Another Look at Square Roots (and Other Less Common Operations) in Fields of Even Characteristic

Roberto Maria Avanzi

Faculty of Mathematics and Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany
Roberto.Avanzi@ruhr-uni-bochum.de

Abstract. We discuss a family of irreducible polynomials that can be used to speed up square root extraction in fields of characteristic two. They generalize trinomials discussed by Fong et al., but our results are not limited to trinomials. In fact, we show for the first time pentanomials and eptanomials allowing fast(er) square root computation. We call such polynomials *square root friendly*. The obvious applications are to point halving methods for elliptic curves and divisor halving methods for hyperelliptic curves.
We note the existence of square root friendly trinomials of a given degree when we already know that an irreducible trinomial of the same degree exists, and formulate a conjecture on the degrees of the terms of square root friendly polynomials. Following similar results by Bluher, we also give a partial result that goes in the direction of the conjecture. We consider irreducible polynomials $p(X)$ such that the square root ζ of a zero x of $p(X)$ is a sparse polynomial and characterize those for which ζ has minimal degree.
We also show how to improve the speed of solving quadratic equations and that the increase in the time required to perform modular reduction is marginal and does not affect performance adversely. Experimental results confirm that the new polynomials mantain their promises. Point halving gets a speed-up of 20% and the performance of scalar multiplication based on point halving is improved by at least 11%.

Keywords: *Binary fields, Polynomial basis, Square root extraction, Trace computation, Quadratic equations, Point halving, Divisor halving.*

1 Introduction

The main topic of this paper is square root extraction in binary fields, even though also other operations are considered.

In [19] it is shown that square root extraction can be accelerated if a suitable irreducible trinomial is used to define a field extension of $\mathbb{F}_2$.

Let $p(X)$ be the irreducible polynomial of degree d used to define the extension field $\mathbb{F}_{2^d}/\mathbb{F}_2$. If the polynomial in X representing the square root

of the image x of X in $\mathbb{F}_{2^d}$ has low weight and/or degree, then general square roots can be extracted in $\mathbb{F}_{2^d}$ efficiently. In this case we call $p(X)$ *square root friendly*. (This definition will be made more precise later.)

We give here sufficient conditions for an irreducible polynomial of odd degree d to yield a low weight $\sqrt{x}$. In particular, we give examples of pentanomials and eptanomials, but in at least one case, that of $\mathbb{F}_{2^{233}}$, that can be defined by trinomials, we show how one can perform square root computations even faster than in [19].

The paper is logically divided in two parts, the first one about mathematical aspects (Section 2) and the second part about implementation issues and performance (Section 3).

As the main motivation comes from elliptic curve cryptography, in particular from point halving based methods for scalar multiplication, in § 2.1 background material about square roots is provided and in § 2.2 point and divisor halving are recalled. In § 2.3 we introduce our family of polynomials and discussed related polynomials. Polynomials for several useful (and used in practice) binary fields are presented in § 2.4, together with observations about the scarcity of other types of polynomials with analogous properties, a result about the existence of square root friendly trinomials, a conjecture about the degrees of the non-leading terms of square root friendly polynomials, and a theorem that supports the conjecture itself.

Finally, we move to practical aspects. Not only square root extraction is implemented (§ 3.1), but also the computation of traces over $\mathbb{F}_2$ (§ 3.2) and solving quadratic equations in $\mathbb{F}_{2^d}$ (§ 3.3). These routines are benchmarked showing gains, whereas multiplication and squaring either experience a negligible slowdown or even a minimal speedup. The costs of various elliptic curve group operations and scalar multiplication algorithms using different reduction polynomials for the definition field are also given. The discussion of these results (§ 3.4) also provides the conclusion to the paper. In particular, we prove that a performance gain of around 20% can be expected for the point halving alone, with a speed increase in excess of 11% for scalar multiplication.

2 Mathematics

2.1 Background on Polynomial Bases and Square Roots

Let $p(X)$ be an irreducible polynomial of *odd* degree d, and the field $\mathbb{F}_{2^d}$ be constructed as $\mathbb{F}_2[X]/(p(X))$. Let x be the image of X in $\mathbb{F}_{2^d}$.

We consider here polynomial basis representation because we are solely concerned with software applications.

Whereas with a normal basis [4] a square root computation is just a shift of the bits internal representation of the field element by one position, whose cost is basically negligible with respect to that of a multiplication, matters are different with polynomial bases. Even the cost of a squaring becomes no longer negligible. Indeed, if $\alpha = \sum_{i=0}^{d-1} a_i x^i$, then $\alpha^2 = \sum_{i=0}^{d-1} a_i x^{2i}$ need to be reduced modulo $p(X)$. The costs are low, but cannot be completely ignored.

Things are more complicated for square roots. Even though squaring just consists in "spacing" the bits of the original element with zeros, the bits of a generic field element cannot be just "compressed". Using Fermat's little theorem $\alpha^{2^d} = \alpha$, we can compute $\sqrt{\alpha} = \alpha^{2^{d-1}}$. This requires $d-1$ squarings, and for large d the cost is that of several field multiplications. In fact, it can be faster to change to a normal basis (we need only to store the change-of-base matrix: the conversion time is then approximately equivalent to one field multiplication, and this method requires to store $O(d^2)$ bits [21] – see also [18]), perform the square root extraction in that representation, then convert back to polynomial basis.

A more efficient method stems from the observation that $\sqrt{\alpha}$ can be expressed in terms of $\zeta := \sqrt{x}$. If

$$\alpha = \sum_{i=0}^{d-1} a_i x^i$$

we separate the even exponents from the odd exponents

$$\alpha = \sum_{i=0}^{(d-1)/2} a_{2i} x^{2i} + \sum_{i=0}^{(d-3)/2} a_{2i+1} x^{2i+1} = (\alpha_{\text{even}})^2 + x \cdot (\alpha_{\text{odd}})^2$$

where

$$\alpha_{\text{even}} = \sum\nolimits_{i=0}^{(d-1)/2} a_{2i} x^i \qquad \text{and} \qquad \alpha_{\text{odd}} = \sum\nolimits_{i=0}^{(d-3)/2} a_{2i+1} x^i$$

and, since square root in a field of even characteristic is a linear operation:

$$\sqrt{\alpha} = \alpha_{\text{even}} + \zeta \cdot \alpha_{\text{odd}} \tag{1}$$

Therefore, once ζ has been computed on a per-field basis, the computation of a generic square root is reduced to "bits extraction and packing", a "rectangular" multiplication of a degree $\leq d-1$ polynomial ζ with a polynomial $\sum_{i=0}^{(d-3)/2} a_{2i+1} x^i$ of degree $\leq (d-1)/2$ in x, and a modular reduction. Intuitively, the cost should approach a half of the cost of a field multiplication and this is confirmed by the analysis in [19, 22].

2.2 Point and Divisor Halving

Let E be an elliptic curve defined over $\mathbb{F}_{2^d}$ by a *Weierstrass equation*

$$E \ : \ Y^2 + XY = X^3 + aX^2 + b$$

with $a, b \in \mathbb{F}_{2^d}$ and having a subgroup $G \leq E(\mathbb{F}_{2^d})$ of large prime order.

Since computing the double of any given point P is the most common operation in a scalar multiplication performed by double-and-add methods, an important direction of research consists in optimizing doubling formulæ (for surveys on scalar multiplication methods and elliptic curve operations see, for example [9, Chs. 9 and 13] or [20, Ch. 3]).

Point halving [22, 26, 27], on the other hand, consists in computing a point R whose double is P, i.e. such that $2R = P$. Given a point $P \in G$, there is a unique $R \in G$ with $2R = P$. Halving is an automorphism of G.

In order to perform this operation one needs to solve a quadratic equation of the form $\lambda^2 + \lambda + \alpha = 0$ for λ, extract a square root, perform two multiplications and some additions. Note that there are two points R_1 and R_2 on the curve with $2R_1 = 2R_2 = P$. To determine which one is in G, an additional check involving a trace computation is required. We refer the reader to [22, 27, 19] for details, including the usage of halving in place of doubling in scalar multiplication algorithms. According to the analysis in [19], halving is about two times faster than doubling.

Birkner [13] has a divisor halving formula for genus two curves based on the Lange–Stevens doubling formulae [24]. Birkner and Thériault [14] deal with genus three divisors. The performance of all known halving formulæ depends (to a variable degree) on the performance of square root extraction. Further uses of point halving to speed up scalar multiplication on (elliptic) Koblitz Curves [23] are found in [7, 10] and [11].

2.3 Square-Root Friendly Polynomials

The speed of square root computation depends on the efficiency of the multiplication of a generic polynomial of degree $\leq (d-1)/2$ by $\zeta = \sqrt{x}$. If ζ is very sparse, for example of weight two or four (i.e. it has just two or four nonzero terms), then this product can be computed by a few shift and XOR operations. In [19] two types of trinomials have been shown that allow this. The kind that interests us is

$$p(X) = X^d + X^m + 1$$

with m odd. Then $x = x^{d+1} + x^{m+1}$ with $d+1$ and $m+1$ even, and

$$\zeta = x^{(d+1)/2} + x^{(m+1)/2} \ ,$$

and $p(X)$ is square root friendly. In fact, this idea is much more general.

Assume the irreducible polynomial $p(X)$ defining $\mathbb{F}_{2^d}$ over $\mathbb{F}_2$ has form

$$p(X) = X \cdot \mathcal{U}(X)^2 + 1 \tag{2}$$

where $\mathcal{U}$ is a polynomial of degree $(d-1)/2$ and even weight w (hence $p(X)$ has weight $w+1$). Then, ζ has a very simple form in $\mathbb{F}_{2^d}$: from $x^2 \cdot \mathcal{U}(x)^2 + x = 0$ we obtain

$$\zeta = x \cdot \mathcal{U}(x) \ ,$$

and ζ is represented by a polynomial of degree $1 + (d-1)/2 = (d+1)/2$ and weight w. Furthermore, note that the *polynomial* product

$$\zeta \cdot \sum_{i=0}^{(d-3)/2} a_{2i+1} x^i$$

has degree bounded by $(d+1)/2+(d-3)/2 = d-1$, therefore *no polynomial reduction is required if a square root is computed by formula* (1). Hence, irreducible polynomials of form (2) are square root friendly.

Note that the low degree of ζ not only guarantees that no modular reduction is necessary, but puts also a bound on the complexity of the multiplication by ζ (even though its sparseness has an even bigger influence). It is therefore interesting to ask whether there are irreducible polynomials $p(X)$ that lead to a ζ of even lower degree. This cannot happen, and requiring the degree of the polynomial representing ζ to be at most $(d+1)/2$ in fact almost characterizes the polynomials (2).

Theorem 1. *Let the field extension $\mathbb{F}_{2^d}/\mathbb{F}_2$ be defined by an irreducible polynomial $p(X)$ of odd degree d, and let x be the image of X in $\mathbb{F}_{2^d}$. Suppose that $\zeta := \sqrt{x}$ is a polynomial of degree at most $(d+1)/2$ in x.*

Then the degree of ζ is exactly $(d+1)/2$ and either

(i) $p(X) = 1 + X \cdot \mathcal{U}(X)^2$ and $\zeta = x \cdot \mathcal{U}(x)$ for some $\mathcal{U}(X) \in \mathbb{F}_2[X]$ of degree exactly $(d-1)/2$; or

(ii) $p(X) = 1 + (X+1) \cdot X^2 \cdot \mathcal{W}(X)^2$ and $\zeta = 1 + (x+1) \cdot x \cdot \mathcal{W}(x)$ for some $\mathcal{W}(X) \in \mathbb{F}_2[X]$ of degree exactly $(d-3)/2$.

Proof. Let $\zeta = \mathcal{V}(x)$ where $\mathcal{V}(X) \in \mathbb{F}_2[X]$. Suppose that $\deg \mathcal{V} \leq (d-1)/2$. Then the relation $\zeta = \mathcal{V}(x)$ implies $\mathcal{V}(x)^2 + x = 0$ and thus x would be a zero of the polynomial $\mathcal{V}(X)^2 + X$ of degree at most $d-1$, contradicting the fact that x has degree d.

From now on let $\deg \mathcal{V} = (d+1)/2$.

If $X|\mathcal{V}(X)$, then x is a root of $1 + \frac{\mathcal{V}(X)^2}{X}$ that has degree d. Therefore this polynomial is $p(X)$ and we are in the first case with $\mathcal{U}(X) = \mathcal{V}(X)/X$.

Now let $X \nmid \mathcal{V}(X)$, which implies $\mathcal{V}(0) = 1$. Since x is a root of $X + \mathcal{V}(X)^2$ it is $p(X)|X + \mathcal{V}(X)^2$ and and $(X+1) \cdot p(X) = (X + \mathcal{V}(X)^2)$. We add $X + 1$ to both sides of the last equality and exploit the linearity of squaring to get $(X+1) \cdot (1+p(X)) = (1+\mathcal{V}(X))^2$.. In turn this implies that $X + 1$ divides $1 + \mathcal{V}(X)$, and since $X|(1 + p(X))$, also X divides $1 + \mathcal{V}(X)$. Hence $\mathcal{V}(X) = 1 + (X + 1) \cdot X \cdot \mathcal{W}(X)$ for some polynomial $\mathcal{W}(X) \in \mathbb{F}_2[X]$ and we are in the second case. □

Definition 1. *An irreducible polynomial of form* (2) *is called a* special square root friendly (SSRF) polynomial of type I.

An irreducible polynomial

$$p(X) = 1 + (X + 1) \cdot X^2 \cdot \mathcal{W}(X)^2 \tag{3}$$

for some $\mathcal{W}(X) \in \mathbb{F}_2[X]$ *of degree exactly* $(d-3)/2$ *is called a* special square root friendly polynomial of type II.

Example 1. The irreducible polynomial $X^{163} + X^{57} + X^{49} + X^{29} + 1$ is of type I, and the corresponding ζ has weight four. On the other hand, the standard NIST polynomial [25] $X^{163} + X^7 + X^6 + X^3 + 1$ defines a ζ of weight 79 (cf. Appendix A).

We do not know whether there are other families of irreducible polynomials which are not trinomials, not of one of the forms (2) or (3), and for which $\sqrt{x}$ has small weight. Even for trinomials $X^d + X^m + 1$ with even m one has to check on a case by case basis, but examples are known [19].

Still, SSRF polynomials abound: see § 2.4.

Remark 1. Changing polynomial is easy without introducing incompatibilities in cryptographic applications: we just change the base used for representation of the field elements before and after the bulk of the computation. In general the cost of one conversion is, as as for the conversion between normal and polynomial bases, about one field multiplication. Therefore this overhead is essentially negligible with respect to the full cost of a scalar multiplication, that is in the order of magnitude of hundreds to thousands of field multiplications (see for example § 5.3 of [5]).

Remark 2. The cost of a square root extraction implemented by using the sparse version of ζ offered by the above polynomials can be roughly estimated using, for example, already published results. For example in [19],

Example 3.12, the NIST-recommended trinomial

$$p(X) = X^{233} + X^{74} + 1 \tag{4}$$

for the finite field $\mathbb{F}_{2^{233}}$ is used. Even though the term X^{74} does not have an odd exponent, ζ has a sparse representation

$$\zeta = (x^{32} + x^{117} + x^{191})(x^{37} + 1) \ .$$

By this, finding a root via equation (1) requires roughly 1/8 of the time of a field multiplication. As we shall show in the next section we can choose

$$p(X) = X^{233} + X^{159} + 1$$

and in this case

$$\zeta = x^{117} + x^{80} \ .$$

The number of shift operations and XOR operations required to multiply by ζ is thus reduced. Furthermore, as already remarked, there is no need to perform a reduction modulo $p(X)$ while with non-square root friendly polynomials this is always necessary, even with the polynomial (4). Implementation results show the cost of a square root to be less than 9% of that of a multiplication. See § 3.4 for precise results.

Remark 3. Similar formulæ for cube root computations are found in [1] – their results are easily partially generalised to any odd characteristic.

Remark 4. Type I SSRF polynomials enjoy another very useful property. In [2] it is proved: *If $\mathbb{F}_{2^d}$ is defined by an irreducible polynomial $p(X)$ whose non constant terms all have odd exponents, then the trace of a field element α represented as $\sum_{i=0}^{d-1} a_i x^i$ with respect to the polynomial basis induced by $p(X)$ is its constant term a_0.* In other words, the only trace-one element in the polynomial basis defined by $p(X)$ is 1. This is very important in the applications: for instance, as remarked in § 2.2, a trace computation is necessary to correctly halve a point. It is especially fortunate that the same family of polynomials makes both square roots *and* trace computations faster. Type II SSRF do not enjoy this property, however they are not relevant for the applications, as we shall see at the beginning of the next Subsection.

2.4 Existence and Degrees of the Terms

SSRF polynomials are easy to find. For example, for extension degree $d = 163$, a simple MAGMA [16] script determined all 713 (special) type I SSRF pentanomials of degree 163 in less than one minute.

But, there are no type II SSRF pentanomials of degree 163. The first (in lexycographic order) eptanomial of degree 163 is $X^{163}+X^{162}+X^{59}+X^{58}+X^3+X^2+1$ However, Type II SSRF trinomials and pentanomials exist. For type II, $1+X^{126}+X^{127}$ is a type II trinomial with $\zeta = 1 + x^{63}+x^{64}$. For fields not defined by trinomials, we know only a few type II pentanomials. For degree 43, for example, we have $X^{43}+X^{42}+X^{29}+X^{28}+1$, for which of course $\zeta = x^{22}+x^{21}+x^{15}+x^{14}+1$. For prime degrees under 500 for which no irreducible trinomial exists (but pentanomials exist), the only degrees for which type II SSRF pentanomials exist are: 13, 43, 59, 101, 109, 157, 173, 181, 269, 311, 397, 461, and 491.

The difficulty in finding type II SSRF polynomials is not a concern. For the applications, the most important SSRF polynomials are those of Type I, because in general they have reasonably low degree sediment (the sediment of an univariate is obtained by removing its leading term). A type II SSRF polynomial of degree d has form $X^d + X^{d-1} + \ldots + 1$ and in general reduction modulo these polynomials is not efficient.

One way of finding good ones would be to try to find a type II SSRF polynomial $p(X)$ of maximum weight, for which $(X+1)\cdot p(X)$ has a low weight, and mimic the reduction procedures explained in [3]. Since pentanomials are always found, this approach is interesting if we find $(X+1)\cdot p(X)$ of weight four. If

$$p_{d,m}(X) = 1 + \sum_{\substack{1\le i\le d-1 \\ i\neq m}} X^{2i} + X^{2i+1}$$

for any d and $\leq m \leq (d-3)/2$, we see that $(X+1)\cdot p_{d,m}(X)$ has weight four only if $m = 1$. Then $p_{d,1}(X)$ is irreducible for $d = 31$, 97, 151, 577, 7879 and for no other degree below 10000. In the first four cases, irreducible trinomials exist, and degree 7879 requires a pentanomial.

In Table 1 we list (Type I) SSRF polynomials of several degrees. The degrees have been taken from the NIST list of recommended binary curves and from the extension degrees used in [12]. All these extension degrees are interesting because they are either used in standards for elliptic curve cryptography or they represent good choices for extension degrees for defining hyperelliptic curve for cryptographic applications.

When no trinomial is available, a pentanomial is listed. We always report the polynomial with least degree sediment. Only in a handful of cases is the SSRF polynomial with least degree sediment the same as the standard one, i.e. the irreducible polynomial with least degree sediment but without the restriction on being square root friendly.

Degree	Irreducible tri/pentanomial	$\zeta = \sqrt{x}$	Standard?
47	$X^{47} + X^5 + 1$	$x^{24} + x^3$	Yes
53	$X^{53} + X^{19} + X^{17} + X^{15} + 1$	$x^{27} + x^{10} + x^9 + x^8$	No
59	$X^{59} + X^{21} + X^{17} + X^{15} + 1$	$x^{30} + x^{11} + x^9 + x^8$	No
67	$X^{67} + X^{25} + X^{17} + X^5 + 1$	$x^{34} + x^{13} + x^9 + x^3$	No
71	$X^{71} + X^9 + 1$	$x^{36} + x^5$	No
73	$X^{73} + X^{25} + 1$	$x^{37} + x^{13}$	Yes
79	$X^{79} + X^9 + 1$	$x^{40} + x^5$	Yes
83	$X^{83} + X^{29} + X^{25} + X^3 + 1$	$x^{42} + x^{15} + x^{13} + x^2$	No
89	$X^{89} + X^{51} + 1$	$x^{45} + x^{26}$	No
97	$X^{97} + X^{33} + 1$	$x^{49} + x^{17}$	No
101	$X^{101} + X^{35} + X^{31} + X^3 + 1$	$x^{51} + x^{18} + x^{16} + x^2$	No
107	$X^{107} + X^{37} + X^{33} + X^{23} + 1$	$x^{54} + x^{19} + x^{17} + x^{12}$	No
109	$X^{109} + X^{43} + X^{41} + X^{23} + 1$	$x^{55} + x^{22} + x^{21} + x^{12}$	No
127	$X^{127} + X + 1$	$x^{64} + x$	Yes
131	$X^{131} + X^{45} + X^{41} + X^9 + 1$	$x^{66} + x^{23} + x^{21} + x^5$	No
137	$X^{137} + X^{21} + 1$	$x^{69} + x^{11}$	Yes
139	$X^{139} + X^{53} + X^{33} + X^{25} + 1$	$x^{70} + x^{27} + x^{17} + x^{13}$	No
149	$X^{149} + X^{51} + X^{47} + X^9 + 1$	$x^{75} + x^{26} + x^{24} + x^5$	No
157	$X^{157} + X^{55} + X^{47} + X^{11} + 1$	$x^{79} + x^{28} + x^{24} + x^6$	No
163	$X^{163} + X^{57} + X^{49} + X^{29} + 1$	$x^{82} + x^{29} + x^{25} + x^{15}$	No
179	$X^{179} + X^{61} + X^{57} + X^{41} + 1$	$x^{90} + x^{31} + x^{29} + x^{21}$	No
199	$X^{199} + X^{67} + 1$	$x^{100} + x^{34}$	No
211	$X^{211} + X^{73} + X^{69} + X^{35} + 1$	$x^{106} + x^{37} + x^{35} + x^{18}$	No
233	$X^{233} + X^{159} + 1$	$x^{117} + x^{80}$	No
239	$X^{239} + X^{81} + 1$	$x^{120} + x^{41}$	No
251	$X^{251} + X^{89} + X^{81} + X^3 + 1$	$x^{126} + x^{45} + x^{41} + x^2$	No
269	$X^{269} + X^{91} + X^{87} + X^{61} + 1$	$x^{135} + x^{46} + x^{44} + x^{31}$	No
283	$X^{283} + X^{97} + X^{89} + X^{87} + 1$	$x^{142} + x^{49} + x^{45} + x^{44}$	No
409	$X^{409} + X^{87} + 1$	$x^{205} + x^{44}$	Yes
571	$X^{571} + X^{193} + X^{185} + X^5 + 1$	$x^{286} + x^{97} + x^{93} + x^3$	No

Table 1. Some special square root friendly trinomials and pentanomials.

We first note that if an irreducible trinomial of a given degree exists, then we can always find one which is square root friendly.

Theorem 2. *Let d be an odd positive integer. If an irreducible trinomial $p(X) = X^d + X^m + 1$ over $\mathbb{F}_2$ of degree d exists, then $p(X)$ can be chosen of form* (2)*, i.e. where the exponent m of the middle term is odd.*

Proof. Let $X^d + X^m + 1$ be an irreducible trinomial with $d > m > 0$ and m even. Then it is easy to prove that the polynomial $X^d + X^{d-m} + 1$ is also irreducible – but $d - m$ is odd. In fact, let $q(X)$ be a monic polynomial over $\mathbb{F}_2$ with $q(X) = 1$, i.e. non-vanishing constant term. Let $q^*(X) = X^{\deg q} q(X^{-1})$ be the *reciprocal polynomial* of $q(X)$. It ie easy to see that q^* is a monic polynomial with non-vanishing constant term. Then, a factorization $q(X) = g(X)h(X)$ implies $q^*(X) = g^*(X)h^*(X)$.

Applying this result to $q(X) = X^d + X^{d-m} + 1$ proves that it must be irreducible, otherwise $p(X) = q^*(X)$ would be also reducible. □

Existence results for pentanomial-defined fields are still an open question. However, on the basis of Table 1 and further experimental results, we found further evidence for an observation of Ahmadi and Menezes: In [2] they list irreducible pentanomials $p(X)$ having only non-constant terms with odd exponent for which the sediment has lowest degree, and observe that if $d \equiv \pm 3 \pmod 8$, the degree of the sediment is at least $d/3$, whereas if $d \equiv \pm 1 \pmod 8$, then the degree of the sediment is usually quite small. In fact, if a trinomial exists for degree d, then a square-root friendly pentanomial with a sediment of degree much lower than $d/3$ usually also exists.

For the extension degrees for which there are no trinomials we computed not only the SSRF pentanomials but also the *eptanomials* with smallest degree sediment – the idea was, that perhaps one can find good eptanomials with a lower degree sediment than the best pentanomials, to improve modular reduction: they are given in Table 2. Similar searches for polynomials with nine and eleven terme have been performed. We immediately observe here that sediment degree differences are very limited, so the eptanomials do not bring advantages. The same observation applies to polynomials with nine or eleven terms. A pattern in the distribution of degrees of the second term of the sediment of SSRF polynomials up to degree 3000 and with up to eleven terms prompts us to formulate the following conjecture:

Conjecture 1. Let d be an odd positive integer, and c be the minimum of the degrees of the sediments of all square root friendly polynomials of degree d. Further, let c' be the minimum of the degrees of the second highest degree term of all the sediments of degree c of square root friendly polynomials of degree d. Then

$$3\,c - d = c - c' = \begin{cases} 8 & \text{if } d \equiv 1 \pmod 3 \\ 4 & \text{if } d \equiv 2 \pmod 3 \end{cases}.$$

A first result in this direction has already been proved by Bluher [15] using a result of Swan [28] (that in fact goes back to Stickelberger). Her result is: *The odd degree polynomial* $p(X) = X^d + \sum_{i \in \mathcal{S}} X^i + 1$ *in* $\mathbb{F}_2[X]$, *where* $\mathcal{S} \subset \{i : i \text{ odd}, 0 < i < d/3\} \cup \{i : i \equiv d \pmod 4, 0 < i < d\}$ *has no repeated roots; if* $d = \pm 1 \pmod 8$, *then* f *has an odd number of irreducible factors; and if* $d = \pm 3 \pmod 8$, *then* f *has an even number of*

Degree	Irreducible eptanomial	$\zeta = \sqrt{x}$
53	$X^{53} + X^{19} + X^{15} + X^5 + X^3 + X + 1$	$x^{27} + x^{10} + x^8 + x^3 + x^2 + x$
59	$X^{59} + X^{21} + X^{17} + X^{13} + X^3 + X + 1$	$x^{30} + x^{11} + x^9 + x^7 + x^2 + x$
67	$X^{67} + X^{25} + X^{17} + X^7 + X^3 + X + 1$	$x^{34} + x^{13} + x^9 + x^4 + x^2 + x$
83	$X^{83} + X^{29} + X^{25} + X^7 + X^5 + X^3 + 1$	$x^{42} + x^{15} + x^{13} + x^4 + x^3 + x^2$
101	$X^{101} + X^{35} + X^{31} + X^9 + X^7 + X + 1$	$x^{51} + x^{18} + x^{16} + x^5 + x^4 + x$
107	$X^{107} + X^{37} + X^{33} + X^{15} + X^9 + X^7 + 1$	$x^{54} + x^{19} + x^{17} + x^8 + x^5 + x^4$
109	$X^{109} + X^{39} + X^{31} + X^9 + X^5 + X^3 + 1$	$x^{55} + x^{20} + x^{16} + x^5 + x^3 + x^2$
131	$X^{131} + X^{45} + X^{41} + X^{13} + X^9 + X + 1$	$x^{66} + x^{23} + x^{21} + x^7 + x^5 + x$
139	$X^{139} + X^{49} + X^{41} + X^7 + X^5 + X^3 + 1$	$x^{70} + x^{25} + x^{21} + x^4 + x^3 + x^2$
149	$X^{149} + X^{51} + X^{47} + X^9 + X^7 + X + 1$	$x^{75} + x^{26} + x^{24} + x^5 + x^4 + x$
157	$X^{157} + X^{55} + X^{47} + X^{15} + X^9 + X^3 + 1$	$x^{79} + x^{28} + x^{24} + x^8 + x^5 + x^2$
163	$X^{163} + X^{57} + X^{49} + X^{15} + X^9 + X + 1$	$x^{82} + x^{29} + x^{25} + x^8 + x^5 + x$
179	$X^{179} + X^{61} + X^{57} + X^{13} + X^9 + X^5 + 1$	$x^{90} + x^{31} + x^{29} + x^7 + x^5 + x^3$
211	$X^{211} + X^{73} + X^{65} + X^{13} + X^{11} + X^3 + 1$	$x^{106} + x^{37} + x^{33} + x^7 + x^6 + x^2$
251	$X^{251} + X^{85} + X^{81} + X^7 + X^5 + X^3 + 1$	$x^{126} + x^{43} + x^{41} + x^4 + x^3 + x^2$
269	$X^{269} + X^{91} + X^{87} + X^{15} + X^{13} + X^{11} + 1$	$x^{135} + x^{46} + x^{44} + x^8 + x^7 + x^6$
283	$X^{283} + X^{97} + X^{89} + X^{13} + X^9 + X + 1$	$x^{142} + x^{49} + x^{45} + x^7 + x^5 + x$
571	$X^{571} + X^{193} + X^{185} + X^{15} + X^{11} + X^3 + 1$	$x^{286} + x^{97} + x^{93} + x^8 + x^6 + x^2$

Table 2. Some special square root friendly eptanomials.

irreducible factors. In fact, by adapting the proof given in [3] of Bluher's result, it is possible to prove the following theorem:

Theorem 3. *Let d be an odd positive integer and the polynomial $p(X) \in \mathbb{F}_2[x]$ have degree d and satisfy one of the following conditions:*

1. *If $d \equiv 1 \pmod 3$, then $p(X)$ is either of the form*
   - $X^d + X^{\frac{d+8}{3}} + \sum_{j \in J} X^j + 1$ *where* $J = \left\{ j \,:\, j \text{ odd},\ 1 \le j < \frac{d-19}{3} \right\}$
   - *or* $X^d + \sum_{j \in J'} X^j + 1$ *where* $J' = \left\{ j \,:\, j \text{ odd},\ 1 \le j \le \frac{d+5}{3} \right\}$;
2. *If $d \equiv 2 \pmod 3$, then $p(X)$ is either of the form*
   - $X^d + X^{\frac{d+4}{3}} + \sum_{j \in J} X^j + 1$ *where* $J = \left\{ j \,:\, j \text{ odd},\ 1 \le j \le \frac{d-11}{3} \right\}$
   - *or* $X^d + \sum_{j \in J'} X^j + 1$ *where* $J' = \left\{ j \,:\, j \text{ odd},\ 1 \le j \le \frac{d+1}{3} \right\}$.

Then $p(X)$ is square-free. Furthermore, if $d \equiv \pm 1 \bmod 8$ then $p(X)$ has an odd number of irreducible factors, whereas $d \equiv \pm 3 \bmod 8$ then $p(X)$ has an even number of irreducible factors and therefore must be reducible.

Remark 5. We already mentioned in Remark 4 that in [2] the polynomials we study are proven to permit efficient computation of traces. In Conjecture 9 of [2] it is also speculated that if an irreducible pentanomial of degree d exists, then an irreducible pentanomial of the same degree defining a polynomial basis with only one trace-one element also exists. In the similar vein we could try to restrict our Conjecture 1 to pentanomials, but we have the following example for degree 1987:

$$X^{1987} + X^{665} + X^{661} + X^{549} + 1 \ .$$

This polynomial has minimal degree sediment, and the sediment has minimal degree second term among all irreducible pentanomials with minimal degree sediment. We have $d \equiv 1 \pmod 3$ and $3\,c - d = 8$, but the second term of the sediment has degree 661, not 657. On the other hand, with eptanomials we find following irreducible

$$X^{1987} + X^{665} + X^{657} + X^{25} + X^{21} + X^{9} + 1 \ ,$$

and we know of no other irreducibles of the form

$$X^{1987} + X^{c} + X^{c'} + \ \textit{other lower odd degree terms} \ + 1$$

with c smaller than 665 or with $c = 665$ and c' smaller than 657.

3 Practical Aspects

For several binary fields $\mathbb{F}_{2^d}$ we implemented a few operations, including (but not only) multiplication, squaring, square roots, and also trace and half-trace computations (the latter is used for solving quadratic equations). For multiplication and squaring we implemented generic routines as well as routines that have been optimized for each degree, by (partially) unrolling all loops and avoiding superfluous operations, especially those involving most significant bits of operands that are known to be zero. For example, the routines for 83 and 89 bit fields, as well as those for 193 and 191 bit fields are different and have different performance, despite the fact that in the first two examples the inputs all fit in 3 words of 32 bits, and in the second they all fit in 6 words. Our implementation was written in the `C` programming language, the compiler used was `gcc` version 4.0.1. In fact the new routines are an extension of the library described in [8] (some performance discrepances are due to the fact that some other routines have been improved in the meantime).

Depending on the nature of the chosen reduction polynomial we implemented some routines such as modular reduction and square root extraction in different ways. In a few cases we implement the same field twice using two different reduction polynomials in order to compare the different situations.

3.1 Extracting a Square Root

To implement square root extraction of a field element α we use formula (1): the only part that changes between different implementations is how the multiplication $\zeta \cdot \alpha_{\text{odd}}$ is realized. Furthermore, among the square root friendly polynomials we only consider the SSRF polynomials of type I.

To compute α_{even} and α_{odd} from α we use a 16-bit to 8-bit look-up table to compress the bits. Using a trick due to Robert Harley, this table is stored in 256 consecutive bytes by mapping the *input* `i` to (`i` + `i` $\gg$ 7) & `0xff`.

If the reduction polynomial $p(X)$ is not SSRF of type I, then the Hamming weight of the square root is counted at the time of initialization of the library - hence only once. If this weight is low enough then a simple routine that XORs together shifted copies of α_{odd} according to which bits in ζ are set is used. If the weight is higher than a certain threshold, then a comb binary polynomial multiplication method is used, where the precomputations relative to ζ are performed only once, at initialization time. A reduction modulo $p(X)$ is then performed, if necessary.

By means of this we can always keep the time of the multiplication $\zeta \cdot \alpha_{\text{odd}}$ to just under a half of the time of a *generic* field multiplication: Note, however, that this time can be substantially higher than the time required for the per-field, ad-hoc optimized implementation of multiplication, as our experimental results in § 3.4 show.

In the case where the degree d reduction polynomial $p(X)$ is square root friendly, we also have a simple routine that XORs together shifted copies of α_{odd} according to which bits in ζ are set, but since the weight is very low and known, the routine can be unrolled completely. Furthermore, as already mentioned, in this case no modular reduction is necessary. This routine is the one that delivers the best performance.

3.2 Trace Computation

We follow here [19] and [2]. For generic binary fields, to compute the trace from $\mathbb{F}_{2^d}$ to $\mathbb{F}_2$ we use the fact that the trace is linear. In other words, once we have computed $\text{Tr}(x^i)$ for all i with $0 \leq i < d$, in order to compute

the trace of an element $\alpha \in \mathbb{F}_{2^d}$, $\alpha = \sum_{i=0}^{d-1} a_i x^i$ where the $a_i \in \mathbb{F}_2$, we have

$$\mathrm{Tr}\,(\alpha) = \sum_{i=0}^{d-1} a_i \,\mathrm{Tr}(x^i) \ .$$

The latter sum can be implemented by a componentwise multiplication (hence, a logical `AND` operation) of the bit vector representing the element α with a bit vector whose i-th component (starting with the zeroth component!) is the trace of x^i, called the *trace vector*, followed by a bit count modulo 2. Both operations can be performed very efficiently. The trace vector is of course computed once for each field.

However, when a square root friendly polynomial is used to represent the field $\mathbb{F}_d$ with d odd, the trace vector contains just one bit set, namely the least significant bit, and $\mathrm{Tr}\,(\alpha) = a_0$.

We therefore implemented two different routines: the first one uses a trace vector and the second one just polls the value of a single bit.

3.3 Solving Quadratic Equations (Half-Traces)

In order to solve quadratic equations of the form

$$\lambda^2 + \lambda = \alpha \tag{5}$$

for $\lambda \in \mathbb{F}_{2^d}$ where $\alpha \in \mathbb{F}_{2^d}$ we implement just one generic routine, that takes the element c as an input as well as a precomputed table.

The *half-trace* operator on $\mathbb{F}_{2^d}$ (d odd) is defined as

$$H(\alpha) = \sum_{i=0}^{(d-1)/2} \alpha^{2^{2i}}$$

and it is easily verified that it is $\mathbb{F}_2$-linear, and that $H(\alpha)$ satisfies $H(\alpha)^2 + H(\alpha) = \alpha + \mathrm{Tr}(\alpha)$. Therefore, in order to solve equation (5) we first have to check whether $\mathrm{Tr}(\alpha) = 0$. Only in that case (5) is solvable: Then we compute the half-trace $H(\alpha)$ of α and $H(\alpha)$, $H(\alpha) + 1$ are the solutions.

One optimization consists in removing the coefficients of even powers of x as detailed in [19]: We write $H(\alpha) = H(\alpha') + \beta$ where α' has fewer nonzero coefficients than α. This can be done by observing that $H(x^{2i}) = H(x^i) + x^{2i} + \mathrm{Tr}(x^{2i})$. We can thus reduce the amount of stored half-traces of powers of x by half.

Our approach differs from the one in [19] in that we do not make ad-hoc attempts to minimize memory requirements. Instead, we reduce

Field	Reduction	Sqrt	Operation Timings (μsec)						Costs relative to one Mul				
(Bits)	Polynomial	Frnd	Mul	Sqr	Inv	Sqrt	Trace	Eq	Sqr	Inv	Sqrt	Trace	Eq
41	41,3,0	Yes	.084	.013	.448	.015	.007	.129	.160	5.340	.178	.078	1.536
43	43,6,4,3,0	No	.090	.016	.453	.327	.020	.128	.179	5.010	3.619	.227	1.422
43	43,17,9,5,0	Yes	.090	.017	.454	.018	.007	.126	.186	5.029	.196	.072	1.400
47	47,5,0	Yes	.089	.013	.478	.015	.007	.128	.150	5.344	.167	.083	1.438
83	83,7,4,2,0	No	.212	.030	.915	.383	.023	.191	.141	4.313	1.806	.107	.901
83	83,29,25,3,0	Yes	.214	.045	.925	.054	.007	.192	.209	4.316	.254	.034	.897
89	89,38,0	No	.253	.023	.958	.354	.023	.200	.089	3.782	1.398	.092	.791
89	89,51,0	Yes	.253	.024	.958	.025	.007	.201	.093	3.782	.098	.027	.794
97	97,6,0	No	.311	.028	1.598	.401	.025	.256	.091	5.139	1.289	.081	.823
97	97,33,0	Yes	.304	.024	1.576	.028	.006	.252	.080	5.191	.091	.020	.829
127	127,1,0	Yes	.418	.053	1.814	.062	.007	.288	.127	4.345	.149	.016	.689
163	163,7,6,3,0	No	.819	.086	5.317	.679	.033	.388	.105	6.491	.829	.040	.474
163	163,57,49,29,0	Yes	.821	.090	5.327	.095	.007	.391	.110	6.491	.116	.009	.476
191	191,9,0	Yes	.896	.083	5.804	.093	.008	.429	.093	6.474	.104	.009	.479
223	223,33,0	Yes	1.198	.098	12.865	.113	.006	.501	.082	10.737	.094	.005	.418
233	233,74,0	No	1.336	.101	14.599	.918	.037	.582	.076	10.925	.687	.028	.436
233	233,159,0	Yes	1.330	.090	14.593	.117	.006	.584	.068	10.971	.088	.005	.439

Table 3. Operations in some Binary Fields on a PowerPC G4 running at 1.5 Ghz. Mul, Sqr, Inv, Sqrt, Tr and Eq denote multiplication, squaring, field inversion, square root extraction, trace and half-trace computation respectively.

the number of half-traces to be accumulated by increasing the amount of precomputations. We compute and store $H(\ell_0 x^{8i+1} + \ell_1 x^{8i+3} + \ell_2 x^{8i+5} + \ell_3 x^{8i+7})$ for all $i \geq 0$ such that $8i + 1 \leq d - 2$ and all $(\ell_0, \ell_1, \ell_2, \ell_3) \in \mathbb{F}_2^4 \setminus \{(0, 0, 0, 0)\}$ such that the degree of the argument of H is at most $d-2$. By means of this we reduce by a factor 32/15 the expected number of table lookups and additions of half-traces.

3.4 Results, Conclusions

We benchmarked our implementation. Table 3 collects the timings of our routines on a PowerPC G4 running at 1.5 Ghz. (Table 4 shows the timings of the same 32-bit code on on an Intel Core 2 Duo CPU running at 1.83 Ghz.) Several field operations are timed, and the costs relative to a field multiplication are provided. The reduction polynomials are given as well as whether the considered polynomial is square root friendly or not.

We have chosen degrees in various more or less evenly spaced ranges to fulfil the following requirements: the "classical" extension degrees 163, 191, and 233 must be included, fields around integer submultiples of these sizes should also be taken into account (that's the reason for degrees in the ranges 40-50 and 80-100, and 127), and we should provide examples of fields where the "standard" polynomial (either in the sense that it comes from standard documents, or that it is the most common one used for

Field	Reduction	Sqrt	Operation Timings (μsec)						Costs relative to one Mul				
(Bits)	Polynomial	Frnd	Mul	Sqr	Inv	Sqrt	Trace	Eq	Sqr	Inv	Sqrt	Trace	Eq
41	41,3,0	Yes	.061	.012	1.367	.020	.009	.067	.203	22.569	.338	.154	1.108
43	43,6,4,3,0	No	.065	.017	1.383	.227	.018	.067	.257	21.214	3.486	.271	1.029
43	43,17,9,5,0	Yes	.067	.017	1.405	.024	.009	.067	.250	20.950	.361	.139	1.000
47	47,5,0	Yes	.063	.012	1.491	.021	.009	.067	.194	23.532	.338	.147	1.059
83	83,7,4,2,0	No	.184	.037	2.744	.277	.017	.126	.198	14.877	1.503	.091	.685
83	83,29,25,3,0	Yes	.181	.036	2.752	.034	.009	.126	.198	15.195	.185	.049	.698
89	89,38,0	No	.203	.035	2.869	.265	.017	.133	.172	14.103	1.302	.082	.654
89	89,51,0	Yes	.205	.040	2.849	.037	.010	.132	.197	13.923	.180	.049	.645
97	97,6,0	No	.224	.031	3.690	.284	.018	.187	.137	16.448	1.267	.081	.831
97	97,33,0	Yes	.235	.049	3.651	.037	.009	.185	.209	15.536	.157	.038	.787
127	127,1,0	Yes	.308	.038	4.378	.043	.009	.217	.124	14.225	.140	.030	.703
163	163,7,6,3,0	No	.468	.067	6.891	.449	.019	.317	.142	14.734	.960	.040	.677
163	163,57,49,29,0	Yes	.473	.070	6.891	.073	.009	.315	.149	14.560	.154	.020	.665
191	191,9,0	Yes	.658	.059	7.609	.063	.007	.354	.089	11.568	.096	.011	.538
223	223,33,0	Yes	.854	.066	10.846	.078	.009	.420	.077	12.699	.092	.010	.492
233	233,74,0	No	.952	.086	12.175	.597	.021	.464	.090	12.787	.627	.022	.488
233	233,159,0	Yes	.944	.089	12.355	.081	.008	.464	.094	13.084	.086	.008	.492

Table 4. Operations in some Binary Fields on an Intel Core 2 Duo running at 1.83 Ghz.

computations in computer algebra systems) already is square root friendly as well as cases where it is not – in the latter scenario we also choose a second defining polynomial that is SSRF of type I. Furthermore, all these combination of cases should happen with trinomial defined fields as well as when pentanomials are used. The "submultiples" ranges are relevant because of Trace Zero Varieties coming from elliptic curves [17, 6], and future investigation will consider the use of point and divisor halving for these algebraic groups.

Our implementation shows some interesting results.

1. The claims made in [19] about the speed of optimized square root extraction for fields defined by suitable trinomials are extended to pentanomials. Our results show for fields of 163, 191, and 233 bits an even further reduced Sqrt/Mul ratio. The gain with respect to generic implementations of square roots ranges from 8 to 20, and it is higher for smaller fields (a recurring theme, cf. [8]).
2. Multiplication and squaring may get marginally slower because the SSRF polynomials usually do not have minimal degree sediments, and thus the reduction routine is more complex. The differences are minimal, field squaring paying a slightly higher toll than field multiplication. – the reason is due to the higher relative increase of complexity for a simple routine such as squaring, which in turn causes, for instance, less efficient register coloring in the `C` compiler. The complexity increase in the reduction routine is relatively small, see for

Field		Group Operations				Scalar Multiplication		
Degree	Sqrt Frnd	Affine Coords.		Mixed $\mathcal{A}+\mathcal{LD}$		Affine Coords.		Mixed
		Add,Dbl	Hlv	mixed A	Dbl	D & A	H & A	D & A
163	N	8.596	3.343	8.525	5.175	1550.3	834.5	1137.2
163	Y	8.601	2.601	8.550	5.200	1551.4	715.0	1141.1
191	Y	8.567	2.592	8.465	5.115	1801.3	825.0	1310.7
233	N	13.001	3.151	8.380	5.030	3369.4	1291.0	1588.6
233	Y	13.039	2.532	8.340	4.990	3379.2	1149.4	1579.8

Table 5. Costs of elliptic curve group operations and scalar multiplication relative to those of a field multiplication. Add, resp. Dbl, Hlv mean Addition, resp. Doubling, Halving, and D & A, resp. H & A means windowed scalar multiplication based on Double-and-Add, resp. Halve-and-Add. Scalar multiplications methods use affine coordinates as well as mixed affine-Lopez-Dahab coordinates.

instance Appendix B – in particular with respect to the number of machine operations required to perform a multiplication.
Sometimes a small performance improvement (presumably due to randomness) is observed. Field inversion and half-trace computation are unaffected by the choice of polynomial.

3. Computing traces with type I SSRF polynomials takes nearly negligible time.
4. In [19] computing an half-trace requires approximately 2/3 the time of a field multiplication, but for fields of the same sizes our ratios are lower than 1/2, because we use a lot of precomputations.

In Table 5 we give estimates of the costs relative to one field multiplication of elliptic curve group addition, doubling, halving and scalar multiplication using various algorithms. We have used the operations counts from [5, § 5.2], but the ratios between field operations come from our Table 3 for the Power PC G4. We can see that the use of type I SSRF polynomials has a noticeable impact on scalar multiplication performance based on point halving. Point halving alone is sped up by about 20%, and the whole scalar multiplication by about 14% for curves defined over $\mathbb{F}_{2^{163}}$ and by 11% if the base field is $\mathbb{F}_{2^{233}}$. These improvements are much larger that the difference in field multiplication performance (a 2.5 ‰ loss for $\mathbb{F}_{2^{163}}$, a 4.5 ‰ gain for $\mathbb{F}_{2^{233}}$). Similar improvements can be achieved on the Yao-like scalar multiplication algorithms from [5].

Square root friendly polynomials should be used when implementing formulæ that make heavy use of square root extraction in fields of characteristic two. For solving quadratic equations then we advise to increase the amount of precomputed half-traces to improve performance. In particular, the improvements in scalar multiplication performance based on

point halving obtained in [19] by using special trinomial can be carried over to fields defined by pentanomials. Around 20% *can be expected for the point halving alone, with an impact of* 11% *to* 14% *on the entire scalar multiplication.*

Acknowledgement. *The author is grateful to Peter Birkner, Toni Bluher, Darrel Hankerson, Alfred Menezes, and Nicolas Thériault for useful discussions. The author also wishes to thank Christian Dobrick for lending him his MacBook.*

References

1. O. Ahmadi, D. Hankerson, and A. Menezes. *Formulas for cube roots in* $\mathbb{F}_{3^m}$. Discrete Applied Math. **155** (3), 260–270, 2007.
2. O. Ahmadi, and A. Menezes. *On the number of trace-one elements in polynomial bases for* $\mathbb{F}_{2^n}$. Designs, Codes and Cryptography, **37**, 493-507, 2005.
3. O. Ahmadi, and A. Menezes. *Irreducible polynomials of maximum weight.* Utilitas Mathematica, **72**, 111–123, 2007.
4. D. W. Ash, I. F. Blake and S. Vanstone. *Low complexity normal bases.* Discrete Applied Math. **25** 191–210, 1989.
5. R. M. Avanzi. *Delaying and Merging Operations in Scalar Multiplication: Applications to Curve-Based Cryptosystems.* To appear in proceedings of SAC 2006.
6. R. M. Avanzi, and E. Cesena. *Trace Zero Varieties over Fields of Characteristic 2: Cryptographic Applications.* SAGA 2007, The first Symposium on Algebraic Geometry and its Applications. Tahiti, May 7–11, 2007.
7. R. M. Avanzi, M. Ciet, and F. Sica. *Faster Scalar Multiplication on Koblitz Curves combining Point Halving with the Frobenius Endomorphism.* Proceedings of PKC 2004, LNCS **2947**, 28–40. Springer–Verlag, 2004.
8. R. M. Avanzi and N. Thériault, *Effects of Optimizations for Software Implementations of Small Binary Field Arithmetic.* To appear in *Proceedings of WAIFI 2007, International Workshop on the Arithmetic of Finite Fields*, Madrid, Spain. June 21-22, 2007. 18 Pages.
9. R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *The Handbook of Elliptic and Hyperelliptic Curve Cryptography.* CRC Press, 2005.
10. R. M. Avanzi, C. Heuberger, and H. Prodinger. *Scalar Multiplication on Koblitz Curves Using the Frobenius Endomorphism and its Combination with Point Halving: Extensions and Mathematical Analysis.* Algorithmica **46** (2006), 249–270
11. R. M. Avanzi, C. Heuberger, and H. Prodinger. *On Redundant* τ*-adic Expansions and Non-Adjacent Digit Sets.* To appear in proceedings of SAC 2006.
12. R. M. Avanzi, N. Thériault, and Z. Wang. *Rethinking Low Genus Hyperelliptic Jacobian Arithmetic over Binary Fields: Interplay of Field Arithmetic and Explicit Formulæ.* CACR Technical Report 2006-07.
13. P. Birkner. *Efficient Divisor Class Halving on Genus Two Curves.* To appear in: Proceedings of Selected Areas in Cryptography – SAC 2006. Springer Verlag LNCS.
14. P. Birkner, and N. Thériault. *Efficient Divisor Class Doubling and Halving on Genus Three Curves.* In preparation.
15. A.W. Bluher. *A Swan-like Theorem.* Finite Fields and Their Applications **12**, 128–138, 2006.
16. Wieb Bosma, John Cannon, and Catherine Playoust, *The* Magma *Algebra System I: The User Language*, J. Symbolic Comput. **24** (1997), 235–265.

17. E. Cesena. *Varietá a Traccia Zero su Campi Binari: Applicazioni Crittografiche. (Trace Zero Varieties over Binary Fields: Cryptographic Applications.)* Master's Thesis. Universitá degli Studi di Milano. 2005. In Italian.
18. J.-S. Coron, D. M'Raïhi, and C. Tymen. *Fast generation of pairs* $(k, [k]P)$ *for Koblitz elliptic curves.* In: *Proceedings of SAC 2001*, LNCS **2259**, 151–164. Springer, 2001.
19. K. Fong, D, Hankerson, J, López, A, Menezes:. *Field Inversion and Point Halving Revisited.* IEEE Trans. Computers 53(8), 1047–1059, 2004.
20. D. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to elliptic curve cryptography.* Springer–Verlag, 2003.
21. B.S. Kaliski Jr. and T.L. Yin. *Storage-efficient finite field basis conversion.* In: *Proceedings of SAC98.* LNCS **1556**, Springer, 1998.
22. E. W. Knudsen. *Elliptic Scalar Multiplication Using Point Halving.* Proocedings of ASIACRYPT 1999, LNCS **1716**, 135–149. Springer, 1999.
23. N. Koblitz. *CM-curves with good cryptographic properties.* In: *Proceedings of CRYPTO 1991*, LNCS **576**, 279–287. Springer, 1991.
24. T. Lange and M. Stevens. *Efficient doubling for genus two curves over binary fields.* In: *Selected Areas in Cryptography – SAC 2004.* LNCS **3357**, 170–181, Springer-Verlag, 2005.
25. National Institute of Standards and Technology. *Recommended Elliptic Curves for Federal Government Use.* NIST Special Publication, July 1999. Available from: `http://csrc.nist.gov/csrc/fedstandards.html`
26. R. Schroeppel. *Point halving wins big.* Talks at: (i) Midwest Arithmetical Geometry in Cryptography Workshop, November 17–19, 2000, University of Illinois at Urbana-Champaign; and (ii) ECC 2001 Workshop, October 29–31, 2001, University of Waterloo, Ontario, Canada.
27. R. Schroeppel. *Elliptic curve point ambiguity resolution apparatus and method.* International Application Number PCT/US00/31014, filed 9 November 2000.
28. R.G. Swan. *Factorization of Polynomials over Finite Fields.* In Pac. J. Math. **19**, 1099–1106, 1962.

A Some values of ζ

In this appendix we provide two comparisons of the values of ζ when type I SSRF polynomials are used and when they are not.

A.1 Degree 83

For the field $\mathbb{F}_{2^{83}}$ we choose $p(X) = X^{83} + X^{29} + X^{25} + X^3 + 1$. The corresponding expression for ζ is $x^{42} + x^{15} + x^{13} + x^2$.

Computer algebra systems, like Magma [16] , usually suggest to use the polynomial $p(X) = X^{83} + X^7 + X^4 + X^2 + 1$ instead. But, in this case $\zeta = x^{82}+x^{80}+x^{79}+x^{77}+x^{76}+x^{74}+x^{73}+x^{71}+x^{70}+x^{68}+x^{67}+x^{65}+x^{64}+x^{62}+x^{61}+x^{59}+x^{58}+x^{56}+x^{55}+x^{53}+x^{52}+x^{50}+x^{49}+x^{47}+x^{46}+x^{44}+x^{43}+x^{41}+x^{39}+x^{38}+x^{36}+x^{35}+x^{33}+x^{32}+x^{30}+x^{29}+x^{27}+x^{26}+x^{24}+x^{23}+x^{21}+x^{20}+x^{18}+x^{17}+x^{15}+x^{14}+x^{12}+x^{11}+x^9+x^8+x^5+x^4+x^3+1.$

The internal representation of ζ, given in hexadecimal notation where the most significant bits are to the left, is `5b6db6db 6dadb6db 6db39`, that has weight 54. It shows some intriguing patterns.

A.2 Degree 163

To define $\mathbb{F}_{2^{163}}$ we choose $X^{163} + X^{57} + X^{49} + X^{29} + 1$, and we have $\zeta = x^{82} + x^{29} + x^{25} + x^{15}$ of weight four.

On the other hand, if the standard polynomial $p(X) = X^{163} + X^7 + X^6 + X^3 + 1$ is used, then $\zeta = x^{162} + x^{159} + x^{156} + x^{153} + x^{150} + x^{147} + x^{144} + x^{141} + x^{138} + x^{135} + x^{132} + x^{129} + x^{126} + x^{123} + x^{120} + x^{117} + x^{114} + x^{111} + x^{108} + x^{105} + x^{102} + x^{99} + x^{96} + x^{93} + x^{90} + x^{87} + x^{84} + x^{81} + x^{79} + x^{78} + x^{76} + x^{75} + x^{73} + x^{72} + x^{70} + x^{69} + x^{67} + x^{66} + x^{64} + x^{63} + x^{61} + x^{60} + x^{58} + x^{57} + x^{55} + x^{54} + x^{52} + x^{51} + x^{49} + x^{48} + x^{46} + x^{45} + x^{43} + x^{42} + x^{40} + x^{39} + x^{37} + x^{36} + x^{34} + x^{33} + x^{31} + x^{30} + x^{28} + x^{27} + x^{25} + x^{24} + x^{22} + x^{21} + x^{19} + x^{18} + x^{16} + x^{15} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^5 + x^4$, of weight 79. The internal representation of ζ is `4 92492492 49249249 2492db6d b6db6db6 db6db6b0`.

B Comparing some modular reduction routines

The use of a square root friendly polynomial can slow down modular reduction, but we already observed that this performance loss is minimal. This is explained by the fact that even though reduction does become more expensive, the amount of additional operations is rather small.

As an example, we report here the reduction code for the two degree 163 polynomials which we used. The input is given as eleven 32-bit words `rA,r9,r8,...,r1,r0` and the reduced output is computed in place in the six least significant words `r5,r4,r3,r2,r1,r0`.

To reduce modulo $X^{163} + X^7 + X^6 + X^3 + 1$, the number of necessary logical operations between CPU registers is 74. Reduction modulo $X^{163} + X^{57} + X^{49} + X^{29} + 1$ takes 89 logical operations.

```
#define bf_mod_163_7_6_3_0(rA,r9,r8,r7,r6,r5,r4,r3,r2,r1,r0) do { \
    /* reduce rA */                                              \
    r5 ^= (rA) ^ ((rA) <<  3) ^ ((rA) <<  4) ^ ((rA) >>  3);     \
    r4 ^=                                      ((rA) << 29);     \
    /* reduce r9 */                                              \
    r5 ^=       ((r9) >> 29) ^ ((r9) >> 28);                     \
    r4 ^= (r9) ^ ((r9) <<  3) ^ ((r9) <<  4) ^ ((r9) >>  3);     \
    r3 ^=                                      ((r9) << 29);     \
    /* reduce r8 */                                              \
    r4 ^=       ((r8) >> 29) ^ ((r8) >> 28);                     \
    r3 ^= (r8) ^ ((r8) <<  3) ^ ((r8) <<  4) ^ ((r8) >>  3);     \
```

```
    r2 ^=                                   ((r8) << 29);      \
    /* reduce r7 */                                            \
    r3 ^=        ((r7) >> 29) ^ ((r7) >> 28);                  \
    r2 ^= (r7) ^ ((r7) <<  3) ^ ((r7) <<  4) ^ ((r7) >>  3);   \
    r1 ^=                                   ((r7) << 29);      \
    /* reduce r6 */                                            \
    r2 ^=        ((r6) >> 29) ^ ((r6) >> 28);                  \
    r1 ^= (r6) ^ ((r6) <<  3) ^ ((r6) <<  4) ^ ((r6) >>  3);   \
    r0 ^=                                   ((r6) << 29);      \
    /* reduce the 29 most significant bits of r5 */            \
    r6 = (r5) >> 3; r5 &= 0x00000007;                          \
    r0 ^= (r6) ^ ((r6) << 3) ^ ((r6) << 6) ^ ((r6) << 7);      \
    r1 ^= ((r6) >> 26) ^ ((r6) >> 25);                         \
} while (0)

#define bf_mod_163_57_49_29_0(rA,r9,r8,r7,r6,r5,r4,r3,r2,r1,r0) do { \
    /* reduce rA */                                                  \
    r6 ^= ((rA) << 22) ^ ((rA) << 14);                               \
    r5 ^=                              ((rA) << 26) ^ ((rA) >>  3); \
    r4 ^=                                           ((rA) << 29); \
    /* reduce r9 */                                                  \
    r6 ^= ((r9) >> 10) ^ ((r9) >> 18);                               \
    r5 ^= ((r9) << 22) ^ ((r9) << 14) ^ ((r9) >>  6);                \
    r4 ^=                              ((r9) << 26) ^ ((r9) >>  3); \
    r3 ^=                                           ((r9) << 29); \
    /* reduce r8 */                                                  \
    r5 ^= ((r8) >> 10) ^ ((r8) >> 18);                               \
    r4 ^= ((r8) << 22) ^ ((r8) << 14) ^ ((r8) >>  6);                \
    r3 ^=                              ((r8) << 26) ^ ((r8) >>  3); \
    r2 ^=                                           ((r8) << 29); \
    /* reduce r7 */                                                  \
    r4 ^= ((r7) >> 10) ^ ((r7) >> 18);                               \
    r3 ^= ((r7) << 22) ^ ((r7) << 14) ^ ((r7) >>  6);                \
    r2 ^=                              ((r7) << 26) ^ ((r7) >>  3); \
    r1 ^=                                           ((r7) << 29); \
    /* reduce r6 */                                                  \
    r3 ^= ((r6) >> 10) ^ ((r6) >> 18);                               \
    r2 ^= ((r6) << 22) ^ ((r6) << 14) ^ ((r6) >>  6);                \
    r1 ^=                              ((r6) << 26) ^ ((r6) >>  3); \
    r0 ^=                                           ((r6) << 29); \
    /* reduce the 29 most significant bits of r5 */                  \
    r6 = (r5) >> 3; r5 &= 0x00000007;                                \
    r2 ^= ((r6) >>  7) ^ ((r6) >> 15);                               \
    r1 ^= ((r6) << 25) ^ ((r6) << 17) ^ ((r6) >>  3);                \
    r0 ^=                              ((r6) << 29) ^  (r6);        \
} while(0)
```