realized by the sliding window method ($w = 4$). Table II provides timings for a scalar multiplication on HECC over different OTFs for the ARM and Pentium processors. Figure 1 depicts the performance of HECC of genus 2 and 3 for equal group orders of length approximately 160bit and 190bit. The execution time of the scalar multiplication essentially depends on the bit length of the scalar ($\approx$ bit length of the group order).

TABLE II. TIMINGS FOR SCALAR MULTIPLICATIONS)

Genus	Fieldsize	Field	Polynomial	Time in ms	
				ARM@80MHz	P4@1.8GHz
2	2^{80}	$GF(p^4)$ with $p = 2^{20} - 3$	$p(x) = x^4 - 2$	44.2	2.24
	2^{96}	$GF(p^4)$ with $p = 2^{24} - 3$	$p(x) = x^4 - 2$	53.3	2.70
	2^{116}	$GF(p^4)$ with $p = 2^{29} - 3$	$p(x) = x^4 - 2$	63.3	3.34
3	2^{56}	$GF(p^4)$ with $p = 2^{14} - 3$	$p(x) = x^4 - 2$	95.0	3.87
	2^{64}	$GF(p^4)$ with $p = 2^{16} + 1$	$p(x) = x^4 - 3$	138.7	7.33
	2^{80}	$GF(p^4)$ with $p = 2^{20} - 3$	$p(x) = x^4 - 2$	183.3	9.37

Figure I clearly shows that HECC of genus 2 perform better than HECC of genus 3. For a security level of approximately 160bit, a scalar multiplication on the ARM processor takes $44.2ms$ for genus-2 HECC and $95ms$ for genus-3 HECC. With increasing group order the cryptosystems perform slower.

The fastest known implementation on the same processor was done in [20], [17]. Over characteristic two fields, a scalar multiplication was achieved in $69ms$ and $90ms$ on HECC of genus 2 and 3, respectively. Thus, for a comparable security level of 160bit, our implementation of HECC of genus 2 over OTF is 57% faster. Genus 3 HECC over OTF and characteristic two fields have equal performance, though the group operations on fields of odd characteristic are more complex [5].

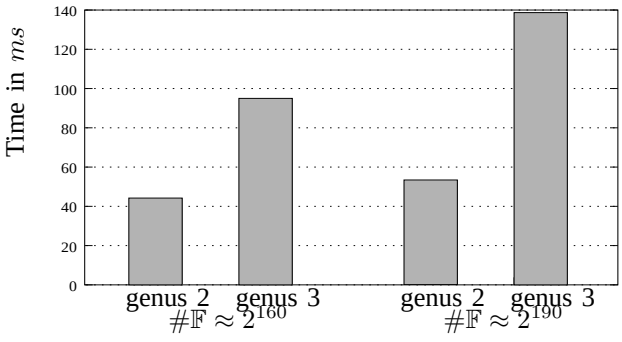

Fig. 1. Timings of Scalar Multiplication for Different Group Orders and Genera (ARM7TDMI@80MHz)

On the Pentium processor, a scalar multiplication (160bit security) takes $2.24ms$ and $3.87ms$ for the two genera. Whereas, the fastest known implementation targeting fields GF(p) take $1.72ms$ and $2.8ms$ and are presented in [19]. Note that the author in [19] used a different processor, namely an AMD Athlon with a clockrate of 1GHz. Note that we caution the reader to compare the two implementations, since the targeted processors have very different characteristics.

C. Optimized Implementation on the ARM and Pentium

In this section we analyze the impact of processor specific optimizations and also compile the code for ARM in the so called *Thumb mode*, which is an energy efficient 16-bit mode of the ARM microprocessor. Furthermore, we decided to optimize the multiplication routine for the ARM and Pentium processors in assembly.

1) Using the Energy and Space Efficient ARM Thumb Mode: The ARM Thumb mode solely uses 16-bit operations, resulting in a smaller code size and lower power consumption which are eminently important for embedded applications. Table III gives the timings for the Thumb mode. Compared to Table II, a slowdown factor of approximately 2 can be observed. However, in the case of our smallest sized OTF, there is a mere 14% slowdown compared to the 32-bit code.

TABLE III. TIMINGS FOR SCALAR MULTIPLICATIONS (ARM@80MHZ IN THUMB MODE)

Genus	Fieldsize	Field	Polynomial	Time in ms
2	2^{80}	$GF(p^4)$ with $p = 2^{20} - 3$	$p(x) = x^4 - 2$	98.9
	2^{96}	$GF(p^4)$ with $p = 2^{24} - 3$	$p(x) = x^4 - 2$	119.6
	2^{116}	$GF(p^4)$ with $p = 2^{29} - 3$	$p(x) = x^4 - 2$	147.2
3	2^{56}	$GF(p^4)$ with $p = 2^{14} - 3$	$p(x) = x^4 - 2$	107.7
	2^{64}	$GF(p^4)$ with $p = 2^{16} + 1$	$p(x) = x^4 - 3$	330.0
	2^{80}	$GF(p^4)$ with $p = 2^{20} - 3$	$p(x) = x^4 - 2$	415.2

Furthermore, instead of using the same OTFs that we used for the 32-bit code, we also can use OTFs with smaller characteristics, e.g. characteristics smaller than 16 bits in length. This yields to higher performance for the ARM Thumb mode. For example in genus-2 HECC over a field of order 2^{80}, we can gain approximately 13% in performance compared to lower extension fields, if we use the field GF(p^8) with $p = 2^{10} - 3$ and $p(x) = x^8 - 2$. For genus-3, a speed up of 15%

can be achieved by using a field $GF(p^8)$ with $p = 2^{14} - 3$ and $p(x) = x^8 - 2$. Table V (Appendix) includes the timing results in ARM Thumb mode for fields with small characteristic.

2) Assembly Optimization on ARM and Pentium: A further speed up can be achieved with the use of processor specific assembly instructions. Due to time restrictions we decided to analyze solely the effect of an optimized multiplication routine. The multiplication routine is the crucial function of this cryptosystem since a scalar multiplication is mainly computed with field multiplications. We optimized a multiplication routine for a fixed field size in assembly.

Our assembly code makes intensive use of the registers by trying to keep often used values in the registers, i.e., performing less memory accesses. By utilizing the multiply and accumulate routine (MLA, UMLAL) we could achieve a speed up of approximately 35% for a field of size 2^{80}. The whole cryptosystem (genus-2) gained 13% in performance with this simple optimization.

Since only four registers can be addressed directly on the Pentium platform, we had to arrange a different approach. For the Pentium platform, we optimized the multiplication over a field of size 2^{96}. The use of special operations from the MMX and SSE2 instruction set resulted in an acceleration of the genus-2 HECC by 27% (a scalar multiplication takes now $2.13ms$).

VI. Conclusions

Since the development of asymmetric cryptosystems based on elliptic and hyperelliptic curves, it has been a challenging task to implement ECC and HECC over fields of odd characteristic. With the advent of OEF, PAFF, and more recently OTF, the performance of ECC and HECC over prime (extension) fields increased drastically. For a fixed security level, OTFs offer different field extensions. Thus, the structure of the field can be varied and adapted to the processor word size which yields to an efficient field arithmetic.

Our implementation of HECC over OTF shows the practical relevance of OTF in cryptographic implementations. The arithmetic over OTFs allows for a very efficient implementation. On a typical 32-bit embedded microprocessor, namely the ARM7TDMI, a scalar multiplication for a 160-bit group order can be performed in $44.2ms$. Compared to the currently fastest implementation over fields of characteristic two, namely genus-2 HECC over GF(2^{81}) on the same processor [20], this is an improvement of approximately 57%. The implementation on the Pentium 4 processor computes a scalar multiplication over a group order of 160 bit in $2.13ms$.

Using the *Thumb Mode* of the ARM microprocessor, a 16-bit implementation of the cryptosystem was realized. This type of implementation is eminently important for power and memory limited environments. The performance of the 16-bit cryptosystem can be improved by changing the underlying field structure of the OTFs. A scalar multiplication of a genus-2 HECC with 160-bit group order can be done in $87.5ms$ in the ARM Thumb mode.

Furthermore, we investigated the influence of an assembly optimized multiplication routine on the efficiency of the whole cryptosystem for both the ARM and the Pentium microprocessor. On the ARM, we could gain approximately 35% in speed for the field multiplication routine compared to the implementation without assembly optimizations. Hence, the entire cryptosystem was accelerated by 13%. Using special MMX and SSE2 instructions, the cryptosystem on the Pentium 4 processor gained 27% in speed.

VII. Appendix

TABLE IV. OTFs and corresponding OEF Representations

Field size	OTF	Corresponding OEF representation $GF(p^m)$	
		p	field generating polynomial
2^{56}	$GF(q^{2^2})$ with $q = 2^{14} - 3$, and $\alpha_0 = 2$	$p = 2^{14} - 3$	$p(x) = x^4 - 2$
2^{64}	$GF(p^{2^3})$ with $q = 2^8 + 1$, and $\alpha_0 = 3$	$p = 2^8 + 1$	$p(x) = x^8 - 3$
2^{64}	$GF(q^{2^2})$ with $q = 2^{16} + 1$, and $\alpha_0 = 3$	$p = 2^{16} + 1$	$p(x) = x^4 - 3$
2^{80}	$GF(q^{2^3})$ with $q = 2^{10} - 3$, and $\alpha_0 = 2$	$p = 2^{10} - 3$	$p(x) = x^8 - 2$
2^{80}	$GF(q^{2^2})$ with $q = 2^{20} - 3$, and $\alpha_0 = 2$	$p = 2^{20} - 3$	$p(x) = x^4 - 2$
2^{96}	$GF(q^{2^3})$ with $q = 2^{12} - 3$, and $\alpha_0 = 2$	$p = 2^{12} - 3$	$p(x) = x^8 - 2$
2^{96}	$GF(q^{2^2})$ with $q = 2^{24} - 3$, and $\alpha_0 = 2$	$p = 2^{24} - 3$	$p(x) = x^4 - 2$
2^{112}	$GF(q^{2^3})$ with $q = 2^{14} - 3$, and $\alpha_0 = 2$	$p = 2^{14} - 3$	$p(x) = x^8 - 2$
2^{116}	$GF(q^{2^2})$ with $q = 2^{29} - 3$, and $\alpha_0 = 2$	$p = 2^{29} - 3$	$p(x) = x^4 - 2$

TABLE V. Timings for Scalar Multiplications over Fields of Smaller Characteristic (ARM@80MHz in Thumb Mode)

Genus	Fieldsize	Field	Polynomial	Time in ms
2	2^{80}	$GF(p^8)$ with $p = 2^{10} - 3$	$p(x) = x^8 - 2$	87.5
	2^{96}	$GF(p^8)$ with $p = 2^{12} - 3$	$p(x) = x^8 - 2$	106.3
	2^{112}	$GF(p^8)$ with $p = 2^{14} - 3$	$p(x) = x^8 - 2$	123.7
3	2^{64}	$GF(p^8)$ with $p = 2^8 + 1$	$p(x) = x^8 - 3$	280.4
	2^{80}	$GF(p^8)$ with $p = 2^{10} - 3$	$p(x) = x^8 - 2$	374.0

References

[1] 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.

[2] S. Baktir, "Efficient Algorithms for Finite Fields, with Applications in Elliptic Curve Cryptography," Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, May 2003.

[3] S. Baktır and B. Sunar, "Optimal Tower Fields," in *IEEE Transactions on Computers*, to appear.

[4] C. Paar, J. Pelzl, and T. Wollinger, "Hyperelliptic Curve Cryptosystems," March 2004. `http://www.hecc.rub.de`.

[5] T. Wollinger, J. Pelzl, and C. Paar, "Cantor versus Harley: Optimization and Analysis of Explicit Formulae for Hyperelliptic Curve Cryptosystem." `http://www.crypto.rub.de/Publikationen/texte/wollinger_etal_explicit_formulae_for_HECC.pdf`, February 2004. Technical Report.

[6] D. V. Bailey and C. Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms," in *Advances in Cryptology — CRYPTO '98* (H. Krawczyk, ed.), LNCS 1462, (Berlin, Germany), pp. 472–485, Springer-Verlag, 1998.

[7] R. M. Avanzi and R. Mihălescu, "Generic Efficient Arithmetic Algorithms for PAFFs (Processor Adequate Finite Fields) and Related Algebraic Structures," in *Tenth Annual Workshop on Selected Areas in Cryptography — SAC 2003*, LNCS, Springer-Verlag, 2003.

[8] N. Koblitz, "A Family of Jacobians Suitable for Discrete Log Cryptosystems," in *Advances in Cryptology - CRYPTO '88* (Shafi Goldwasser, ed.), LNCS 403, (Berlin), pp. 94 – 99, Springer-Verlag, 1988.

[9] Y. Sakai, K. Sakurai, and H. Ishizuka, "Secure Hyperelliptic Cryptosystems and their Performance," in *Public Key Cryptography: First International Workshop on Practice and Theory in Public Key Cryptography — PKC'98* (H. Imai and Y. Zheng, eds.), LNCS 1431, (Berlin), pp. 164 – 181, Springer-Verlag, 1998.

[10] 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*, vol. E83-A NO.4, pp. 692 – 703, April 2000. IEICE Trans.

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

[12] 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*, pp. 497 – 502, 2002. in Japanese.

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

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

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

[16] T. Lange, "Formulae for Arithmetic on Genus 2 Hyperelliptic Curves," September 2003. `http://www.ruhr-uni-bochum.de/itsc/tanja/preprints/expl_sub.pdf`.

[17] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar, "Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves," in *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003* (C. D. Walter, Ç. K. Koç, and C. Paar, eds.), pp. 349 – 365, Springer-Verlag, September 2003. LNCS 2779.

[18] 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. LNCS.

[19] R. M. Avanzi, "Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations." Cryptology ePrint Archive, Report 2003/253, 2003. `http://eprint.iacr.org/`.

[20] J. Pelzl, T. Wollinger, and C. Paar, "High Performance Arithmetic for Hyperelliptic Curve Cryptosystems of Genus Two," in *International Conference on Information Technology: Coding and Computing - ITCC 2004*, IEEE Computer Society, April 5-7 2004.

[21] N. Koblitz, "Hyperelliptic cryptosystems," *Journal of Cryptology*, vol. 1, no. 3, pp. 129–150, 1989.

[22] N. Koblitz, *Algebraic Aspects of Cryptography*. Berlin, Germany: Springer-Verlag, fi rst ed., 1998.

[23] A. Menezes, Y. Wu, and R. Zuccherato, *An Elementary Introduction to Hyperelliptic Curves*. Berlin, Germany: Springer-Verlag, fi rst ed., 1998. In: Koblitz, N., Algebraic Aspects of Cryptography, Springer-Verlag Berlin, 1998.

[24] W. Fulton, *Algebraic Curves - An Introduction to Algebraic Geometry*. Reading, Massachusetts: W. A. Benjamin, Inc., 1969.

[25] D. Cantor, "Computing in Jacobian of a Hyperelliptic Curve," in *Mathematics of Computation*, vol. 48(177), pp. 95 – 101, January 1987.

[26] D. Mumford, "Tata lectures on theta II," in *Prog. Math.*, vol. 43, Birkhäuser, 1984.

[27] P. Gaudry and R. Harley, "Counting Points on Hyperelliptic Curves over Finite Fields," in *ANTS IV* (W. Bosma, ed.), LNCS 1838, (Berlin), pp. 297 – 312, Springer Verlag, 2000.

[28] V. Shoup, "NTL: A libary for doing Number Theory (version 5.2)," 2003. `http://www.shoup.net/ntl/index.html`.