Elliptic and Hyperelliptic Curves on Embedded μP

THOMAS WOLLINGER, JAN PELZL, VOLKER WITTELSBERGER, and CHRISTOF PAAR
University of Bochum
and
GÖKAY SALDAMLI and ÇETIN K. KOÇ
Oregon State University

It is widely recognized that data security will play a central role in future IT systems. Providing public-key cryptographic primitives, which are the core tools for security, is often difficult on embedded processor due to computational, memory, and power constraints. This contribution appears to be the first thorough comparison of two public-key families, namely elliptic curve (ECC) and hyperelliptic curve cryptosystems on a wide range of embedded processor types (ARM, ColdFire, PowerPC). We investigated the influence of the processor type, resources, and architecture regarding throughput. Further, we improved previously known HECC algorithms resulting in a more efficient arithmetic.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Data communications, Security and protection*; C.3 [**Special-Purpose and Application-based Systems (J.7)**]: Microprocessor/Microcomputer Applications, Real-Time and Embedded Systems, Signal Processing Systems; C.5.3 [**Computer System Implementation**]: Microcomputers; Microprocessors; E.3 [**Data Encryption**]: Public-key Cryptosystems; F.2.0 [**Analysis of Algorithms and Problem Complexity (B.6-7, F.1.3)**]: General

General Terms: Algorithms, Design, Performance, Security

Additional Key Words and Phrases: Elliptic curves cryptosystem, hyperelliptic curve cryptosystem, implementation

1. INTRODUCTION

It is widely recognized that data security will play a central role in the majority of future IT systems. Many of these future IT applications will be realized as embedded systems. A lot of those applications rely heavily on security

Authors' addresses: Thomas Wollinger, Jan Pelzl, Volker Wittelsberger and Christof Paar, Department of Electrical Engineering and Information Sciences, Communication Security Group (COSY), Ruhr-Universität Bochum, Universitätsstrasse 150, 44780 Bochum, Germany; email: {wollinger, pelzl,wittelsberger,cpaar}@crypto.ruhr-uni-bochum.de; Gökay Saldamli and Çetin K. Koç, ECE Department, 220 Corvallis, Oregon 97331, USA; email: {saldamli,koc}@ece.orst.edu.

mechanisms such as security for wireless phones, faxes, wireless computing, pay-TV, and copy protection schemes for audio/video consumer products as well as digital cinemas. Note that a large share of those embedded applications will be wireless. Wireless applications can be easily eavesdropped, which makes the communication channel especially vulnerable and the need for security even more obvious. In addition, authentication technologies are desired in order to prevent the threat of generating personal profiles when using the stated application. This merging of communications and computation functionality requires data processing in real time, and embedded systems have shown to provide appropriate solutions for many applications (e.g., cellular phones).

All modern security protocols, such as IPSec [Kent and Atkinson 1998], SSL [Freier et al. 1996], TLS [Dierks and Allen 1999] use symmetric-key algorithms as well as public-key algorithms. Providing highly arithmetic-intensive public-key cryptographic primitives in an embedded environment is often difficult due to the computational, memory, and power constraints. This contribution surveys the implementation of the two most promising public-key cryptosystems for embedded applications, namely ECC and HECC. The elliptic curve cryptosystem (ECC) was introduced in Koblitz [1987] and Miller [1986], and is based on the difficulty of the Diffie–Hellman (DHP) problem in the group of points of an elliptic curve over a finite field. The DH problem is closely related to the well-studied discrete logarithm (DL) problem. Since their introduction, ECC have been extensively studied not only by the research community but also in industry. In particular, there are several standards involving EC, such as the IEEE P1363 [IEEE 1999] standardization effort. Hyperelliptic curve cryptosystems (HECC) were first suggested in 1988 by Koblitz [1988]. In contrast to ECC, it has only been until recently that Koblitz's idea to use HEC for cryptographic applications, has been analyzed and implemented both in software [Krieger 1997; Sakai et al. 1998; Smart 1999; Sakai and Sakurai 2000; Matsuo et al. 2001; Miyamoto et al. 2002; Kuroki et al. 2002; Lange 2002a; Pelzl 2002] and in more hardware-oriented platforms such as FPGAs [Wollinger 2001; Wollinger and Paar 2002; Boston et al. 2002].

It is important to point out that ECCs and HECCs seem to be specially promising for the use in embedded environments where memory and speed is constrained. The suitability for constrained systems results from the *short* operand sizes of ECC and HECC compared to other public-key schemes, for example, RSA- [Rivest et al. 1978] or DL-based systems. It is widely accepted that for most cryptographic applications based on EC or HEC the necessary group order is 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. Therefore, we will need a field order $q \approx 2^{40}$ for genus 4, $q \approx 2^{54}$ for genus 3, and $q \approx 2^{80}$ for genus 2 HEC. Hence, one needs 40-bit to 80-bit long operands to compute the group operations for these curves. In the case of ECC we have to work with operand lengths of approximately 160 bits. Whereas in the case of RSA, the operands will be approximately 1024 bits in order to achieve the same security. It is widely believed that HECC is less efficient, because of the complex structure of the group operations. Further, until now there was no detailed analysis of the efficiency of these cryptosystems on embedded processors.

Implementations of certain cryptosystems always are application depend. Imagine a scenario, where a number of PDAs communicate with a server over a secure channel. Each of the PDAs uses a different cryptographic primitive (algorithm, curve, field polynomial, and so on). Therefore, the cryptographic engine running on a server has to support a whole suite of cryptographic algorithms, whereas each algorithm has to cope different input parameters. In contrast, the implementations on constrained platforms (like the PDA) normally need only one cryptographic algorithm with a fixed set of input parameters. Hence, implementations with fixed parameters are attractive for embedded applications and those with flexible parameters for systems with fewer constraints such as servers.

We address the following questions in our contribution:

—How well do ECC and HECC perform on the most common embedded platforms (ARM, ColdFire, and PowerPC)?
—How can we decrease the complexity of the genus-3 HECC group operations?
—How do our improvements on HECC influence the performance compared to ECC?
—What is the influence of the available resources of the board on the performance?
—How well does an implementation targeted for cryptosystems with fixed parameters perform versus a system that is designed to handle different parameters?

This appears to be the first thorough comparison of ECC and HECC taking latest advances in HECC implementation techniques into account. We improved previously known HECC algorithms resulting in more efficient arithmetic operations. We were able to achieve a highly competitive throughput for the cryptosystems implemented on a wide range of important embedded platforms. The best timings for the scalar multiplication for HEC cryptosystems could be achieved on the PowerPC running at 50 MHz, resulting in 117 and 84.9 ms for genus 2 and 3, respectively. The scalar multiplication for ECC could be performed fastest on the same platform in 106.3 ms. Our highly optimized formulae for HECC allow (contrary to common believe) the same throughput than ECC and further, in some cases HECC outperformed ECC. We showed that for the two algorithm types implemented, the instruction cache on the PowerPC had a fundamental influence regarding the speed of one scalar multiplication. The time needed to perform one scalar multiplication can be decreased by more than a factor of 3 when using the instruction cache, and by almost a factor of 8 when using instruction as well as data cache. In addition, we could speed up the throughput of the HEC scalar multiplication by up to 50% by focusing on a fixed underlying field and curve, which is the most likely scenario for implementations on embedded systems. Combining all of these results, we showed that both families of algorithms are well suited for embedded applications.

The remainder of the paper is organized as follows. Section 2 summarizes contributions dealing with previous implementations of HECC and ECC.

Section 3 gives a brief overview of the mathematical background related to both cryptosystems. Section 4 presents the software and hardware methodology used for this publication and Section 5 introduces the arithmetic of ECC and HECC. Finally, we end this contribution with a discussion of our results and some conclusions.

2. PREVIOUS WORK

Although ECC and HECC were proposed in late 1980s, HECC has so far failed to flourish to the same extent as ECC. This is unfortunate, as HECC has the potential for much smaller operand lengths. Indeed, the number of points on a Jacobian of a curve of genus g over a finite field of q elements is roughly q^g. There is thus the hope of secure HECC with $g > 1$ having operands and arithmetic related to fields considerably smaller than in the elliptic curve setting, where $g = 1$. In this section, we present some results from previous state-of-the-art implementations.

2.1 Implementation of ECC

Many research results in both software and hardware deal with realizations of ECC. The high-level operations of ECC are mostly standard, which are described in standard bodies like IEEE P1363 [IEEE 1999], ANSI X9.62 [ANSI X9.62-1999 1999], and ANSI 9.63 [ANSI X9.63-199x 1998]. Moreover, one can find commercially and also publicly available software implementations of ECC.

Point addition can be performed by one field inversion, two multiplications, and one squaring (see Chapter 3). In cases where inversion is much more expensive than multiplication, EC point addition can be computed by using projective or Jacobian coordinates. Comparisons of the various types of coordinate systems can be found in Chudnovsky and Chudnovsky [1987] and Cohen et al. [1998].

EC point multiplication poses the exponentiation problem in abelian groups. Thus, any method for the general exponentiation problem can be applied to EC multiplication. These include: the binary, m-ary, and sliding window methods, methods based on signed digit representations [Morain and Olivos 1990], and combinations of these ideas. These methods are summarized in Gordon [1998] and Blake et al. [1999]. A comparison of implementation results can be reached through Guajardo and Paar [1997] and López and Dahab [1999]. An implementation of a different approach introduced by Montgomery [1987] can be found in López and Dahab [1999]. Fast ECC implementations are, for example, reported in Schroeppel et al. [1995], López and Dahab [1999], King [2001].

Moreover, there are some special classes of ECs, which allow for efficient implementations. For curves defined over small subfields, scalar multiplication can be significantly accelerated by using a Frobenius expansion. The efficient algorithms are presented in Solinas [1997].

For the most significant hardware implementations consider Agnew et al. [1993], Rosner [1999], Gao et al. [1999], Orlando and Paar [2000], and Gura et al. [2001].

Table I. Execution Times of Previous HEC Implementations in Software

	Processor	Genus	Field	$t_{\text{Scalarmult.}}$ ms
Krieger [1997]	Pentium @ 100 MHz	2	$\mathbb{F}_{2^{64}}$	520
		3	$\mathbb{F}_{2^{42}}$	1200
		4	$\mathbb{F}_{2^{31}}$	1100
Sakai et al. [1998]	Alpha @ 467 MHz	3	$\mathbb{F}_{2^{59}}$	83.3
		3	$\mathbb{F}_{2^{89}}$	25700
		3	$\mathbb{F}_{2^{113}}$	37900
		4	$\mathbb{F}_{2^{41}}$	96.6
	Pentium-II @ 300 MHz	3	$\mathbb{F}_{2^{59}}$	11700
		4	$\mathbb{F}_{2^{41}}$	10900
Sakai and Sakurai [2000]	Alpha21164A @ 600 MHz	3	$\mathbb{F}_p(\log_2 p = 60)$	98
		3	$\mathbb{F}_{2^{59}}$	40
		4	$\mathbb{F}_{2^{41}}$	43
Matsuo et al. [2001]	PentiumIII @ 866 MHz	2	$186-bit$OEF	1.98
Miyamoto et al. [2002]	PentiumIII @ 866 MHz	2	$186-bit$ OEF	1.69
Kuroki et al. [2002]	Alpha21264 @ 667 MHz	3	$\mathbb{F}_{2^{61}-1}$	0.932
Lange [2002a]	Pentium-IV @ 1.5 GHz	2	$\mathbb{F}_{2^{160}}$	18.875
		2	$\mathbb{F}_{2^{180}}$	25.215
		2	$\mathbb{F}_p(\log_2 p = 160)$	5.663
		2	$\mathbb{F}_p(\log_2 p = 180)$	8.162

2.2 Implementation of HECC

Since HECCs were proposed, there have been several software implementations on general-purpose machines [Krieger 1997; Sakai et al. 1998; Smart 1999; Sakai and Sakurai 2000; Matsuo et al. 2001; Miyamoto et al. 2002; Kuroki et al. 2002; Lange 2002a; Pelzl 2002] and hardware [Wollinger 2001; Wollinger and Paar 2002; Boston et al. 2002]. The results of previous HECC software implementations are summarized in Table I.[1] The first three contributions presented in the table implemented Cantor's algorithm, whereas the other publications used explicit formulae. We are not aware of any publications of HECC implementations on embedded processors.

3. MATHEMATICAL BACKGROUND

In this section, we briefly introduce the theory of ECC and HECC, restricting attention to material that is relevant for this work. We will consider curves over binary extension fields only.

More detail about ECC can be found in Koblitz [1987] and Miller [1986] and in standards [IEEE 1999; ANSI X9.62-1999 1999; ANSI X9.63-199x 1998]. The modern classic reference for the theory of ECCs is Silverman [1986]. The interested reader is referred to Menezes et al. [1996] and Koblitz [1988, 1989, 1998] for more background on HECC.

[1]Table I presents timings for curves of genus smaller than five.

3.1 Elliptic Curve Cryptosystem

An elliptic curve $E(\mathbb{F}_{2^m})$ over $\mathbb{F}_{2^m}$ is defined by parameters $a, b \in \mathbb{F}_{2^m}$ satisfying $b \neq 0$ and consists of the set of solutions, or points, $P = (x, y)$ for $x, y \in \mathbb{F}_{2^m}$ to the equation:

$$y^2 + xy \equiv x^3 + ax^2 + b$$

and the point at infinity O. The set of points on $\mathbb{F}_{2^m}$ forms an abelian group under the following addition rule.

Let $(x_1, y_1) \in \mathbb{F}_{2^m}$ and $(x_2, y_2) \in \mathbb{F}_{2^m}$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 + \lambda + x_1 + x_2 + a\ , \quad y_3 \equiv \lambda(x_1 + x_3) + x_3 + y_1, \quad \text{with } \lambda \equiv \frac{y_2 + y_1}{x_2 + x_1}$$

3.2 Hyperelliptic Curve Cryptosystem

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 solutions $(x, y) \in \mathbb{F} \times \mathbb{F}$ to the 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 is said to be nonsingular if there are no pairs $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$, which simultaneously satisfy the equation of the curve C and the partial differential equations $2v + h(x) = 0$ and $h'(x)v - f'(x) = 0$.

If we want to define the Jacobian over $\mathbb{F}$, denoted by $\mathbb{J}_C(\mathbb{F})$, we say that a divisor $D = \sum m_i P_i$ 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 $\mathbb{F}$ [Menezes et al. 1996].

Each element of the Jacobian can be represented uniquely by a reduced divisor [Fulton 1969; Cantor 1987]. This divisors can be represented as a pair of polynomials $u(x)$ and $v(x)$ with $\deg v(x) < \deg u(x) \leq g$, with $u(x)$ dividing $y^2 + h(x)y - f(x)$ and where the coefficients of $u(x)$ and $v(x)$ are elements of $\mathbb{F}$ [Mumford 1984, p. 3.17]. In the remainder of this paper, a divisor D represented by polynomials will be denoted by $div(u, v)$. Algorithm 1 describes the group addition of two divisors on $\mathbb{J}_C(\mathbb{F})$.

Algorithm 1 Group Addition

```
Require: $D_1 = \mathrm{div}(u_1, v_1)$, $D_2 = \mathrm{div}(u_2, v_2)$
Ensure: $D = \mathrm{div}(u, v) = D_1 + D_2$
 1: $d = \gcd(u_1, u_2, v_1 + v_2 + h) = s_1u_1 + s_2u_2 + s_3(v_1 + v_2 + h)$
 2: $u'_0 = u_1u_2/d^2$
 3: $v'_0 = [s_1u_1v_2 + s_2u_2v_1 + s_3(v_1v_2 + f)]d^{-1}(\mathrm{mod}\, u'_0)$
 4: $k = 1$
 5: repeat
 6:   $u'_k = \frac{f - v'_{k-1}h - (v'_{k-1})^2}{u'_{k-1}}$
 7:   $v'_k = (-h - v'_{k-1}) \bmod u'_k$
 8: until $\deg u'_k \leq g$
 9: Output $(u = u'_k, v = v'_k)$
```

Table II. Improvements of the Group Operations on Genus-2 HEC

	Field Characteristic	Cost	
		Addition	Doubling
Nagao [2000]	*general*	$3I + 70M/S$	$3I + 76M/S$
Nagao [2000]	*odd*	$1I + 55M/S$	$1I + 55M/S$
Harley [2000]	*odd*	$2I + 27M/S$	$2I + 30M/S$
Matsuo et al. [2001]	*odd*	$2I + 25M/S$	$2I + 27M/S$
Miyamoto et al. [2002]	*odd*	$I + 26M/S$	$I + 27M/S$
Takahashi [2002]	*odd*	$I + 25M/S$	$I + 29M/S$
Lange [2002a]	*general*	$I + 22M + 3S$	$I + 22M + 5S$
	two	$I + 22M + 2S$	$I + 20M + 4S$
Lange [2002b]	*general*	$47M + 4S(40M + 3S)^a$	$40M + 6S$
	two	$46M + 2S$	$33M + 6S$
Lange [2002c]	*odd*	$47M + 7S(36M + 5S)^a$	$34M + 7S$
	even, $h \neq 0$	$46M + 4S(35M + 5S)^a$	$35M + 6S$
	even, $h = 0$	$44M + 6S(34M + 6S)^a$	$29M + 6S$

[a] Mixed addition.

3.3 Fast Group Operation for HEC

The formulae given for the group operation of HEC can be written explicitly, resulting in more efficient arithmetic. The explicit formulae was first presented in Gaudry and Harley [2000], in which the authors noticed that, according to the properties of the input divisors, the group operations can be unrolled into all possible cases. This technique is combined with the use of the Karatsuba multiplication algorithm Karatsuba and Ofman [1963] and the Chinese remainder theorem to further reduce the overall complexity of the group operations. The computational complexity of the formulae for genus-2 curves and the corresponding references are given in Table II. For the remainder of this contribution, we will denote a field multiplication by M, a field inversion by I, and a field squaring by S. In some references, the authors did not distinguish between multiplications and squarings, denoted as M/S.

Table III summarizes the achievements regarding genus-3 curves. We were able to optimize the formulae for genus-3 HEC and further generalize the results presented in Kuroki et al. [2002] to arbitrary characteristic. Tables IV and V present the explicit formulae for a group addition and a group doubling, respectively.

The improvements are based on the following techniques:

(1) Montgomery's trick of simultaneous inversions [Cohen 1993, Algorithm 10.3.4]
(2) Reordering of normalization step [Takahashi 2002]
(3) Karatsuba multiplication
(4) Calculation of the resultant r of u_1 and u_2 for the group addition as well as of u_1 and $h + 2v_1$ using Bezout's matrix
(5) Choice of HEC with certain properties

The idea of Montgomery to use simultaneous inversions saves one inversion compared to the presented formulae in Harley [2000]. This trick is applied

Table III. Improvements of the Group Operations on Genus-3 HEC

	Field Characteristic	Cost	
		Addition	Doubling
[Nagao 2000]	*general*	$4I + 200M/S$	$4I + 207M/S$
[Nagao 2000]	*odd*	$2I + 154M/S$	$2I + 146M/S$
[Kuroki et al. 2002]	*odd*	$I + 81M/S$	$I + 74M/S$
see Appendix	*general*	$I + 70M + 6S$	$I + 61M + 10S$
	two	$I + 65M + 6S$	$I + 53M + 10S$
	two, $h(x) = 1$	$I + 65M + 6S$	$I + 22M + 7S$

Table IV. Explicit Formulae for Addition on a Genus-3 HEC

Input	Weight three reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ $h = x^3 + h_2x^2 + h_1x + h_0$, where $h_i \in \mathbb{F}_2$; $f = x^7 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$;
Output	A weight three reduced divisor $D_3 = (u_3, v_3) = D_1 + D_2$

Step	Procedure	Cost
1	Resultant r of u_1 and u_2 (Bezout)	$12M + 2SQ$
2	Almost inverse $inv = r/u_1 \bmod u_2$	$4M$
3	$s' = rs \equiv (v_2 - v_1)inv \bmod u_2$ (Karatsuba)	$11M$
4	$s = (s'/r)$ and make s monic	$I + 6M + 2S$
5	$z = su_1$	$6M$
6	$u' = [s(z + w_4(h + 2v_1)) - w_5((f - v_1h - v_1^2)/u_1)]/u_2$	$15M$
7	$v' = -(w_3z + h + v_1) \bmod u'$	$8M$
8	u', i.e. $u_3 = (f - v'h - v'^2)/u'$	$5M + 2SQ$
9	$v_3 = -(v' + h) \bmod u_3$	$3M$
Total	in fields of arbitrary characteristic in fields of characteristic 2	$I + 70M + 6S$ $I + 65M + 6S$

Table V. Explicit Formulae for Doubling on a Genus-3 HEC

Input	A weight three reduced divisors $D_1 = (u_1, v_1)$ $h = x^3 + h_2x^2 + h_1x + h_0$, where $h_i \in \mathbb{F}_2$; $f = x^7 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$;
Output	A weight three reduced divisor $D_2 = (u_2, v_2) = [2]D_1$

Step	Procedure	Cost	
1	Resultant r of u_1 and $h + 2v_1$ (Bezout)	$6M + 2S$	–
2	Almost inverse $inv = r/(h + 2v_1) \bmod u_1$	$4M$	–
3	$z = ((f - hv_1 - v_1^2)/u_1) \bmod u_1$	$7M + 2S$	$3M + 2S$
4	$s' = zinv \bmod u_1$ (Karatsuba)	$11M$	–
5	$s = (s'/r)$ and make s monic	$I + 6M + 2S$	$I + 2M + 1S$
6	$G = su_1$	$6M$	$6M$
7	$u' = u_1^{-2}[(G + w_4v_1)^2 + w_4hG + w_5(hv_1 - f)]$	$5M + 2S$	$2S$
8	$v' = -(Gw_3 + h + v_1) \bmod u'$	$8M$	$7M$
9	u', i.e. $u_2 = (f - v'h - v'^2)/u'$	$5M + 2S$	$1M + 2S$
10	$v_2 = -(v' + h) \bmod u_2$	$3M$	$3M$
Total	in fields of arbitrary characteristic in fields of characteristic 2 in fields of characteristic 2 and with $h(x) = 1$	$I + 61M + 10S$ $I + 53M + 10S$	$I + 22M + 7S$

in Step 4 in Table IV and in Step 5 in Table V. The second technique allows us to calculate the required monic polynomial u with less field operations (first introduced in Takahashi [2002]). Applying Karatsuba's method saves additional field multiplications in Step 3 in Table IV and in Step 4 in Table V. The calculation of the resultant using Bezout's matrix in the case of genus-3 HEC can be performed very efficiently compared to former publications [e.g., Kuroki et al. 2002] (Step 1 in Table IV and Table V). Notice, that there is no benefit for genus-2 HEC when using Bezout's matrix in the corresponding steps. Finally, we found out that the ideal types of genus-3 curves seem to be of the form $y^2 + y = f(x)$ over fields of characteristic two (Table V). To our knowledge these genus-3 curves have no security limitations [Gaudry 2000; Scholten and Zhu 2002]. More detailed information about the optimization techniques used can be found in Pelzl et al. [2003].

3.4 Security of the Cryptosystems

The security of ECC and HECC relies on the DHP: given a prime p, a generator α of $\mathbb{Z}_p^\star$, and elements $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$ [Menezes et al. 1997]. The DHP in the group of an EC or in the Jacobian of HEC is not explicitly considered here.

The Pollard rho method and its variants [Gallant et al. 1998; Pollard 1978; Wiedemann 1986] are the most important examples of algorithms for solving the DLP for ECC and HECC with complexity $O(\sqrt{n})$ in groups of order n. In Gaudry [2000], it is shown that index-calculus algorithms in the Jacobian of HEC of genus greater than 4 have a lower complexity than the Pollard rho method. However, for some special cases of curves, ECC can be attacked with complexity lower than $O(\sqrt{n})$ [Menezes et al. 1993]. This attack was generalized for arbitrary genus in Frey and Rück [1994], and Rück [1999]. In Gaudry et al. [2000], an attack compromising EC over the underlying finite field $\mathbb{F}_{2^m}$, where m is composite, is presented.

The cryptosystems used are defined over finite fields of order between 2^{162} and 2^{191}. According to the work of Lenstra and Verheul [2000], 160-bit and 191-bit ECC system may be considered of equivalent security to 1825-bit and 3214-bit RSA systems, respectively. Further, adequate security for commercial use can be achieved with 160-bit ECC until the year 2019 and with 191-bit ECC until the year 2040 [Lenstra and Verheul 2000]. This notion of commercial security is based on the hypothesis that a 56-bit block cipher offered adequate security in 1982 for commercial applications.

4. METHODOLOGY

The overall performance of EC and HEC cryptosystems depends not only on the specific algorithms but also on the underlying implementation and the processor type used. In particular, we analyzed how different ECCs and HECCs perform with respect to certain settings of both the software routines and the hardware components.

Table VI. Hardware Platforms Used

	Board & Processor	Clock Rate [MHz]	Memory/Cache [kByte]	Tools
ARM	Evaluator-7T KS32C50100	50	512 flash EPROM 512 SRAM 8 cache	ARM Developer Suite 1.2
ColdFire	SBC5307 Arnewsh MFC5307	90	4 SRAM 8 cache	SingleStep Deb. 7.6.2 Diab Data Comp. 4.3f.
PowerPC	TQsystem MPC823E	50	8 data cache 16 instruction cache	SingleStep Deb. 7.6.2 Diab Data Comp. 4.3f.

4.1 The Software

We implemented different variants of ECCs and HECCs. For EC, projective coordinates according to the standard IEEE P1363 [IEEE 1999] with standardized curves over different extension fields $\mathbb{F}_{2^n}$ were implemented. For HEC, we applied the currently fastest explicit formulae for the group operations on curves of genus two and three over fields of characteristic two (Tables II and III). Genus-2 curves are implemented for $h(x) \neq 1$ because other curves are considered insecure [Galbraith 2001]. For genus-3 curves, our implementation includes arbitrary curves with different properties.

We further examine the performance gain of special field reduction routines in contrast to standard reduction routines. For the remainder of this contribution, we refer to *special* reduction when using a fixed field extension polynomial. Whereas the term *standard* reduction is used for a generically implemented reduction routine and the extension polynomial is not known in advance. For a server with different cryptographic applications, standard routines have to be implemented whereas implementations on constrained platforms need only specialized settings.

The characteristic of the underlying fields is two and the cardinality of the groups is between 2^{160} and 2^{195}. All operations are implemented for 32-bit microprocessors using the C programming language. Due to portability, the implementation was not optimized for a specific platform.[3] Compiler settings for optimal speed were used depending on the tools available.

4.2 The Hardware

In this contribution, different hardware architectures for embedded systems, namely ARM7, ColdFire, and PowerPC were chosen as testing platforms for the extensive analysis of ECC and HECC. Further, the influence of the data cache and the instruction cache was analyzed on the PowerPC.

The platforms and features are introduced in Table VI. More detailed information about each processor can be found in Appendix D.

[3]Significant speed gains can be achieved by implementing the core routines in assembly using processor specific operations.

5. ARITHMETIC

In the following subsection, the implemented finite field algorithms and the group operations of ECC and HECC are investigated in detail.

5.1 Finite Field Algorithms

The speed of the underlying implementation of the field arithmetic is crucial for the overall performance of the whole cryptosystem. Restricting oneself to a certain field with a fixed field extension polynomial offers the possibility to benefit from special field reduction routines. In our work, we investigated the performance gain of such special routines versus standard routines and the resulting benefit to the overall performance. A brief summary of the algorithms used for the field arithmetic is given below.

Field addition, multiplication, squaring, inversion, and reduction are the basis for the group operations on ECs and HECs. Adding elements in $\mathbb{F}_{2^n}$ is simply accomplished by a bit-wise XOR of the components. A field multiplication of m_1 words times m_2 words is split up into several multiplications with a smaller number of words. The algorithm is a modified version of Karatsuba and Ofman [1963]. For fields in even characteristic, squaring can be done very fast by table lookups. The modified extended Euclidean algorithm (EEA) is applied for inversion in $\mathbb{F}_{2^n}$ [Hankerson et al. 2000]. Further, we were able to speed up this algorithm with a small modification concerning the calculation of the degree difference.

To represent elements of the extension field $\mathbb{F}_{2^n} = \mathbb{Z}_2/p(x)$, we need to choose an irreducible polynomial. In von zur Gathen and Nöcker [2000], the authors conjecture that the minimal number of terms $\sigma_q(n)$ in irreducible polynomials of degree n in $GF(q)$, where q is a prime power, is for all $n \geq 1$, $\sigma_2(n) \leq 5$ and $\sigma_q(n) \leq 4$ for $q \geq 3$. This conjecture has been verified for $q = 2$ and $n \leq 10{,}000$ [Blake et al. 1993; Golomb 1967; von zur Gathen and Nöcker 2000; Zierler 1970; Zierler and Brillhart 1968, 1969] and for $q = 3$ and $n \leq 539$ [von zur Gathen 2001]. Hence, we found extension polynomials that are either trinomials of the form $p(x) = x^n + x^k + 1$ or pentanomials of the form $p(x) = x^n + x^{k_1} + x^{k_2} + x^{k_3} + 1$.

The implemented general reduction function considers tri- and pentanomials and is able to treat arbitrary values k_i. The reduction itself is done word-wise according to Hankerson et al. [2000, Algorithm 6]. In order to achieve a higher speed-up, we additionally implemented a special reduction function for each underlying field, where the k_i are fixed.

5.2 Group Arithmetic on Elliptic Curves

The implementation of the high-level EC group operations uses projective coordinates according to the standard IEEE P1363 [IEEE 1999]. The operations performed are as follows:

—*point addition*—in general this algorithm requires 5 field squarings, 15 general field multiplications

—*point doubling*—this algorithm requires 5 field squarings, 5 general field multiplications,

—*scalar multiplication*—we implement the addition-subtraction method as outlined in IEEE P1363[IEEE 1999]

5.3 Group Arithmetic on Hyperelliptic Curves

For the group operations on HECs of genus 2 and 3 the explicit formulae were implemented (for more detail see Tables II and III). We considered the case where the coefficients of $h(x)$ are elements of $\mathbb{F}_2$. For genus-3 curves with $h(x) = 1$ were investigated. For the main operation of the cryptosystem, the repeated addition of a divisor, we used the sliding window exponentiation algorithm [Menezes et al. 1997, Section 14.6.1].

6. RESULTS

This section summarizes and analyzes our implementation results. The emphasis lies on the performance of the different platforms, the comparison of the targeted cryptosystems with different implementation options, as well as on the influence of the hardware settings.

ECC and HECC are implemented using general reduction routines. In addition, we implemented HECC with special (fixed) reduction polynomials to be able to analyze the performance gain. In the case of genus-3 curves, we were able to find a more efficient group operation, when using the $h(x) = 1$. Unfortunately, this speed up is not possible with curves of odd genus, because of security reasons (for more information see Section 3.4).

6.1 ECC and HECC on Different Platforms

We implemented ECC and HECC on different embedded platforms with high practical relevance, namely ARM, ColdFire, and PowerPC (Figure 1). In addition to these embedded platforms we timed the code also on a general-purpose machine (a Pentium IV). All timings of the scalar multiplication concerning group orders around 2^{160}, 2^{170}, 2^{180}, and 2^{190} can be found in Table VII. For the boards at hand we could achieve the best timings for the HECC implementation on the PowerPC. One scalar multiplication for HECC took 117 and 84.9 ms for genus-2 and genus-3 curves, respectively. The scalar multiplication for ECC can be performed fastest on the PowerPC at 50 MHz resulting in 106.3 ms.

However, solely considering the clock frequency of the processors is of very limited value. Due to the different hardware architectures of the platforms and the varying board features the actual timings can be quite different, though the processor clockrate is equal (see Section 6.4).

6.2 Standard versus Special Implementation

There are two major ways of implementing a cryptographic algorithm. One way is to allow all possible input parameters, for example, arbitrary curves and irreducible polynomials. This form is referred to as standard implementation and is used in server applications or cryptographic libraries. Further, it is sufficient

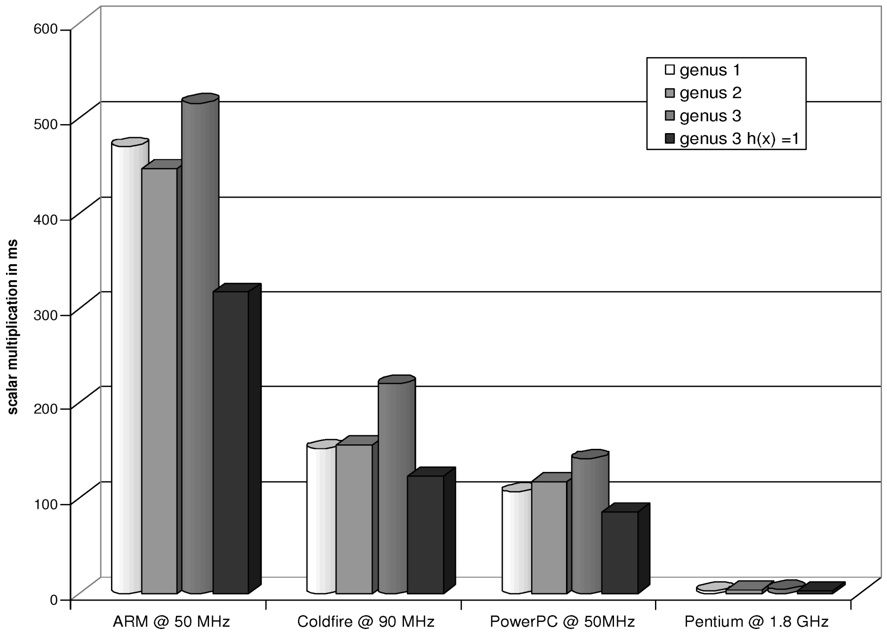

Fig. 1. Implementation of ECC and HECC on different platforms (group order: $\approx 2^{160}$).

Table VII. Timings of the Scalar Multiplication of ECC and HECC on Different Platforms (in ms)

Group Order		ECC	HECC $g=2$	HECC $g=3$	HECC $g=3, h(x)=1$
$\approx 2^{160}$	ARM @ 50 MHz	469.96	446.46	515.46	316.6
	ColdFire @ 90 MHz	152.1	155.6	219.4	123.6
	PowerPC @ 50 MHz	106.3	117	141.4	84.9
	Pentium @ 1.8 GHz	2.6	3.61	4.15	2.58
$\approx 2^{170}$	ARM @ 50 MHz	397.12	461.36	523.12	321.12
	ColdFire @ 90 MHz	132.8	161.5	225.1	126.9
	PowerPC @ 50 MHz	94.5	121.2	145.4	87
	Pentium 1.8 GHz	2.43	3.8	4.84	2.7
$\approx 2^{180}$	ARM @ 50 MHz	515.95	516.5	577.5	356.99
	ColdFire @ 90 MHz	171.7	183.4	246.7	146.2
	PowerPC @ 50 MHz	121.8	138.1	160.1	96.8
	Pentium @ 1.8 GHz	2.8	4.3	5.77	2.92
$\approx 2^{190}$	ARM @ 50 MHz	436.01	542.68	581.24	360.24
	ColdFire @ 90 MHz	157.8	187.6	258.5	147
	PowerPC @ 50 MHz	112.4	141.7	167.8	101.8
	Pentium @ 1.8 GHz	2.78	4.47	5.49	3.01

to target specific implementations of algorithms when constrained in memory and processor power (e.g., allowing only standardized curves or even a fixed curve). The more specific the implementation the higher the efficiency. In this subsection, we focus on the impact of using the specific versus the standard implementation.

Table VIII. Influence of Special and Standard Field Reduction (all Timings in μs, Platform: ARM @ 50 MHz)

Field	General Red.		Special Red.		General / Special	
	mult	squ	mult	squ	mult	squ
2^{54}	50	28	32	10	1.56	2.8
2^{55}	50	28	32	10	1.56	2.8
2^{59}	65	42	33	11	1.97	3.82
2^{60}	59	27	32	10	1.75	2.7
2^{61}	65	42	33	11	1.97	3.82
2^{63}	50	28	32	9	1.56	3.11
2^{81}	84	35	62	13	1.35	2.69
2^{83}	103	54	62	13	1.66	4.15
2^{88}	104	56	62	13	1.68	4.31
2^{91}	104	56	62	13	1.68	4.31
2^{95}	84	35	62	12	1.35	2.92

6.2.1 *Performance of Underlying Field Arithmetic.* We implemented the frequently used finite field functions, namely modular multiplication and modular squaring in two different ways. At first we used a *standard* implementation with a reduction function capable of handling arbitrary irreducible polynomials. Second, we fixed the polynomial and therefore had to program separate reduction routines for each of the finite fields used, and we refer to this option as *special*. Table VIII shows the timings for multiplication and squaring with different underlying fields using standard and special reduction routines on the ARM microprocessor.

Analyzing the throughput of the functions the special modular multiplication routine is up to two times faster compared to the standard implementation. In the case of squaring, the gain is even higher and an increase in performance by a factor of 4 can be achieved. The difference in the performance gain relies on the reduction routine, playing a crucial role in the squaring routine.

Figure 2 depicts the timings of the modular arithmetic for different fields. The evaluation of this figure yields to the following conclusions:

(1) The performance rises unusually between the fields $\mathbb{F}_{2^{63}}$ and $\mathbb{F}_{2^{81}}$. The increase results from the fact that the implementation is targeted for 32-bit processors. The field elements in $\mathbb{F}_{2^{63}}$ can be represented with two words, whereas in the case of $\mathbb{F}_{2^{81}}$ three words have to be provided.

(2) In the specific implementation no input parameters are used because they are chosen in advance, resulting in a nearly monotonic slope for a constant number of words. The standard implementation depends heavily on the chosen irreducible polynomial, which can be seen from the nonmonotonic slope of the graphs. In our implementation, we used trinomials and pentanomials. The latter case applied when there were no irreducible trinomials available. For example in the case of the underlying field $\mathbb{F}_{2^{55}}$, we used a trinomial and for the field $\mathbb{F}_{2^{59}}$, a pentanomial was used. The larger overhead for a standard routine using a pentanomial instead of a trinomial leads to a decrease in speed for multiplication and squaring.

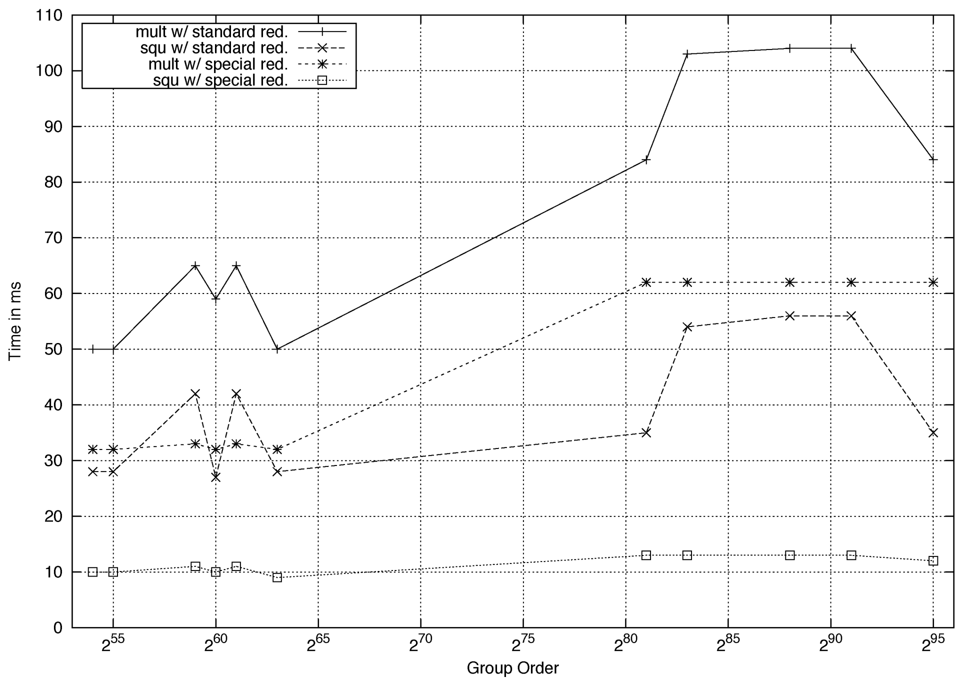

Fig. 2. Comparison of standard versus special implementation of the field arithmetic (platform: ARM @ 50 MHz).

Table IX. Influence of Different Reduction Routines on HECC Scalar Multiplication (all Timings in ms, Platform: ARM @ 50 MHz)

Group Order	Standard Reduction			Special Reduction			Standard / Special		
	g = 2	g = 3		g = 2	g = 3		g = 2	g = 3	
		$h(x) = 1$			$h(x) = 1$			$h(x) = 1$	
$\approx 2^{160}$	565.97	449.62	749.36	446.46	316.6	515.46	1.27	1.42	1.45
$\approx 2^{170}$	682.86	454.81	758.72	461.36	321.12	523.12	1.48	1.42	1.45
$\approx 2^{180}$	766.64	504.75	837.71	516.5	356.99	577.5	1.48	1.41	1.45
$\approx 2^{190}$	681.36	513.66	852.15	542.68	360.24	581.24	1.26	1.43	1.47

6.2.2 *Influence on the Scalar Multiplication.* Table IX shows how the different implementations of the underlying library influence the performance of the HECC. For genus-2 curves, the ratio of the standard implementation to that of the special implementation is in the range of 1.27–1.48. In the case of genus-3 curves, scalar multiplication can be accelerated by almost 50%. The performance gain is not as huge as for the plain field operations (see Section 6.2.1) because of additional overhead and other underlying functions (e.g., inversion) that are not optimized.

6.3 Comparing the Performance of the Different Cryptosystems

Figure 3 shows the performance of different cryptosystems implemented with standard and special reduction routines on the ARM microprocessor. Note that analyzing the performance only for one specific platform is not

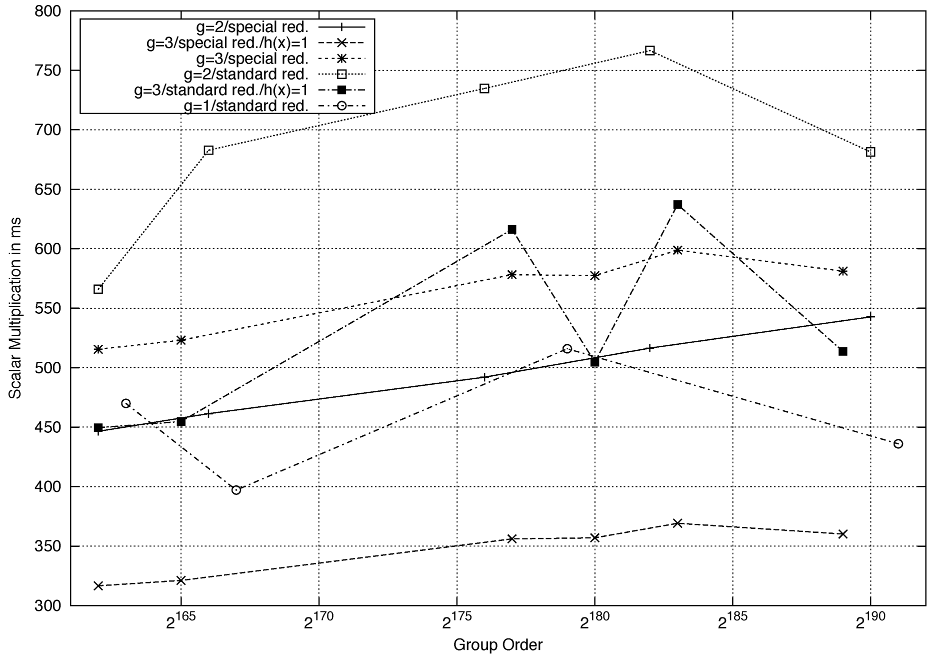

Fig. 3. Performance comparison of different ECC and HECC implementations (platform: ARM @ 50 MHz)

sufficient to draw conclusions about the general performance of the different cryptosystems. Figures for the other embedded platforms can be found in Appendix A.

From the figures follow that the computation time for a group operation is dependent on a variety of factors that are interrelated, for example, the complexity of the group operation depends on the curve parameters and so on. This fact is obvious if we consider the different implemented genus-3 curves. When using genus-3 curves with $h(x) = 1$, a highly efficient group operation can be achieved compared to arbitrary $h(x)$. Further, interdependence of the run-time and the hardware architecture is noticeable. Consider for example the relative performance between genus-2 HEC using special reduction and genus-3 HEC using general reduction: On the ARM (Figure 3) and the ColdFire (Figure 4), these genus-3 implementations are the worst followed by the genus-2 curves. Analyzing the performance of the same curves on the PowerPC (Figure 5), the genus-2 curve has the worst timings. Hence, the performance is heavily related to properties of the underlying platform.

In the case of genus-3 HECC with $h(x) = 1$ using special reduction routines, the ARM takes the shortest time to compute the scalar multiplication. The graphs of the standard implementation show that the cryptographic systems for genus-1 and genus-3 with $h(x) = 1$ have approximately the same performance. In the cases of a group order of 2^{160} and 2^{180}, HECC can even outperform ECC. Regarding performance, these two implementations are followed by genus-2 and genus-3 HECC with arbitrary $h(x)$. In conclusion, it can be seen

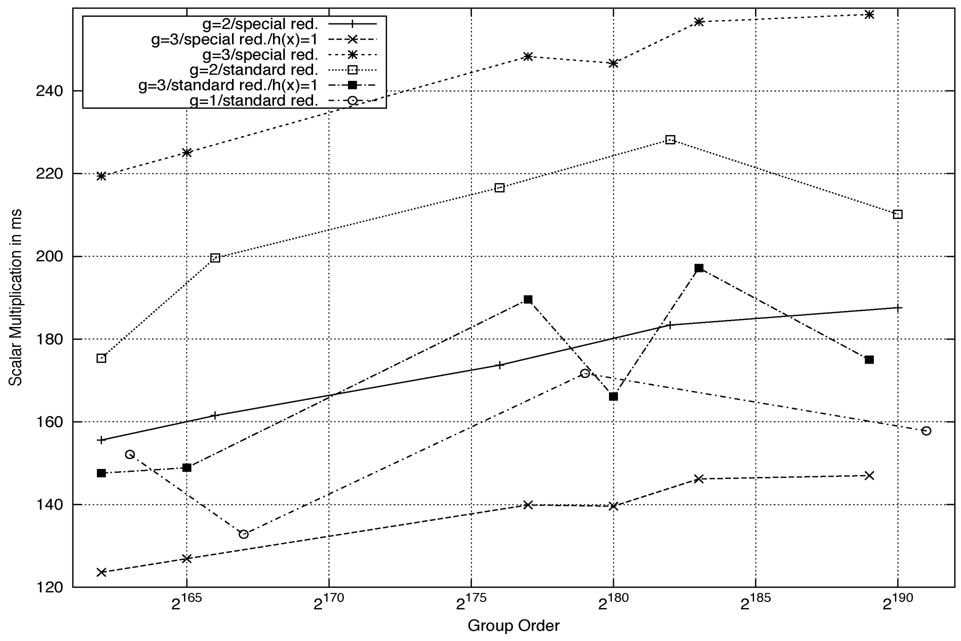

Fig. 4. Performance comparison of different ECC and HECC implementations (platform: ColdFire @ 90 MHz).

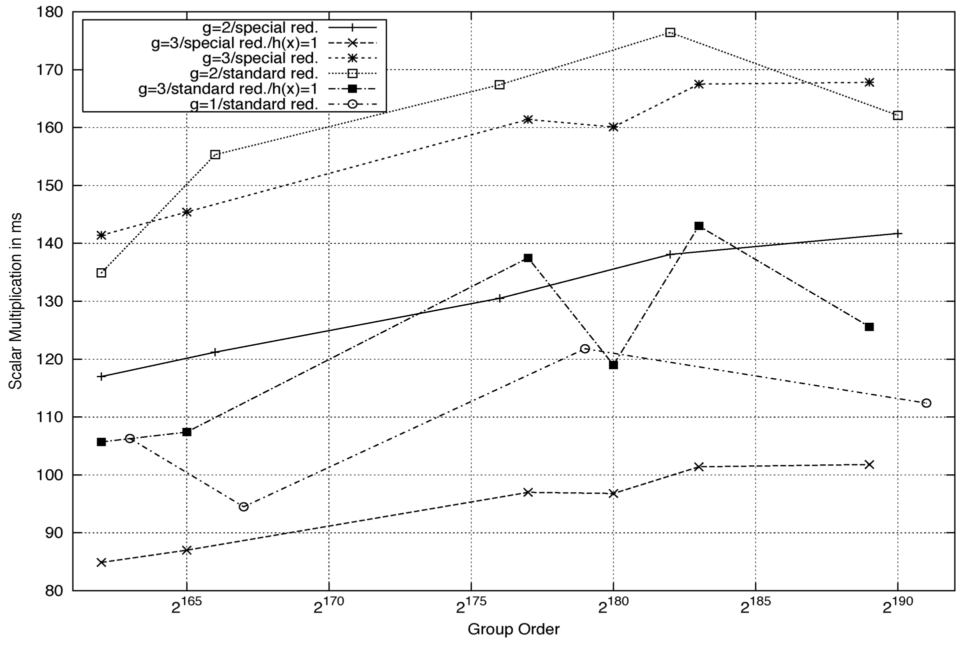

Fig. 5. Performance comparison of different ECC and HECC implementations (platform: PowerPC @ 50 MHz).

Table X. Influence of Different Cache Options on ECC and HECC Performance (all Timings in ms, Platform: PowerPC @ 50 MHz, see Table XI for Ratios)

	Group Order	Cache, serialized			No Cache	
		Data + Instruction	Instruction	Data	Serialized	Not Serialized
ECC	2^{163}	106.4	271.9	626.2	828.1	1249
	2^{167}	94.5	241	553.5	732.4	1062.8
	2^{179}	122	311.4	713.7	944.6	1371.4
	2^{191}	112.5	286.3	659	871.7	1264.7
HECC, g = 2	2^{162}	117	272.8	695.8	886.1	1293
	2^{166}	121.2	280.7	722.2	916.6	1339
	2^{176}	130.5	300.9	776.2	984.9	1438
	2^{182}	138.1	317.9	821.8	1042	1521
	2^{190}	141.7	328.8	841.7	1071	1562

Table XI. Ratios of the ECC and HECC Scalar Multiplication Using Different Cache Settings (Platform: Power PC @ 50 MHz, see Table X for the Timings)

	Group Order	No Cache / Data + Instruction	No Cache / Instruction	No Cache / Data	No Serialized / Serialized (No Cache)
ECC	2^{163}	7.78	3.05	1.32	1.51
	2^{167}	7.75	3.04	1.32	1.45
	2^{179}	7.74	3.03	1.32	1.45
	2^{191}	7.75	3.04	1.32	1.45
HECC, g = 2	2^{162}	7.57	3.25	1.27	1.46
	2^{166}	7,56	3.27	1.27	1.46
	2^{176}	7.55	3.27	1.27	1.46
	2^{182}	7.55	3.28	1.27	1.46
	2^{190}	7.56	3.26	1.27	1.46

that using certain curves can have significant performance pay-offs in terms of performance.

Interestingly, genus-3 HECC have the worst timing in standard implementation and therefore do not look promising. On the other hand, when restricting ourselves to curves with $h(x) = 1$ and special field operations, the best performance is achieved.

Contrary to common belief we were able to show: (1) genus-3 HEC with $h(x) = 1$ can outperform ECC and (2) these genus-3 curves are faster than genus-2 curves. Thus, besides ECC, HECC is perfectly suited for embedded security applications.

6.4 Influence of Cache

The performance of a cryptographic system depends a lot on the processor and on the available resources of the board. In this subsection, we analyze the influence of different cache settings.

Table X shows the influence of the cache targeting ECC and genus-2 HECC implementations on the PowerPC. Normalizing these timings with respect to the obtained execution times with disabled cache leads to the ratios stated in Appendix C, Table XI. It is noticeable that there is almost no difference in the impact of the cache setting for ECC and HECC.

The data cache is advantageous when intensive memory access is necessary. The utilization of the instruction cache dominates in projects consisting of small functions, which get called frequently. In our case, the latter applies: the code size is relatively small and the functions called most frequently consist of only few commands. This is confirmed by the timings on the PowerPC. It can be seen that the performance increases by a factor of 1.3 when using only data cache and by a factor of 3.3 when only using instruction cache. The computation of a scalar multiplication is about a factor of 7.7 faster if using instruction and data cache. Since we are using a cache 16 k byte cache, the most relevant subroutines are permanently cached. In addition, the serialized mode compared to the nonserialized mode can speed up the design by almost 50%.

Hence, we advise using at least an instruction cache, or better, both kinds of cache when running ECC or HECC.

6.5 Koblitz Curves

Koblitz, or subfield, curves are a very special type of algebraic curves [Koblitz 1991]. They are well studied in the ECC case (including standardization). On the other hand, relatively little work has been done for subfield curves for HECC, with the exception of Günter et al. [2000]. Thus, we decided to only implement subfield curves for the ECC case. Comparing HECC and ECC subfield curves is certainly an interesting undertaking, but it is not obvious whether such a comparison would be meaningful as the cryptographic security considerations in both cases might be different. We implemented the Frobenius map using Koblitz curves targeting the group order $\approx 2^{160}$. On the ARM (@50 MHz) it took 75.29 ms, on the ColdFire (@90 MHz) 33.9 ms, and on the PowerPC (@ 50 MHz) 23.3 ms.

7. CONCLUSION

The work at hand presents the first implementation of HECC on embedded systems and provides a thorough comparison of ECC and HECC on a variety of relevant embedded hardware architectures. In addition, optimized explicit formulae for the group operation of genus-3 HECC are introduced. The best performance for a HECC scalar multiplication took 84.9 ms on the PowerPC. Our implementations demonstrate that HECC is perfectly suited for use in constrained environments. Contrary to common belief, HECC can reach the same throughput as ECC.

Further, we investigated the influence of using specific arithmetic versus standard arithmetic and the dependence of hardware settings on the performance. We found a clear quantitative improvement of 50% by specializing the reduction routine for HECC. Independent of the cryptosystem, the presence of cache can speed up the performance by almost a factor of eight for the processor used in our comparison.

This contribution clearly shows that HECC (as equal alternative to ECC) can be the cryptosystem of choice for future embedded security applications.

APPENDIX

A. PERFORMANCE OF ECC AND HECC ON DIFFERENT HARDWARE ARCHITECTURES

For each of the embedded platforms ColdFire and PowerPC the distribution of the performance considering ECC and HECC for different underlying fields are shown in Figures 4 and 5, respectively. For the figure targeting the ARM platform (see Section 6.3 Figure 3).

B. EXPLICIT FORMULAE FOR GENUS THREE HECC

The explicit formulae for the group operations on HEC of genus three and arbitrary characteristic as well as the most efficient formulae for doubling on a special HEC for characteristic two is presented in Tables IV and V.

C. CACHE INFLUENCE

In Table IV, the ratios analyzing the influence of the cache for two different implementations on the PowerPC are given. The timings of the scalar multiplication using different cache settings can be found in Section 6.4 (Table X).

D. HARDWARE PLATFORMS

This section introduces the hardware platforms used in our contribution.

ARM: ARM (Advanced RISC Machine) processors are typically used for embedded applications such as small network devices, controllers, and mobile phones. Especially for secure systems like Online Banking, Pay-TV, Network Security and so on. A *SecurCore* variant of the ARM7 processor was developed, which has instruction independent power peaks to avoid side-channel attacks.

On the ARM microprocessor [ARM 2000], instruction decoding is performed with static (i.e., hard-wired) logic for a faster result. The ARM7TDMI is based on von Neuman architecture and is licensed by ARM Ltd. All instructions have a fixed uniform length to simplify the decoding procedure. Since direct manipulation of data in the memory is not possible, a load/store architecture handles data processing through registers. The simple address mode allows to determine all load/store addresses from the register contents and the instruction parameters. For low power consumption the ARM7 possesses the *Thumb Instruction Set*, which is restricted to 16-bit and allows compact code, and thus, is feasible for small hand-held devices such as PDAs.

The ARM7TDMI consists of a program control unit, an address generator, an integer data path, and a general-purpose register bank. The data path contains a 32-bit integer ALU, a multiply-add unit, and a barrel shifter. The 32-bit ALU performs simple integer arithmetic operations such as add and subtract. The core features a multicycle 32×32 to 64-bit multiplier. It has a total of 37 registers: 31 general-purpose 32-bit registers, and 6 status registers. Speed-critical control signals are pipelined so that system control functions can be implemented in standard low-power logic. The ARM7TDMI does not support floating-point arithmetic in hardware and does not have any DSP-specific features.

ColdFire: The ColdFire microprocessor is the successor of the 68000 series. Besides the use as low cost controller (laser printers), the ColdFire is used in general-purpose industry applications such as network elements (routers, bridges).

In version 3, the processor consists of two independent, decoupled pipeline structures [Motorola 2000a]. The instruction fetch pipeline is a six-stage pipeline for prefetching instructions and contains logic for branch prediction. To maximize the performance, the 4 k byte on-chip SRAM provides one-cycle access for the ColdFire core. This SRAM can store processor stack and critical code or data segments to maximize performance. The processor core possesses a hardware integer divide unit and supports a 16×16 and 32×32 bit multiplication. The ColdFire features 16 32-bit general-purpose registers.

PowerPC: Typical applications for embedded PowerPCs include powerful general-purpose microcontroller, data acquisition systems, applications in robotics, and automotive and consumer electronics.

The standard PowerPC architecture has a fully static design that consists of three functional blocks: the integer block, hardware multiplier/divider, and load/store block [Motorola 2000b]. The core supports integer operations on a 32-bit internal data path and 32-bit arithmetic hardware. Its interface to the internal and external busses is 32 bits. The PowerPC integer block supports 32×32-bit fixed-point general-purpose registers and can execute one integer instruction per clock cycle. The core is integrated with the memory management units, an instruction cache, and a data cache. The 8 k byte data cache allows single-cycle accesses. The two way 16 k byte instruction cache is set-associative.

The PowerPC offers the possibility to disable the data cache as well as the instruction cache separately. For this reason, we investigate four different options for the cache: cache enabled, data cache only, instruction cache only, and cache disabled. The timings under these options provide information about the performance depending on the cache. For programs with intensive memory access, the data cache may play a more significant role than the instruction cache whereas for projects consisting of small functions which get called several times, the instruction cache might be more important.

The PowerPC allows disabling of the pipelining mode. If the core is in *non-serialized* mode, no pipelining is applied. Hence, the next processor command is executed only after the previous has been processed completely. In *serialized* mode, full pipelining is enabled.

REFERENCES

Agnew, G. B., Mullin, R. C., and Vanstone, S. A. 1993. An implementation of elliptic curve cryptosystems over $F_{2^{155}}$. *IEEE J. Select. Areas Commun. 11*, 5 (June), 804–813.

ANSI X9.62-1999. 1999. The Elliptic Curve Digital Signature Algorithm. Tech. rep., ANSI.

ANSI X9.63-199x. 1998. Elliptic Curve Key Agreement and Key Transport Protocols. Draft, ANSI. Working document.

ARM. 2000. ARM Evaluator-7T Board User Guide. Available at http://www.arm.com/support/.

Blake, Gao, and Lambert. 1993. Constructive problems for irreducible polynomials over finite fields. In *Information Theory and Applications*. Springer-Verlag, Berlin, 1–23.

Blake, I., Seroussi, G., and Smart, N. 1999. *Elliptic Curves in Cryptography.* London Mathematical Society Lecture Notes Series, vol. 265. Cambridge University Press, Cambridge, UK.

BOSTON, N., CLANCY, T., LIOW, Y., AND WEBSTER, J. 2002. Genus two hyperelliptic curve coprocessor. In *Cryptographic Hardware and Embedded Systems—CHES 2002*, J. c. K. K. B. S. Kaliski and C. Paar, eds. Lecture Notes in Computer Science, vol. 2523. Springer-Verlag, Berlin, 529–539.

CANTOR, D. 1987. Computing in Jacobian of a hyperelliptic curve. *Math. Comput. 48*, 177, 95–101.

CHUDNOVSKY, D. AND CHUDNOVSKY, G. 1987. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Adv. Appl. Math 7*, 385–434.

COHEN, H. 1993. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics, vol. 138. Springer-Verlag, Berlin, Germany (Third corrected printing 1996).

COHEN, H., MIYAJI, A., AND ONO, T. 1998. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology—ASIACRYPT'98*, K. Ohta and D. Pei, eds. Lecture Notes in Computer Science, vol. 1514. Springer-Verlag, Berlin, 51–65.

DIERKS, T. AND ALLEN, C. 1999. *RFC 2246: The TLS Protocol Version 1.0*. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA.

FREIER, A. O., KARLTON, P., AND KOCHER, P. C. 1996. *The SSL Protocol Version 3.0*. Transport Layer Security Working Group Internet-draft.

FREY, G. AND RÜCK, H.-G. 1994. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comput. 62*, 206 (Apr.), 865–874.

FULTON, W. 1969. *Algebraic Curves—An Introduction to Algebraic Geometry*. W. A. Benjamin, Inc., Reading, M.

GALBRAITH, S. 2001. Supersingular curves in cryptography. In *Lecture Notes in Computer Science*, vol. 2248. 495–517.

GALLANT, R., LAMBERT, R., AND VANSTONE, S. 1998. Improving the parallelized Pollard lambda search on binary anomalous curves. Available at http://www.certicom.com/chal/download/paper.ps.

GAO, L., SHRIVASTAVA, S., AND SOBELMAN, G. 1999. Elliptic curve scalar multiplier design using FPGAs. In *Workshop on Cryptographic Hardware and Embedded Systems—CHES 19999*, Ç. Koç and C. Paar, eds. Lecture Notes in Computer Science, vol. 1717. Springer-Verlag, Berlin.

GAUDRY, P. 2000. An algorithm for solving the discrete log problem on hyperelliptic curves. In *Advances in Cryptology—EUROCRYPT 2000*, B. Preneel, ed. Lecture Notes in Computer Science, vol. 1807. Springer-Verlag, Berlin, Germany, 19–34.

GAUDRY, P. AND HARLEY, R. 2000. Counting points on hyperelliptic curves over finite fields. In *ANTS IV*, W. Bosma, ed. Lecture Notes in Computer Science, vol. 1838. Springer Verlag, Berlin, 297–312.

GAUDRY, P., HESS, F., AND SMART, N. P. 2000. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. Tech. Rep. HPL 2000-10, HP Labs. Available at http://www.hpl.hp.com/techreports/2000/HPL-2000-10.html.

GÜNTER, C., LANGE, T., AND STEIN, A. 2000. Speeding up the arithmetic on koblitz curves of genus two. In *Seventh Annual Workshop on Selected Areas in Cryptography—SAC 2000.* Lecture Notes in Computer Science, vol. 2012. Springer-Verlag, Berlin, Germany, 106–117.

GOLOMB, S. 1967. *Shift Register Sequences*. Holden-Day, San Francisco, California, USA.

GORDON, D. M. 1998. A survey of fast exponentiation methods. *J. Algorithms 27*, 129–146.

GUAJARDO, J. AND PAAR, C. 1997. Efficient algorithms for elliptic curve cryptosystems. In *Advances in Cryptology—CRYPTO '97*, B. Kaliski, ed. Lecture Notes in Computer Science, vol. 1294. Springer-Verlag, Berlin, Germany, 342–356.

GURA, N., CHANG, S., EBERLE, H., SUMIT, G., GUPTA, V., FINCHELSTEIN, D., GOUPY, E., AND STEBILA, D. 2001. An end-to-end systems approach to elliptic curve cryptography. In *Cryptographic Hardware and Embedded Systems—CHES 2001*. Lecture Notes in Computer Science, vol. 1965. Springer-Verlag, Berlin, 351–366.

HANKERSON, D., HERNANDEZ, J. L., AND MENEZES, A. 2000. Software implementation of elliptic curve cryptography over binary fields. In *Second International Workshop on Cryptographic Hardware and Embedded Systems—CHES 2000*, Ç. Koç and C. Paar, eds. Lecture Notes in Computer Science, vol. Springer-Verlag, Berlin.

HARLEY, R. 2000. Fast Arithmetic on Genus Two Curves. Available at http://cristal.inria.fr/harley/hyper/. adding.txt and doubling.c.

IEEE 1999. *IEEE P1363 Standard Specifications for Public Key Cryptography*. IEEE. Last Preliminary Draft.

Karatsuba, A. and Ofman, Y. 1963. Multiplication of multidigit numbers on automata. *Sov. Phys. Dokl. (English translation) 7*, 7, 595–596.

Kent, S. and Atkinson, R. 1998. *RFC 2401: Security Architecture for the Internet Protocol*. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA.

King, B. 2001. An improved implementation of elliptic curves over *GF*(2) when using projective point arithmetic. In *Eighth Annual Workshop on Selected Areas in Cryptography—SAC 2001*, S. Vaudenay and A. M. Youssef, eds. Lecture Notes in Computer Science, vol. 2259. Springer-Verlag, Berlin, Germany, 134–150.

Koblitz, N. 1987. Elliptic curve cryptosystems. *Math. Comput. 48*, 203–209.

Koblitz, N. 1988. A family of jacobians suitable for discrete log cryptosystems. In *Advances in Cryptology—Crypto '88*, S. Goldwasser, ed. Lecture Notes in Computer Science, vol. 403. Springer-Verlag, Berlin, 94–99.

Koblitz, N. 1989. Hyperelliptic cryptosystems. *J. Cryptol. 1*, 3, 129–150.

Koblitz, N. 1991. Cm—curves with good cryptographic properties. In *Advances in Cryptology—CRYPTO '91*, J. Feigenbaum, ed. Lecture Notes in Computer Science, vol. 576. Springer-Verlag, Berlin, Germany, 279–287.

Koblitz, N. 1998. *Algebraic Aspects of Cryptography*, 2nd ed. Springer-Verlag, Berlin, Germany.

Krieger, U. 1997. signature.c. M.S. thesis, Mathematik und Informatik, Universität Essen, Fachbereich 6, Essen, Germany.

Kuroki, J., Gonda, M., Matsuo, K., Chao, J., and Tsujii, S. 2002. Fast genus three hyperelliptic curve cryptosystems. In *The 2002 Symposium on Cryptography and Information Security, Japan—SCIS 2002*.

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

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

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

Lenstra, A. and Verheul, E. 2000. Selecting cryptographic key sizes. In *Third International Workshop on Practice and Theory in Public Key Cryptography—PKC 2000*, H. Imai and Y. Zheng, eds. Lecture Notes in Computer Science, vol. 1751. Springer-Verlag, Berlin.

López, J. and Dahab, R. 1999. Fast multiplication on elliptic curves over $GF(2^n)$. In *Cryptographic Hardware and Embedded Systems—CHES 1999*, J. Ç. K. Koç and C. Paar, eds. Lecture Notes in Computer Science, vol. 1717. Springer-Verlag, 316–327.

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

Menezes, A., Okamoto, T., and Vanstone, S. 1993. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inf. Theory 39*, 5 (Sep.), 1639–1646.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. 1997. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, USA.

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

Miller, V. 1986. Uses of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO '85*, H. C. Williams, ed. Lecture Notes in Computer Science, vol. 218. Springer-Verlag, Berlin, Germany, 417–426.

Miyamoto, Y., Doi, H., Matsuo, K., Chao, J., and Tsuji, S. 2002. A fast addition algorithm of genus two hyperelliptic curve. In *The 2002 Symposium on Cryptography and Information Security—SCIS 2002*. IEICE, Japan, 497–50. (in Japanese).

Montgomery, P. 1987. Speeding the Pollard and Elliptic Curve methods of factorization. *Math. Comp.* 48, 243–264.

Morain, F. and Olivos, J. 1990. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theor. Inf. Applic. 24*, 6, 531–543.

Motorola. 2000a. MFC5307 User's Manual. http://e-www.motorola.com/collateral/MCF5307-BUM.pdf.

Motorola. 2000b. MPC823 User's Manual. http://e-www.motorola.com/brdata/PDFDB/docs/MPC823UM.pdf.

Mumford, D. 1984. Tata lectures on theta II. In *Progress in Mathematics,* vol. 43. Birkhäuser.

Nagao, K. 2000. Improving group law algorithms for Jacobians of hyperelliptic curves. In *ANTS IV*, W. Bosma, ed. Lecture Notes in Computer Science, vol. 1838. Springer Verlag, Berlin, 439–448.

Orlando, G. and Paar, C. 2000. A high-performance reconfigurable elliptic curve processor for $GF(2^m)$. In *Cryptographic Hardware and Embedded Systems—CHES 2000*, Ç. K. Koç and C. Paar, eds. Lecture Notes in Computer Science, vol. 1965. Springer-Verlag.

Pelzl, J. 2002. Hyperelliptic Cryptosystems on Embedded Microprocessor. M.S. thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universitaet Bochum, Bochum, Germany.

Pelzl, J., Wollinger, T., and Paar, C. 2003. Low cost security: Explicit formulae for genus-4 hyperelliptic curves. In *Tenth Annual Workshop on Selected Areas in Cryptography—SAC 2003*. Springer-Verlag, Berlin, Germany.

Pollard, J. M. 1978. Monte Carlo methods for index computation mod *p*. *Math. Comput. 32*, 143 (July), 918–924.

Rivest, R. L., Shamir, A., and Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communi. ACM 21*, 2 (Feb.), 120–126.

Rosner, M. 1999. Elliptic Curve Cryptosystems on Reconfigurable Hardware. M.S. thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA.

Rück, H.-G. 1999. On the discrete logarithm in the divisor class group of curves. *Math. Comput. 68*, 226, 805–806.

Sakai, Y. and Sakurai, K. 2000. 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. 692–703. IEICE Trans.

Sakai, Y., Sakurai, K., and Ishizuka, H. 1998. Secure hyperelliptic cryptosystems and their performance. In *Public Key Cryptography*. Lecture Notes in Computer Science, vol. 1431. Springer-Verlag, Berlin, 164–181.

Scholten, J. and Zhu, J. 2002. Hyperelliptic curves in characteristic 2. *Int. Math. Res. Notices 17*, 905–917.

Schroeppel, R., Orman, H., O'Malley, S., and Spatscheck, O. 1995. Fast key exchange with elliptic curve systems. In *Advances in Cryptology—CRYPTO '95*, D. Coppersmith, ed. 963. Springer-Verlag, Berlin, Germany, 43–56.

Silverman, J. H. 1986. *The Arithmetic of Elliptic Curves*. Springer-Verlag, New York.

Smart, N. 1999. On the performance of hyperelliptic cryptosystems. In *Advances in Cryptology—EUROCRYPT '99*, J. Stern, ed. Lecture Notes in Computer Science, vol. 1592. Springer-Verlag, 165–175.

Solinas, J. 1997. An improved algorithm for arithmetic on a family of elliptic curves. In *Advances in Cryptology—CRYPTO '97*, B. Kaliski, ed. 1294. Springer-Verlag, Berlin, Germany, 357–371.

Takahashi, M. 2002. Improving Harley algorithms for jacobians of genus 2 hyperelliptic curves. In *SCIS*. IEICE, Japan. (in Japanese).

V1on zur Gathen, J. 2001. Irreducible trinomials over finite fields. In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation—ISSAC2001*, B. Mourrain, ed. ACM Press, 332–336.

Von zur Gathen, J. and Nöcker, M. 2000. Exponentiation in finite fields: theory and practice. In *Applied Algebra, Agebraic Algorithms and Error Correcting Codes—AAECC-12*, T. Mora and H. Mattson, eds. Lecture Notes in Computer Science, vol. 1255. Springer-Verlag, Berlin, 88–113.

Wiedemann, D. H. 1986. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory IT-32*, 1 (Jan), 54–62.

Wollinger, T. 2001. Computer Architectures for Cryptosystems Based on Hyperelliptic Curves. M.S. thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA.

WOLLINGER, T. AND PAAR, C. 2002. Hardware architectures proposed for cryptosystems based on hyperelliptic curves. In *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems—ICECS 2002,* vol. III. 1159–1163.

ZIERLER, N. 1970. On $x^n + x + 1$ over $GF(2)$. *Inf. Control 16*, 67–69.

ZIERLER, N. AND BRILLHART, J. 1968. On primitive trinomials (mod2). *Inf. Control 13*, 541–554.

ZIERLER, N. AND BRILLHART, J. 1969. On primitive trinomials (mod2): II. *Inf. and Control 14*, 566–569.

Received March 2003; revised September 2003; accepted October 2003