Optimal Irreducible Polynomials for GF(2^m) Arithmetic

Michael Scott

Dublin City University
Ballymun
Dublin
Ireland.
mike@computing.dcu.ie

Abstract. The irreducible polynomials recommended for use by multiple standards documents are in fact far from optimal on many platforms. Specifically they are suboptimal in terms of performance, for the computation of field square roots and in the application of the "almost inverse" field inversion algorithm. In this paper we question the need for the standardisation of irreducible polynomials in the first place, and derive the "best" polynomials to use depending on the underlying processor architecture. Surprisingly it turns out that a trinomial polynomial is in many cases not necessarily the best choice. Finally we make some specific recommendations for some particular types of architecture.

Keywords: Irreducible polynomials. Arithmetic in $\mathbb{F}_{2^m}$.

1 Introduction

The main application of $\mathbb{F}_{2^m}$ arithmetic (where m is an odd prime) is in elliptic curve cryptography [11], although recently it has also found constructive application in pairing-based cryptography, specifically in the implementation of the η_T pairing, which is one of the fastest known [5]. Clearly we would like the underlying $\mathbb{F}_{2^m}$ arithmetic to be implemented as efficiently as possible.

Elements in $\mathbb{F}_{2^m}$ can be represented as a multi-precision string of bits, each bit representing a coefficient in a polynomial of degree at most $(m-1)$, a so-called *polynomial basis representation.* The field is specified in conjunction with an *irreducible polynomial*, and multiplication of field elements is performed modulo this irreducible polynomial.

Addition of field polynomials is a simple coefficient-by-coefficient XOR operation. By grouping coefficients into blocks, each block the size of one computer word of w bits, the addition can be carried out using $\lceil m/w \rceil$ word XOR operations, an operation supported by all computer architectures. This can be thought of as the same as multiprecision integer addition, without the carries.

Multiplication can be performed using the standard school-boy long multiplication algorithm, where in the calculation of each partial product carries are simply ignored. The carry-free nature of this type of arithmetic can be an advantage (implementing multiplication using Karatsuba's algorithm for example is much simpler and faster), and a disadvantage (most architectures do not support carry-free word multiplications).

Squaring is much faster than multiplication – it can be achieved by simply inserting a zero between each coefficient. Multiplication or squaring both result in a polynomial of degree at most $2m-2$, and it is this polynomial which must be reduced modulo the irreducible polynomial. And it is this reduction process which is the focus of this paper.

For security reasons the parameter m is chosen as a prime number. Certain fields are recommended for use in elliptic curve cryptography in popular standards. For example Certicom in their "Standards for Efficient Cryptography" (available from www.secg.org) recommend $m \in \{113, 131, 163, 193, 233, 239, 283, 409, 571\}$, and it is these fields that we will focus on here. They also insist on specific irreducible polynomials for each case ("... must use [these] reduction polynomials ... to encourage interoperability"), although interestingly they offer a choice of two polynomials for the case of $m = 239$, on the grounds that both have been commonly used in practise.

Unfortunately the recommended irreducible polynomials have several disadvantages

- Few of them are optimal in terms of the actual implementation of the reduction algorithm on popular architectures.
- Few are suitable for the calculation of field square roots, which is required for elliptic curve point halving operations [4], and for the η_T pairing [5].
- Many are unsuitable for use with the almost inverse algorithm for field inversion [11].

The reason for the standardisation of irreducible polynomials is that the represention of field elements depends upon it. Therefore it is important for example in a communications protocol in which field elements are transmitted from one party to another, that all participants are aware of the irreducible polynomial in use by the others. However it is a simple matter to include the parameters that define the irreducible polynomial as part of the domain information which each participant needs to aware of anyway. In this case the irreducible polynomial might be chosen to facilitate the participant with the lowest computational power, and hence in most need of an optimal choice of polynomial. Alternatively, one can change from one polynomial representation to another should that be necessary as described in [13], section A.7. The change of basis algorithm is a little awkward, and requires a precomputed $m \times m$ bit conversion matrix. It could represent a significant overhead for a poorly-resourced processor, and so should only be expected of the more powerful protocol participants.

The fact that standard elliptic curve domain parameters are described in terms of the standard polynomials presents a barrier to converting them to a more suitable polynomial, and most implementors just stick with the standard polynomials for implementation as well as for representation.

2 Irreducible polynomials

For the field $\mathbb{F}_{2^m}$ it is in practise always possible to chooose as an irreducible polynomial either a trinomial

$$x^m + x^a + 1$$

or a pentanomial

$$x^m + x^a + x^b + x^c + 1$$

In both cases the optimal reduction algorithm is linear and fast compared to the worse-than-linear multiplication algorithm. For efficiency the reduction algorithm of choice [11] requires that $m - a \geq w$.

In [8] it is stated that "performance reasons impose that irreducible polynomials have the shortest number of non-zero terms", and it appears to be a commonly held view that trinomials are better than pentanomials. As we shall see this is not necessarily the case.

The standard polynomials are selected according to the following criteria (with the exception of one of the $m = 239$ polynomials). If a trinomial exists, the trinomial with the smallest value for a is chosen. Otherwise the pentanomial is chosen with the smallest a, then the smallest b given a, and then the smallest c given a and b. Given these rules it is relatively easy to find the standard polynomial for any value of m. They are also most likely to fulfil the condition that $m-a \geq w$. There is also a widespread belief that such polynomials are also in some sense optimal. For example Ahmadi and Menezes [3] suggest a construction where the middle terms are "all of relatively low degree and are close to each other, which in turn facilitates efficient multiplication of polynomials modulo $f(x)$".

However there does not seem to be any basis for this suggestion for software implementations, although it may have merit in hardware. For example, as we shall see, the standard polynomial for $\mathbb{F}_{2^{283}}$ is far from being optimal, despite adhering to these recommendations.

Recently there have emerged applications in which it is important that square rooting should be fast, specifially point halving algorithms for fast elliptic curve point multiplication [4], [11], and the η_T pairing [5]. Fast square rooting requires that a (and b and c for a pentanomial) must all be odd [10]. Such polynomials are easy to find, but unfortunately most of the standard polynomials are not of this form [4].

It has been known for some time that the Schroeppel et al. "almost inverse" algorithm for field inversions is more efficient if $a \geq w$ for a trinomial, and $c \geq w$ for a pentanomial [14] (although in [10] a couple of strategies due to Knudsen and Schroeppel are described which can to an extent circumvent this restriction). Many of the standard irreducible polynomials do not satisfy this condition, although in practise it may be that this algorithm is not the algorithm of choice [11]. Nonetheless the authors of [14] complain that "Unfortunately, most of the field polynomials specified in ANSI X9.62 do have terms of low degree. This increases the timings of the almost inverse algorithm by up to 30%. Therefore, we conclude that the choice of polynomials in ANSI X9.62 is rather unfortunate, and may be revised if that is practically feasible.”.

A third, and hitherto largely neglected, issue is that of performance. Although the reduction algorithm is always fast compared with multiplication, it applies also to squarings. One of the advantages of using methods based on $\mathbb{F}_{2^m}$ fields is that squarings are potentially so fast. But if the reduction algorithm is slow this advantage will be offset.

3 An Example

For the field $\mathbb{F}_{2^{163}}$ the standard irreducible polynomial is $x^{163}+x^7+x^6+x^3+1$. The reduction algorithm as described by Hankerson et al. in section 2.3.5 of [11] for a 32-bit processor is illustrated in Algorithm 1.

Algorithm 1 Fast reduction modulo $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$

```
INPUT: A binary polynomial represented as 11 32-bit words $g[.]$
OUTPUT: $g[.]$ reduced modulo $f(x)$, represented as 6 32-bit words
for $i \leftarrow 10$ downto 6 do
    $t \leftarrow g[i]$
    $g[i] \leftarrow 0$
    $g[i-6] \leftarrow g[i-6] \oplus (t \ll 29)$
    $g[i-5] \leftarrow g[i-5] \oplus (t \ll 4) \oplus (t \ll 3) \oplus t \oplus (t \gg 3)$
    $g[i-4] \leftarrow g[i-4] \oplus (t \gg 28) \oplus (t \gg 29)$
end for
$t \leftarrow g[5] \gg 3$
$g[0] \leftarrow g[0] \oplus t$
$t \leftarrow (t \ll 3)$
$g[1] \leftarrow g[1] \oplus (t \gg 28) \oplus (t \gg 29)$
$g[0] \leftarrow g[0] \oplus t \oplus (t \ll 4) \oplus (t \ll 3)$
$g[5] \leftarrow g[5] \oplus t$
```

After the loop there is some tidying-up to be done to deal with the most significant word of the result. Concentrating on the time-critical loop, unrolling it (and omitting for clarity the rest of the algorithm) we get Algorithm 2.

Note that this requires 30 word shift operations 35 word XORs, which can be considered as 5 times the 6 shifts and 7 XORs required by each iteration of the loop.

Now consider a different irreducible polynomial, $f(x) = x^{163} + x^{99} + x^{97} + x^3 + 1$. The equivalent algorithm is shown in Algorithm 3.

In both cases the number of memory reads and writes are the same. However it is clear from inspection alone that this last algorithm will be faster. In fact it requires only 20 word shifts and 30 XORs, or 5 times the 4 shifts and 6 XORs required by each iteration of the loop. To understand the reasons for

Algorithm 2 Loop-unrolled reduction modulo $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$

```
INPUT: A binary polynomial represented as 11 32-bit words $g[.]$
OUTPUT: $g[.]$ partially reduced modulo $f(x)$, represented as 6 32-bit words
$g_{10} \leftarrow g[10],\ g_9 \leftarrow g[9],\ g_8 \leftarrow g[8],\ g_7 \leftarrow g[7],\ g_6 \leftarrow g[6]$
$g[10] \leftarrow g[9] \leftarrow g[8] \leftarrow g[7] \leftarrow g[6] \leftarrow 0$
$g_6 \leftarrow g_6 \oplus (g_{10} \gg 28) \oplus (g_{10} \gg 29)$
$g[5] \leftarrow g[5] \oplus (g_{10} \gg 3) \oplus (g_{10} \ll 4) \oplus (g_{10} \ll 3) \oplus g_{10} \oplus (g_9 \gg 28) \oplus (g_9 \gg 29)$
$g[4] \leftarrow g[4] \oplus (g_{10} \ll 29) \oplus (g_9 \gg 3) \oplus (g_9 \ll 4) \oplus (g_9 \ll 3) \oplus g_9 \oplus (g_8 \gg 28) \oplus (g_8 \gg 29)$
$g[3] \leftarrow g[3] \oplus (g_9 \ll 29) \oplus (g_8 \gg 3) \oplus (g_8 \ll 4) \oplus (g_8 \ll 3) \oplus g_8 \oplus (g_7 \gg 28) \oplus (g_7 \gg 29)$
$g[2] \leftarrow g[2] \oplus (g_8 \ll 29) \oplus (g_7 \gg 3) \oplus (g_7 \ll 4) \oplus (g_7 \ll 3) \oplus g_7 \oplus (g_6 \gg 28) \oplus (g_6 \gg 29)$
$g[1] \leftarrow g[1] \oplus (g_7 \ll 29) \oplus (g_6 \gg 3) \oplus (g_6 \ll 4) \oplus (g_6 \ll 3) \oplus g_6$
$g[0] \leftarrow g[0] \oplus (g_6 \ll 29)$
```

Algorithm 3 Loop-unrolled reduction modulo $f(x) = x^{163} + x^{99} + x^{97} + x^3 + 1$

```
INPUT: A binary polynomial represented as 11 32-bit words $g[.]$
OUTPUT: $g[.]$ partially reduced modulo $f(x)$, represented as 6 32-bit words
$g_{10} \leftarrow g[10],\ g_9 \leftarrow g[9],\ g_8 \leftarrow g[8],\ g_7 \leftarrow g[7],\ g_6 \leftarrow g[6]$
$g[10] \leftarrow g[9] \leftarrow g[8] \leftarrow g[7] \leftarrow g[6] \leftarrow 0$
$g_8 \leftarrow g_8 \oplus g_{10} \oplus (g_{10} \gg 2)$
$g_7 \leftarrow g_7 \oplus (g_{10} \ll 30) \oplus g_9 \oplus (g_9 \gg 2)$
$g_6 \leftarrow g_6 \oplus (g_9 \ll 30) \oplus g_8 \oplus (g_8 \gg 2)$
$g[5] \leftarrow g[5] \oplus (g_{10} \gg 3) \oplus g_{10} \oplus (g_8 \ll 30) \oplus g_7 \oplus (g_7 \gg 2)$
$g[4] \leftarrow g[4] \oplus (g_{10} \ll 29) \oplus (g_9 \gg 3) \oplus g_9 \oplus (g_7 \ll 30) \oplus g_6 \oplus (g_6 \gg 2)$
$g[3] \leftarrow g[3] \oplus (g_9 \ll 29) \oplus (g_8 \gg 3) \oplus g_8 \oplus (g_6 \ll 30)$
$g[2] \leftarrow g[2] \oplus (g_8 \ll 29) \oplus (g_7 \gg 3) \oplus g_7$
$g[1] \leftarrow g[1] \oplus (g_7 \ll 29) \oplus (g_6 \gg 3) \oplus g_6$
$g[0] \leftarrow g[0] \oplus (g_6 \ll 29)$
```

the speed up, consider the constants that arise in the shift operations. These are the numbers $m \bmod w$, $w-(m \bmod w)$, $m-a \bmod w$, $w-(m-a \bmod w)$, $m-b \bmod w$, $w-(m-b \bmod w)$, $m-c \bmod w$ and $w-(m-c \bmod w)$. So for the standard polynomial these values in pairs are (3, 29), (28, 4), (29, 3), and (0, 32) respectively. For the fortuitous values of 0 and 32 that arise in this case, the former implies no shifting, and the latter, a shift by the word length, results in a zero. This piece of luck results in a saving of 1 XOR and 2 shifts per loop iteration.

Our selection of irreducible polynomial is based on the idea of "making our own luck" in the choice of irreducible polynomial. In our case the shift values are (3, 29), (0, 32), (30, 2), and (0, 32) and this explains the savings achieved. Given that m is odd our chances of being lucky like this can only improve if we insist that a, b and c are also all chosen to be odd.

We note that this same idea is also alluded to in recent papers by Ahmadi et al [1], section 4.2, and by Hankerson and Rodríguez-Henríquez [12], who also independently suggested the use of the polynomial $x^{163}+x^{99}+x^{97}+x^3+1$.

4 Further analysis

We define a trinomial where $m-a \equiv 0 \bmod w$ as a *lucky* trinomial (LT), and a trinomial for which this condition does not hold as an *ordinary* trinomial (OT). We define a pentanomial where $m-a \equiv m-b \equiv m-c \equiv 0 \bmod w$ as a *lucky* pentanomial (LP). A pentanomial for which two out of three of these values is congruent to zero is called a *fortunate* pentanomial (FP), the rest are *ordinary* pentanomials (OP). In general, as we will see, a lucky trinomial beats a lucky pentanomial, which beats an ordinary trinomial, which in turn beats a fortunate pentanomial. Whereas a lucky trinomial is relatively rare, a lucky pentanomial can often be found, and hence in many cases the pentanomial is superior to a trinomial.

Is an irreducible lucky pentanomial always possible? The answer is no – a lucky irreducible pentanomial is not possible for any wordlength w which is a multiple of 4 if $m \equiv \pm 3 \bmod 8$, as a direct consequence of a recent theorem of Buhler [6], [2]. Of course all popular computer word-lengths are a multiple of 4 (typically 8, 16, 32 or 64).

However under the same circumstances useful irreducible trinomials do not exist either [7], and so in these cases we must be satisfied with a fortunate pentanomial (if we can find one).

When lucky pentanomials do exist for $m \equiv \pm 1 \bmod 8$, we get the added bonus on a small eight bit processor that the shifts by $m \bmod w$ and $w-m \bmod w$, are in fact shifts by 1-bit and 7-bits (or visa versa), which are likely to be efficient.

Note that these definitions are word-length dependant. So whereas a lucky trinomial might exist for one wordlength, it may not be so lucky for a larger wordlength.

Finally we should point out that there is another way to be lucky. It is possible that for example $m - a \equiv m - b \bmod w$. These means that the same shifted values can be re-used, with some savings.

5 Some real-world architectures

Of course the significance of all these potential savings depends on the particular computer architecture. In this paper we consider four representative real-world examples.

- A 32-bit Pentium or MIPS type of processor, which has fast 1-cycle XOR and shift instructions. We ignore the fact that these architectures often support multiple pipelines – we will assume that execution time is simply preportional to the total number of such instructions (but given the complexity of this architecture and its many variations, we recognise that this may be a rather reckless assumption).
- A 32-bit ARM processor. This architecture supports a "barrel shifter", and an XOR instruction for example can at no extra cost shift one of its operands by any number of bits. Therefore shifts are effectively free on this architecture.
- A Texas Instruments ultra-low power msp430 16-bit processor, as used in wireless sensor networks. This architecture supports a 1 cycle XOR instruction, but has only a 1-bit shift instruction. Therefore shifts by multiple bits require multiple instructions. Fortunately the instruction set does allow a 1-cycle byte swap within a register, and simple masking instructions, which means that for example an 8-bit left shift of a 16-bit register can be accomplished in 2 instructions.
- An Atmel Atmega128 low power 8-bit processor, another favourite for applications in wireless sensor networks. Again a 1-cycle 1-bit shift operation is supported, along with a 1-cycle nibble swap within a register, as well as register masking.

To find the optimal irreducible polynomial in any particular circumstance, we cost each XOR and shift operation appropriately. An XOR always has a cost of 1. A shift may have a cost of zero (for the ARM), or a larger cost. In the case of the msp430 processor the costs are given in Table 1.

On this architecture right shifts can be slightly more expensive than left shifts, as all rotates and right shifts are through the carry flag, and this must be cleared to obtain a logical right shift as required here. However this only applies to the first of a sequence of right shifts. Note that if the algorithm needs a left-shift by 3, and also a left shift of the same value by 4 within a single iteration, then we assume a compiler will be smart enough to shift once by 3 and follow that by a further shift by 1 bit. A shift left by 15 bits is best achieved by a rotate right followed by a masking. Similar methods can be deployed for the Atmel 8-bit processor, and the costs of shifts for this processor are given in Table 2.

Table 1. Shift cost in clock cycles for msp430 16-bit processor

Shift size	Left shift	Right shift
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	5	5
8	2	2
9	3	3
10	4	4
11	5	5
12	6	6
13	5	5
14	4	4
15	3	3

Table 2. Shift cost in clock cycles for Atmega128 8-bit processor

Shift size	Left shift	Right shift
1	1	1
2	2	2
3	3	3
4	2	2
5	3	3
6	4	4
7	3	3

6 Results

For each candidate irreducible trinomial and pentanomial we calculate the associated costs and we select the cheapest, depending on the cost function that applies for that architecture. Recall that an irreducible polynomial which is good for a 16-bit processor may not be so good for a 32-bit processor, for example $(m-a)$ may be a multiple of 16, but not of 32. In some cases the outcome is not clear-cut, as it would depend for example on whether or not there would be an intention to use the "almost inverse" algorithm. In the case of more than one solution with the same cost, we favour the solution that is friendly for the "almost inverse" algorithm. We do however insist that all exponents in the irreducible polynomial are odd, given the recently realised significance of field square roots. In any case, in no instances were optimal polynomials found with any even exponents.

Some notable outcomes

- In some cases a pentanomial may be cheaper than a trinomial. In fact for the msp430 architecture a trinomial is never optimal for the fields considered here. However some algorithms (other than the reduction algorithm) may be more efficient with a trinomial, so there may still be reasons for preferring a trinomial if one should exist. For the Atmel 8-bit processor, trinomials make something of a come-back, as for the first time we find some lucky trinomials.
- The "folklore" requirement that for a pentanomial irreducible polynomial the middle terms (a, b and c) being close to one another and small leads to a more efficient algorithm, does not appear to have any validity.

Consider for example the field $\mathbb{F}_{2^{233}}$. This supports the trinomial $x^{233}+x^{159}+1$. However for the Pentium cost model the pentanomial $x^{233}+x^{201}+x^{105}+x^{9}+1$ is superior. Note that (233-201), (233-105) and (233-9) are all multiples of 32. Whereas the trinomial requires 4 XORS and 4 shifts per loop iteration, the pentanomial costs 5 XORS and only 2 shifts. However for the ARM model the trinomial is still superior as shifts are free. For the msp430 an optimal polynomial is $x^{233}+x^{185}+x^{121}+x^{105}+1$. In this case we do find a solution which also accomodates the "almost inverse" algorithm. Note that in no cases are the middle terms particularly close to one another.

Avanzi [4] suggests choosing the square-root friendly irreducible polynomials with odd exponents and the least degree sediment, that is with the smallest size of a. However there seems to be no compelling reason for doing so. For the $m = 163$ case the cost as we calculate it is even greater with this choice than that for the standard polynomial. See Appendix A for a worked example of the square root algorithm.

7 Redundant Trinomials

In a recent paper Doche [9], building on earlier work by Brent and Zimmerman [7], has suggested the use of redundant trinomials in place of pentanomials. However our surprising observation that pentanomials can actually be faster than

Table 3. Optimal irreducible polynomials for Pentium-type 32-bit processor

m	Optimal polynomial	Cost of this polynomial	standard polynomial	Type
113	$x^{113}+x^{15}+1$	8	8	OT
131	$x^{131}+x^{97}+x^{65}+x^{3}+1$	11	13	OP
163	$x^{163}+x^{99}+x^{97}+x^{3}+1$	10	13	FP
193	$x^{193}+x^{73}+1$	8	8	OT
233	$x^{233}+x^{201}+x^{105}+x^{9}+1$	7	8	LP
239	$x^{239}+x^{207}+x^{111}+x^{47}+1$	7	8	LP
283	$x^{283}+x^{249}+x^{219}+x^{27}+1$	10	16	FP
409	$x^{409}+x^{377}+x^{185}+x^{57}+1$	7	8	LP
571	$x^{571}+x^{507}+x^{475}+x^{417}+1$	10	16	FP

Table 4. Optimal irreducible polynomials for ARM-type 32-bit processor

m	Optimal polynomial	Cost of this polynomial	standard polynomial	Type
113	$x^{113}+x^{15}+1$	4	4	OT
131	$x^{131}+x^{99}+x^{97}+x^{95}+1$	7	7	OP
163	$x^{163}+x^{99}+x^{97}+x^{3}+1$	6	7	FP
193	$x^{193}+x^{73}+1$	4	4	OT
233	$x^{233}+x^{159}+1$	4	4	OT
239	$x^{239}+x^{203}+1$	4	4	OT
283	$x^{283}+x^{249}+x^{219}+x^{27}+1$	6	8	FP
409	$x^{409}+x^{87}+1$	4	4	OT
571	$x^{571}+x^{507}+x^{475}+x^{417}+1$	6	8	FP

Table 5. Optimal irreducible polynomials for msp430 16-bit processor

m	Optimal polynomial	Cost of this polynomial	standard polynomial	Type
113	$x^{113}+x^{97}+x^{65}+x^{33}+1$	10	13	LP
131	$x^{131}+x^{115}+x^{81}+x^{67}+1$	16	22	FP
163	$x^{163}+x^{131}+x^{129}+x^{115}+1$	16	22	FP
193	$x^{193}+x^{145}+x^{129}+x^{113}+1$	10	12	LP
233	$x^{233}+x^{185}+x^{121}+x^{105}+1$	13	16	LP
239	$x^{239}+x^{207}+x^{111}+x^{47}+1$	9	13	LP
283	$x^{283}+x^{225}+x^{203}+x^{107}+1$	17	33	FP
409	$x^{409}+x^{377}+x^{185}+x^{57}+1$	13	19	LP
571	$x^{571}+x^{507}+x^{475}+x^{417}+1$	17	31	FP

Table 6. Optimal irreducible polynomials for Atmega128 8-bit processor

m	Optimal polynomial	Cost of this polynomial	standard polynomial	Type
113	$x^{113}+x^{9}+1$	7	7	LT
131	$x^{131}+x^{115}+x^{81}+x^{67}+1$	13	15	FP
163	$x^{163}+x^{131}+x^{129}+x^{115}+1$	13	17	FP
193	$x^{193}+x^{73}+1$	7	11	LT
233	$x^{233}+x^{185}+x^{121}+x^{105}+1$	9	12	LP
239	$x^{239}+x^{207}+x^{111}+x^{47}+1$	9	12	LP
283	$x^{283}+x^{249}+x^{219}+x^{27}+1$	13	20	FP
409	$x^{409}+x^{377}+x^{185}+x^{57}+1$	9	11	LP
571	$x^{571}+x^{507}+x^{475}+x^{417}+1$	13	21	FP

trinomials rather undermines the basis of their results. Certainly for applications in cryptography the premise [7] that pentanomials are "considerably more expensive in applications" is not supported. Nevertheless there may be cases where a redundant trinomial (or indeed pentanomial) may be superior to the irreducible polynomials suggested here. For example, for the trickly $m = 163$ case the lucky redundant pentanomial $x^{165} + x^{69} + x^{37} + x^{5} + 1$ is superior in all cases.

In practice any small improvement possible with redundant polynomials may be more than offset by the extra complications involved in field element comparisons, inversions and by the requirement for possibly one or more extra computer words to be added to the field representation [9]. Note our extra condition that m and a be odd further constrains the choice of redundant trinomials, so many of the solutions presented in [9] are unsuitable.

8 Conclusions

In the light of recent developments the irreducible polynomials as recommended in some standards are in urgent need of an overhaul. In fact we would argue that the choice of irreducible polynomial should be left to the implementor, and that issues that arise from the use of different irreducible polynomials by communicating parties should be dealt with in some other way other than through standardisation.

We have derived polynomials that, when used for modular reduction, are in all cases at least as fast (and often much faster) than those suggested in the standards. We have also described a simple methodology for determining the best polynomial to use in any given circumstance, using a simple and easy to develop costing model. All of our suggested polynomials support a very fast field square root operation, and there seems to be no good reason not to use a square-root friendly polynomial in all cases.

Alternatively, based on the method described here, it might be possible to come up with good compromise irreducible polynomials, which while not necessarily being optimal in all cases, would nevertheless be an improvement on the current standards. As can be seen from the tables above, all agree on the optimal irreducible polynomial for the case $m = 571$, and there are other examples of majority agreement for other values of m.

9 Acknowledgments

Thanks are due to Darrel Hankerson and an anonymous referee of an early draft who pointed out to me the work of Brent and Zimmerman [7] and Doche [9].

References

1. O. Ahmadi, D. Hankerson, and A. Menezes. Software implementation of arithmetic in GF(3^m). CACR Technical Reports, 2007. `http://www.cacr.math.uwaterloo.ca/techreports/2005/cacr2007-15.pdf`.
2. O. Ahmadi and A. Menezes. Irreducible polynomials of maximum weight. CACR Technical Reports, 2005. `http://www.cacr.math.uwaterloo.ca/techreports/2005/cacr2005-01.pdf`.
3. O. Ahmadi and A. Menezes. On the number of trace-one elements in polynomial bases for GF(2^m). *Designs, Codes and Cryptography*, 37:493–507, 2005.
4. R. Avanzi. A note on square roots in binary fields. Cryptology ePrint Archive, Report 2007/103, 2007. `http://eprint.iacr.org/2007/103`.
5. P.S.L.M. Barreto, S. Galbraith, C. OhEigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007. `http://eprint.iacr.org/2004/375`.
6. A. W. Bluher. A Swan-like theorem. *Finite Fields Appl.*, 12:128–138, 2006.
7. R. Brent and P. Zimmerman. Algorithms for finding almost irreducible and almost primitive trinomials. Proceedings of a conference in honour of Professor H.C. Williams, 2003. `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pd/rpb212.pdf`.
8. M. Ciet, J. J. Quisquater, and F. Sica. A short note on irreducible trinomials in binary fields. Proceedings of the 23rd Symposium on Information Theory in the Benelux, 2002. `citeseer.ist.psu.edu/559928.html`.
9. C. Doche. Redundant trinomials for finite fields of characteristic 2. In *ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 122–133. Springer-Verlag, 2005.
10. K. Fong, D. Hankerson, J. López, and A. Menezes. Field inversion and point halving revisited. Technical report CORR 2003-18, University of Waterloo, 2002.
11. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curves Cryptography.* Springer, 2004.
12. D. Hankerson and F. Rodríguez-Henríquez. Parallel formulation of scalar multiplication on koblitz curves. CACR Technical Reports, 2007. `http://www.cacr.math.uwaterloo.ca/techreports/2005/cacr2007-18.pdf`.
13. IEEE Computer Society, New York, USA. *IEEE Standard Specifications for Public-Key Cryptography – IEEE Std 1363:2000*, 2000. `http://grouper.ieee.org/groups/1363`.
14. P. De Win, S. Mister, B. Preneel, and M. Wiener. On the performance of signature schemes based on elliptic curves. In *ANTS, 3rd International Symposium*, volume 1423 of *Lecture Notes in Computer Science*, pages 252–266. Springer-Verlag, 1998.

A Calculating square roots

As pointed out by Fong et al. [10] if using an irreducible polynomial with all odd exponents, for example the trinomial $x^m + x^a + 1$, and setting $\zeta = x^{(m+1)/2} + x^{(a+1)/2}$, then a field square root can be expressed as

$$\sqrt{\alpha} = \alpha_{even} + \zeta.\alpha_{odd}$$

where α_{even} are the even indexed bits of α collapsed into a half-sized bit array, and α_{odd} are the odd indexed bits again collapsed into a half-sized bit array. The multiplication by the term $x^{(m+1)/2}$ of ζ is performed by simply abutting α_{odd} onto α_{even} as the higher order bits of the final result. The multiplication of α_{odd} by the other term(s) of ζ must then be finally dealt with, and the amount of work required for this step depends to an extent on the particular irreducible polynomial. Note that no reduction of the resulting polynomial is required, as it will be of degree $m - 1$.

To extract the odd and even indices of α, small look-up tables can be used. For example the byte 01111011 needs to be split into evens 1101 and odds 0111, and so 256 element tables can be pre-computed, with `evens[123]=13` and `odds[123]=7` etc.

In this context being "lucky" as regards an irreducible trinomial, requires that $(m+1)/2$ and $(a+1)/2$ are multiples of the word length. Now consider the case $m = 271$, and the lucky pentanomial $x^{271} + x^{207} + x^{175} + x^{111} + 1$. Here as it happens we are doubly lucky as $(m+1)/2$, $(a+1)/2$, $(b+1)/2$ and $(c+1)/2$ are all multiples of 16, which is good news for a computer with a word length of 16 or 8 bits. Below we present the C code for calculating a field square root on an 8-bit processor, where the input is an array `alpha[.]` of 34 bytes and the square root is an array `beta[.]` also of 34 bytes (and initially cleared to zero).

```
for (i=0;i<34;i++)
{
    n=i/2;
    w=alpha[i];          /* get a byte of alpha[.] */

    we=evens[w];         /* extract 4 even bits */
    wo=odds[w];          /* extract 4 odd bits  */
    i++;
    w=alpha[i];          /* get another byte of alpha */
    we|=(evens[w]<<4);   /* create 8 bits of evens */
    wo|=(odds[w]<<4);    /* create 8 bits of odds  */

    beta[n]^=we;         /* store evens */
    beta[n+17]=wo;       /* abutt odds  */
    beta[n+13]^=wo;      /* multiply by other terms of zeta */
    beta[n+11]^=wo;
    beta[n+7]^=wo;
}
```