Chapter 11

Finite Field Arithmetic

Christophe Doche

Contents in Brief

In this chapter, we are mainly interested in performance; see Section 2.3 for a theoretical presentation of finite fields. In the following, we consider three kinds of fields that are of great cryptographic importance, namely prime fields, extension fields of characteristic 2, and optimal extension fields. We will describe efficient methods for performing elementary operations, such as addition, multiplication, inversion, exponentiation, and square roots. The material that we give is implicitly more related to a software approach; see Chapter 26 for a presentation focused on hardware. Efficient finite field arithmetic is crucial in efficient elliptic or hyperelliptic curve cryptosystems and is the subject of abundant literature [JUN 1993, LINI 1997, SHP 1999]. See also the preliminary version of a book written by Shoup and available online [SHO], introducing basic concepts from computational number theory and algebra, and including all the necessary mathematical background.

There are some software packages implementing the algorithms described below, such as ZEN [ZEN], which is a set of optimized C libraries dedicated to finite fields. There are also more general libraries like NTL [NTL] or Lidia [LIDIA]. In addition, several computer algebra systems contain functions for handling finite fields, for example Magma [MAGMA].

11.1 Prime fields of odd characteristic

Most of the algorithms detailed here carry through as well to $\mathbb{Z}/N\mathbb{Z}$ for arbitrary moduli N, usually with some obvious modifications. However, here we are mainly interested in prime moduli. Methods to find either an industrial-grade prime or a certified prime number p of the desired size are described in Chapter 25.

11.1.1 Representations and reductions

For representing finite prime fields we usually use the isomorphism between $\mathbb{F}_p$ and $\mathbb{Z}/p\mathbb{Z}$. Elements of $\mathbb{Z}/p\mathbb{Z}$ are equivalence classes and we have to choose a *representative*, that is a particular element in the class, to perform computations. The most standard choice is to represent $x \in \mathbb{Z}/p\mathbb{Z}$ by the unique integer in $[0, p-1]$, which is in the class of x. We can also use other representatives such as the ones belonging to $[-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$ or even an *incompletely reduced number*, which is not uniquely determined, for it belongs to an interval of length greater than p; see Remark 26.45 (ii) and [YAN 2001, YASA$^+$ 2002].

Given an integer u of arbitrary size we must be able to reduce it, i.e., to find the integer in $[0, p-1]$ which is congruent to u modulo p. This *modular reduction* is achieved by computing the remainder of a Euclidean division.

However, since all the reductions are performed modulo the same prime number p, there exist several improvements which, for instance, involve some precomputations. The most popular ones are certainly the Montgomery and the Barrett methods; see Section 10.4. In this case the cost of a reduction of a $2n$-word integer modulo an n-word integer is asymptotically equal to a size n multiplication.

To compute the remainder faster, other ideas include the choice of a special modulus allowing a fast reduction; see Algorithm 10.25 for the use of a different normalization than the one initially suggested by Knuth in Algorithm 10.30. Quisquater first proposed this method, which speeds up the determination of an approximation of the quotient; see Remark 10.33 and Example 10.34. However, this reduction method will increase the length of the modulus p by at least one digit, resulting in additional multiplications when performing arithmetic in $\mathbb{F}_p$.

In the remainder, we address prime field arithmetic itself. Whatever representation is chosen, prime field addition and subtraction algorithms are straightforward in terms of the corresponding multiprecision algorithms for integers, cf. Algorithms 10.3 and 10.5. For example, with classical representation, if u, v are integers in $[0, p-1]$, then $u + v < 2p$ and the modular addition of u and v is simply $u + v$ or $u + v - p$. In the same way, modular subtraction of u and v is $u - v$ if $u \geqslant v$ and $u - v + p$ when $u < v$. Montgomery representation is compatible with addition and subtraction as well.

Now let us investigate multiplication algorithms in $\mathbb{F}_p$.

11.1.2 Multiplication

Except special methods, like the one explained in [CHCH 1999] that involves precomputations, there are mainly two ways two perform a modular multiplication. The first one consists of a simple integer multiplication with the schoolbook or Karatsuba methods, i.e., one of the Algorithms 10.8 or 10.11, followed by a reduction. The choice of the algorithm depends on the nature and the size of the integer operands as well as on the computer architecture used.

The second one is designed as a single operation. In this case elementary multiplications and reductions are interleaved so that the size of the intermediate results remains bounded. These two options apply to Montgomery representation as well.

11.1.2.a Classical representation

Algorithm 11.1 is a general scheme to compute a modular multiplication. We have

$$uv \equiv \left(\sum_{i=0}^{n-1} u_i v b^i \right) \pmod{p}$$

which can be written

$$uv \equiv \Big(\big(\ldots\big((u_{n-1}v \bmod p)b + u_{n-2}v \bmod p\big)b + \cdots + u_1 v \bmod p\Big)b + u_0 v\Big) \pmod p.$$

If we set $t_{-1} = 0$ and $t_i = (t_{i-1}b + u_{n-i-1}v) \bmod p$ then $t_{n-1} \equiv uv \pmod p$. We deduce the following algorithm:

Algorithm 11.1 Interleaved multiplication-reduction of multiprecision integers

INPUT: Two n-word integers $u = (u_{n-1} \ldots u_0)_b$ and v.
OUTPUT: An integer t such that $t \equiv uv \pmod p$.

1. $t \leftarrow 0$
2. **for** $i = 0$ **to** $n-1$ **do**
3. $\quad t \leftarrow tb + u_{n-i-1}v$
4. $\quad$ approximate $q = \lfloor t/p \rfloor$ with $\hat{q}$ [see methods below]
5. $\quad t \leftarrow t - \hat{q}p$
6. **return** t

The approximation of q can be achieved by Knuth, Barrett, or Quisquater methods. Knuth's approach has already been explained, cf. Algorithm 10.30. The last two methods are described in detail by Dhem in [DHE 1998]. See also Section 10.4.1 and Remark 10.33. Barrett writes

$$q = \left\lfloor \frac{t}{p} \right\rfloor = \left\lfloor \frac{\frac{t}{2^{n-1}} \frac{2^{2n}}{p}}{2^{n+1}} \right\rfloor$$

where n is the number of bits of p, then approximates q by

$$\left\lfloor \frac{\left\lfloor \frac{t}{2^{n-1}} \right\rfloor \left\lfloor \frac{2^{2n}}{p} \right\rfloor}{2^{n+1}} \right\rfloor$$

where we assume that $\left\lfloor \frac{2^{2n}}{p} \right\rfloor$ has been precomputed. Dhem introduced a more general variant, namely

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{t}{2^{n+\beta}} \right\rfloor R}{2^{\alpha-\beta}} \right\rfloor \quad \text{with} \quad R = \left\lfloor \frac{2^{n+\alpha}}{p} \right\rfloor.$$

These additional parameters allow us to perform corrections on the remainder only at the end of the whole multiplication process, when they are suitably tuned. Algorithm 11.1 works at a word level and if $b = 2^\ell$ then optimal results are obtained with $\alpha = \ell + 3$ and $\beta = -2$. In this case we have $q - 1 \leqslant \hat{q} \leqslant q$ and the intermediate results grow moderately. More precisely, given $u, v < 2^{n+1}$ then $t \equiv uv \pmod p$ returned by Algorithm 11.1 is less than 2^{n+1}. To get the final result, at most one subtraction is needed. This implies also that an exponentiation or any other long computation can be done with only one correction at the end of the whole process with the same choice of parameters.

Quisquater's method [QUI 1990, QUI 1992] consists in multiplying p by a suitable coefficient δ such that the reduction modulo δp is easy. Set

$$\delta = \left\lfloor \frac{2^{n+\ell+2}}{p} \right\rfloor \quad \text{and get} \quad \hat{q} = \left\lfloor \frac{t}{2^{n+\ell+2}} \right\rfloor.$$

The $(\ell + 2)$ highest bits of δp are now equal to 1 and the corresponding quotient is obviously determined, since it is simply equal to the most significants bits of t. There is a fast way to compute δ, namely put

$$\hat{\delta} = \left\lfloor \frac{2^{2\ell+6}}{\left\lfloor \frac{p}{2^{n-\ell-3}} \right\rfloor} \right\rfloor,$$

which verifies $\delta \leqslant \hat{\delta} \leqslant \delta + 1$, and a simple test gives the correct value. It is also possible to reduce the size of δ; see [DHE 1998, p. 24]. This normalization avoids overflows in Algorithm 11.1 while computing a multiplication or even a modular exponentiation.

Now suppose that one has the result $x \bmod \delta p$ while we still want $x \bmod p$. For this, we could perform $(x \bmod \delta p) \bmod p$ but since an exact division is faster (see Section 10.5.3) it is better to compute

$$x \bmod p = \frac{\delta x \bmod \delta p}{\delta}. \tag{11.1}$$

Note that this method has been used in several smart cards; see for example [QUWA+ 1991] or [FEMA+ 1996].

11.1.2.b Montgomery multiplication

Montgomery representation, see Section 10.4.2, was in fact introduced to carry out quick modular multiplications. This property comes from the equality

$$\big((xR \bmod p)(yR \bmod p)R^{-1} \bmod p\big) = (xyR) \bmod p$$

which implies that $\text{REDC}([x][y]) = [xy]$. Recall that Montgomery reduction is also useful to convert elements between normal and Montgomery representations. Indeed, $[x] = \text{REDC}(xR')$ where $R' = R^2 \bmod p$ has been precomputed and stored, and $\text{REDC}([x]) = x$.

Example 11.2 Take $p = 2011$, $b = 2^3$, $R = 4096$ so that $R' = 1454$. Let $x = 45$, $y = 97$. Then

$$\begin{aligned} [x] &= \text{REDC}(45 \times 1454) = 1319 = (2447)_8 \\ [y] &= \text{REDC}(97 \times 1454) = 1145 = (2171)_8 \\ [x][y] &= 1510255 \\ [xy] &= \text{REDC}(1510255) = 1250 = (2342)_8 \\ xy &= \text{REDC}(1250) = 343. \end{aligned}$$

One checks that $45 \times 97 \equiv 343 \pmod{2011}$.

Of course, Montgomery method is completely irrelevant when only one product is needed. Instead, operands are converted to and kept in Montgomery representation as long as possible. For instance, if one wants $[x^2y]$, simply perform $\text{REDC}([x][xy])$.

The following algorithm computes directly $\text{REDC}(uv)$ given multiprecision integers u and v in Montgomery representation. It combines Algorithms 10.8 and 10.22.

Algorithm 11.3 Multiplication in Montgomery representation

INPUT: An n-word integer $p = (p_{n-1} \dots p_0)_b$ prime to b, $R = b^n$, $p' = -p^{-1} \bmod b$ and two n-word integers $u = (u_{n-1} \dots u_0)_b$ and $v = (v_{n-1} \dots v_0)_b$ such that $0 \leqslant u, v < p$.
OUTPUT: The n-word integer $t = (t_{n-1} \dots t_0)_b$ equal to $\text{REDC}(uv) = (uvR^{-1}) \bmod p$.

1. $t \leftarrow 0$
2. **for** $i = 0$ **to** $n - 1$ **do**
3. $\quad m_i \leftarrow \big((t_0 + u_i v_0)p'\big) \bmod b$

4. $t \leftarrow (t + u_i v + m_i p)/b$
5. **if** $t \geqslant p$ **then** $t \leftarrow t - p$
6. **return** t

Remarks 11.4

(i) If R is chosen such that $R > 4p$ and if u and v are positive and less than $2p$ then $\text{REDC}(uv)$ is bounded by $2p$ as well. This means that it is possible to avoid the subtraction in Line 5 of Algorithm 11.1 during long computations as exponentiations. At the end of the whole process the result is normalized in $\mathbb{Z}/p\mathbb{Z}$ at the cost of a single subtraction [LEN 2002].

(ii) See [KOAC$^+$ 1996] for a comparison of different variations of Montgomery method.

Example 11.5 Let us perform again the computation of Example 11.2 but at a word level with Algorithm 11.3. Let $u = [45] = 1319 = (2447)_8$ and $v = [97] = 1145 = (2171)_8$. Then

i	u_i	t_0	m_i	$u_i v$	$m_i p$	$t + u_i v + m_i p$	t
—	—	—	—	—	—	—	0
0	7	0	3	$(17517)_8$	$(13621)_8$	$(33340)_8$	$(3334)_8$
1	4	4	0	$(10744)_8$	0	$(14300)_8$	$(1430)_8$
2	4	0	4	$(10744)_8$	$(17554)_8$	$(32150)_8$	$(3215)_8$
3	2	5	3	$(4362)_8$	$(13621)_8$	$(23420)_8$	$(2342)_8$

One obtains $\text{REDC}(uv) = (2342)_8 = [xy] = 1250$ as previously.

Concerning modular squaring, the computation of the square of an integer can be achieved faster (see Section 10.3.3); however the reduction takes the same time as in the case of a modular multiplication. Note that there are some dedicated methods like [HOOH$^+$ 1996], which are worth being implemented if modular exponentiation is to be computed, as squarings are a very frequent operation.

11.1.3 Inversion and division

To get the inverse of some integer x, we can use the multiplicative structure of $\mathbb{F}_p^*$ which implies that $x^{p-2} \times x = x^{p-1} \equiv 1 \pmod{p}$. However, Collins [COL 1969] showed that the average number of arithmetic operations required by this approach is nearly twice as large as for the Euclid extended algorithm, which computes integers u, v such that $xu + pv = 1$. See Section 10.6 for an exhaustive presentation of extended gcd algorithms.

In the following section more specific methods are described, including Montgomery inversion and a useful trick to compute several inverses simultaneously.

11.1.3.a Modular inversion

We start with a simplified and improved version of Algorithm 10.6.3, to compute the inverse of x modulo p, introduced by Brent and Kung [BRKU 1983] and known as the *plus-minus* method.

Algorithm 11.6 Plus-minus inversion method

INPUT: An odd modulus p and an integer $x < p$ prime to p.
OUTPUT: The integer $x^{-1} \bmod p$.

1. $A \leftarrow x$, $B \leftarrow p$, $U_A \leftarrow 1$, $U_B \leftarrow 0$ and $\delta \leftarrow 0$
2. **while** $|A| > 0$ **do**
3. **while** $A \equiv 0 \pmod 2$ **do**
4. $A \leftarrow A/2$, $U \leftarrow (U/2) \bmod p$ and $\delta \leftarrow \delta - 1$
5. **if** $\delta < 0$ **then**
6. $T \leftarrow A$, $A \leftarrow B$, $B \leftarrow T$
7. $T \leftarrow U_A$, $U_A \leftarrow U_B$, $U_B \leftarrow T$
8. $\delta \leftarrow -\delta$
9. **if** $(A + B) \equiv 0 \pmod 4$ **then**
10. $A \leftarrow (A + B)/2$ and $U_A \leftarrow \big((U_A + U_B)/2\big) \bmod p$
11. **else** $A \leftarrow (A - B)/2$ and $U_A \leftarrow \big((U_A - U_B)/2\big) \bmod p$
12. **if** $B = 1$ **then** $u \leftarrow U_B$ **else** $u \leftarrow p - U_b$
13. **return** u

Remarks 11.7

(i) Algorithm 11.6 is based on the observation that if A and B are both odd then either $A+B$ or $A-B$ is divisible by 4. If $A+B \equiv 0 \pmod 4$ then $\gcd(A, B) = \gcd\big((A+B)/2, B\big)$ with $(A+B)/2$ even and $|(A+B)/2| \leqslant \max\{|A|, |B|\}$. Similar results hold if $A-B \equiv 0 \pmod 4$.

(ii) The counter δ is used to compare A and B, as the direct comparison can be time-consuming, especially in hardware. Further improvements are described in [TAK 1998, MEBU$^+$ 2004]. The corresponding algorithms are well suited for hardware realizations and can be implemented in parallel.

Example 11.8 Take $p = 2^7 - 1 = 127$ and $x = 45$. In the following table are given the values of δ, A, B, U_A, and U_B at the end of the main while loop.

δ	A	B	U_A	U_B
0	86	127	64	0
1	42	43	111	32
0	32	43	12	32
5	22	1	40	48
4	6	1	34	48
3	2	1	96	48
2	0	1	0	48

So, the inverse of 45 modulo 127 is 48.

In a case where the modulus p is prime, one can also use a completely different algorithm due to Thomas et al. [THKE$^+$ 1986] to compute the inverse of x modulo p.

Algorithm 11.9 Prime field inversion

INPUT: A prime modulus p and an integer x prime to p.
OUTPUT: The integer $x^{-1} \bmod p$.

1. $z \leftarrow x \bmod p$ and $u \leftarrow 1$
2. **while** $z \neq 1$ **do**
3. $\quad q \leftarrow -\lfloor p/z \rfloor$
4. $\quad z \leftarrow p + qz$
5. $\quad u \leftarrow (qu) \bmod p$
6. **return** u

Remarks 11.10

(i) Algorithm 11.9 is very simple to implement and is reported to be faster than the extended Euclidean algorithm for some types of primes, for example Mersenne primes. Indeed, in this case, the computation of q in Line 3 can be carried out very efficiently. Note that there exists also a dedicated algorithm to compute an inverse modulo a Mersenne prime [CRPO 2001, p. 428].

(ii) In general, the number of iterations needed by Algorithm 11.9 is less than for extended gcd algorithms.

(iii) The modular division $(k/x) \bmod p$ can be directly obtained with Algorithms 11.6 and 11.9. Namely, modify the first line of each algorithm and replace the statements $U_A \leftarrow 1$ and $u \leftarrow 1$ respectively by $U_A \leftarrow k$ and $u \leftarrow k$.

Example 11.11 Again, take $p = 2^7 - 1 = 127$ and $x = 45$. Here are the values of q, z, and u at the end of the while loop.

q	z	u
-2	37	125
-3	16	6
-7	15	85
-8	7	82
-18	1	48

Again, we find that the inverse of 45 modulo 127 is 48. In this case only 5 iterations are needed instead of 7 for Algorithm 11.6, cf. Example 11.8.

11.1.3.b Montgomery inversion and division

Montgomery's article also deals with inversions and divisions [MON 1985]. Kaliski [KAL 1995] develops specific algorithms to compute them. Recall the settings of Section 10.4.2 and let u be an integer. Then the *Montgomery inverse* of u is defined as $\text{INV}(u) = (u^{-1}R^2) \bmod p$. So if $u = [x] = xR$, we see that $\text{INV}([x]) = (x^{-1}R) \bmod p = [x^{-1}]$. Thus we have

$$\text{REDC}([x]\,\text{INV}[x]) = R \bmod p = [1].$$

Algorithm 11.12 Montgomery inverse in Montgomery representation

INPUT: Two n-word integers u and p such that $u \in [1, p-1]$. The integer $R = 2^m = b^n$ and the precomputed value $R' = R^2 \bmod p$.
OUTPUT: The n-word integer v equal to $\text{INV}(u) = (u^{-1}R^2) \bmod p$.

1. $r \leftarrow u, s \leftarrow 1, t \leftarrow p, v \leftarrow 0$ and $k \leftarrow 0$
2. **while** $r > 0$ **do**
3. **if** $t \equiv 0 \pmod 2$ **then** $t \leftarrow t/2$ and $s \leftarrow 2s$
4. **else if** $r \equiv 0 \pmod 2$ **then** $r \leftarrow r/2$ and $v \leftarrow 2v$
5. **else if** $t > r$ **then** $t \leftarrow (t-r)/2, v \leftarrow v+s$ and $s \leftarrow 2s$
6. **else** $r \leftarrow (r-t)/2, s \leftarrow v+s$ and $v \leftarrow 2v$
7. $k \leftarrow k+1$
8. **if** $v \geqslant p$ **then** $v \leftarrow v-p$
9. $v \leftarrow p-v$
10. **if** $k < m$ **then** $v \leftarrow \text{REDC}(vR')$ and $k \leftarrow k+m$
11. $v \leftarrow \text{REDC}(vR')$
12. $v \leftarrow \text{REDC}(v2^{2m-k})$
13. **return** v

Remarks 11.13

(i) Lines 1 to 9 compute the so-called *almost Montgomery inverse* i.e., $(u^{-1}2^k) \bmod p$ for some k such that $c \leqslant k \leqslant m+c$, where c is the binary length of p.

(ii) It is possible to change the end of Algorithm 11.12 in order to compute directly the inverse of u, i.e., $u^{-1} \bmod p$ with one or two extra Montgomery multiplications, namely replace Lines 10, 11, and 12 by

10. **if** $k > m$ **then** $v \leftarrow \text{REDC}(v)$ and $k \leftarrow k-m$
11. $v \leftarrow \text{REDC}(v2^{m-k})$

(iii) To divide $[x]$ by $[y]$ it suffices to do $\text{REDC}\big([x]\,\text{INV}([y])\big)$ and get $[xy^{-1}]$.

Example 11.14 With the settings of Example 10.24, let us compute the Montgomery inverse of $[45] = 1319$. Since $p = 2011$ is a 4-word integer in base 8, we have $m = 12$.

- After Line 9, Algorithm 11.12 has computed the almost Montgomery inverse of 1319, which is 1252, and found $k = 17$. This means that $1319^{-1} \times 2^{17} \bmod 2011 = 1252$.
- Lines 10 and 11 compute $\text{REDC}(1252R') = 142$ and finally $\text{REDC}(142 \times 2^{24-17}) = 1387$, which is the Montgomery inverse of 1319. We check that $\text{REDC}(1319 \times 1387) = 74 \equiv R \pmod p$.
- If we want the inverse of 1319 modulo 2011, we perform $\text{REDC}(1252) = 1267$ and set $k \leftarrow 5$. Then $\text{REDC}(1267 \times 2^{12-5}) = 1485 \equiv 1319^{-1} \pmod{2011}$.

The next section allows us to compute the inverse of several numbers modulo the same number p.

11.1.3.c Simultaneous inversion

One needs *a priori* j extended $\gcd$ computations to find the inverses of the j elements $a_1, \ldots, a_j$ modulo p. Here we present a trick of Montgomery that allows us to do the same with only one extended $\gcd$ and $3j-3$ multiplications modulo p. The basic idea is to get the inverse of $\prod_i a_i$ and to multiply it by suitable terms to recover $a_j^{-1}, \ldots, a_1^{-1}$ [COH 2000].

Algorithm 11.15 Simultaneous inversion modulo p

INPUT: A positive integer p and j integers $a_1, \ldots, a_j$ not zero modulo p.
OUTPUT: The inverses $b_1, \ldots, b_j$ of the $a_1, \ldots, a_j$ modulo p.

1. $c_1 \leftarrow a_1$
2. **for** $i = 2$ **to** j **do** $c_i \leftarrow a_i c_{i-1}$
3. compute (u, v, d) with $uc_j + vp = d$ [d is equal to 1]
4. **for** $i = j$ **down to** 2 **do**
5. $\quad b_i \leftarrow (uc_{i-1}) \bmod p$ and $u \leftarrow (ua_i) \bmod p$
6. $b_1 \leftarrow u$
7. **return** $(b_1, \ldots, b_j)$

Remarks 11.16

(i) Let N be a nonprime modulus. If one tries to apply Algorithm 11.15 to the nonzero residues $a_1, \ldots, a_j$ modulo N, there are two possibilities. If $a_1, \ldots, a_j$ are all coprime to N then Algorithm 11.15 returns $a_1^{-1}, \ldots, a_j^{-1}$ modulo N. If at least one integer is not coprime to N then the $\gcd$ computed in Line 3 is different from 1. In this case, if the Lines 4 to 7 of Algorithm 11.15 are replaced by the following statements

4. **if** $d = N$ **then**
5. $\quad i \leftarrow 1$
6. $\quad$ **repeat**
7. $\quad\quad d \leftarrow \gcd(a_i, N)$ and $i \leftarrow i + 1$
8. $\quad$ **until** $d > 1$
9. **return** d

then a nontrivial factor of N is returned.

(ii) This modified algorithm is especially useful for Lenstra's elliptic curve method, cf. Section 25.3.3, where one tries to find factors of N by computing scalar multiples on a curve modulo N.

11.1.4 Exponentiation

This part deals with specific exponentiation methods for finite fields $\mathbb{F}_p$. The general introduction to the subject can be found in Chapter 9.

11.1.4.a Ordinary exponentiation

To compute x^n, for $x \in \mathbb{F}_p$, one could perform the exponentiation in $\mathbb{Z}$ and then reduce the result. Of course, this approach is completely inefficient even for rather small n. However, a systematic

reduction after each intermediate step, i.e., a squaring or a multiplication, seems inadequate as well since a modular reduction is quite slow. So, a compromise must be found. One can also use Barrett or Quisquater multiplication algorithms without the remainder correction steps; see Section 11.1.2.a. With appropriate settings, intermediate results are kept bounded such that they still fit in the allocated space, and only at the end of the exponentiation one corrects the result so that it belongs to $[0, p-1]$.

11.1.4.b Montgomery exponentiation

All algorithms presented in Chapter 9 can be adapted to Montgomery representation. The changes are always the same and rather simple: as explained in Section 11.1.2.b, one converts to and from Montgomery representation only for input/output, so any amount of operations can be done in between. These ideas are illustrated in the following adaptation of the classical square and multiply algorithm, cf. Section 9.1.1.

Algorithm 11.17 Binary exponentiation using Montgomery representation

INPUT: An element x of $\mathbb{F}_p$, a positive integer $n = (n_{t-1} \dots n_0)_2$ such that $n_{t-1} = 1$, the integers R and $R' = R^2 \bmod p$.
OUTPUT: The element $x^n \in \mathbb{F}_p$.

1. $y \leftarrow R \bmod p$ and $t \leftarrow \text{REDC}(xR')$
2. **for** $i = t-1$ **down to** 0
3. $\quad y \leftarrow \text{REDC}(y^2)$
4. $\quad$ **if** $n_i = 1$ **then** $y \leftarrow \text{REDC}(ty)$
5. **return** $\text{REDC}(y)$

Remark 11.18 Conversion to Montgomery representation is done in Line 1. One has $y = [1]$ and $t = [x]$. In the for loop at each step a Montgomery squaring and possibly a Montgomery multiplication is performed. Finally we come back to the standard representation by a Montgomery reduction. At the end, $y = [x^n]$ so that $\text{REDC}(y) = x^n$, as expected. Note that also here incomplete reduction can be applied.

11.1.5 Squares and square roots

Given a nonzero integer a modulo p, the Legendre symbol $\left(\frac{a}{p}\right)$ defined in Section 2.3.4 is equal to 1 if and only a is a quadratic residue modulo p. From the reciprocity law (2.6) and Theorem 2.103, it is easy to derive an efficient way to compute it.

Algorithm 11.19 Legendre symbol

INPUT: An integer a and an odd prime number p.
OUTPUT: The Legendre symbol $\left(\frac{a}{p}\right)$.

1. $k \leftarrow 1$
2. **while** $p \neq 1$ **do**
3. $\quad$ **if** $a = 0$ **then return** 0
4. $\quad v \leftarrow 0$
5. $\quad$ **while** $a \equiv 0 \pmod 2$ **do** $v \leftarrow v+1$ and $a \leftarrow a/2$

6. **if** $v \equiv 1 \pmod 2$ **and** $p \equiv \pm 3 \pmod 8$ **then** $k \leftarrow -k$
7. **if** $a \equiv 3 \pmod 4$ **and** $p \equiv 3 \pmod 4$ **then** $k \leftarrow -k$
8. $r \leftarrow a, a \leftarrow p \bmod r$ **and** $p \leftarrow r$
9. **return** k

Remark 11.20 This algorithm is useful, for example, to determine the number of points lying on an elliptic or hyperelliptic curve defined over a finite field of small prime order, cf. Chapter 17.

Example 11.21 Take the prime $p = 163841$, $a = 109608$ and let us compute $\left(\frac{a}{p}\right)$ with Algorithm 11.19. The next table displays the values of r, a, v and k after Line 8.

r	a	v	k
13701	13130	3	1
6565	571	1	−1
571	284	0	−1
71	3	2	1
3	2	0	−1
1	0	1	1

These computations reflect the following sequence of equalities

$$\begin{aligned}
\left(\frac{109608}{163841}\right) &= \left(\frac{8}{163841}\right)\left(\frac{13701}{163841}\right) \\
&= \left(\frac{13130}{13701}\right) = \left(\frac{2}{13701}\right)\left(\frac{6565}{13701}\right) \\
&= -\left(\frac{571}{6565}\right) \\
&= -\left(\frac{284}{571}\right) = -\left(\frac{4}{571}\right)\left(\frac{71}{571}\right) \\
&= \left(\frac{3}{71}\right) \\
&= -\left(\frac{2}{3}\right) \\
&= 1.
\end{aligned}$$

So, 109608 is a quadratic residue modulo 163841.

When it is known that a is a square, it is often required to determine x such that $x^2 \equiv a \pmod p$. For instance, this occurs to actually find a point lying on an elliptic or hyperelliptic curve.

Lemma 11.22 Given a quadratic residue $a \in \mathbb{F}_p$, there are explicit formulas when $p \not\equiv 1 \pmod 8$ to determine $x \in \mathbb{F}_p$ such that $x^2 \equiv a \pmod p$. Namely,

- $x \equiv \pm a^{(p+1)/4} \pmod p$ if $p \equiv 3 \pmod 4$
- $x \equiv \pm a^{(p+3)/8} \pmod p$ if $p \equiv 5 \pmod 8$ and $a^{(p-1)/4} = 1$
- $x \equiv \pm 2a(4a)^{(p-5)/8} \pmod p$ if $p \equiv 5 \pmod 8$ and $a^{(p-1)/4} = -1$.

When $p \equiv 1 \pmod 8$ an algorithm of Tonelli and Shanks solves the problem. In fact, this algorithm is correct for all primes.

Algorithm 11.23 Tonelli and Shanks square root computation

INPUT: A prime p and an integer a such that $\left(\frac{a}{p}\right) = 1$.
OUTPUT: An integer x such that $x^2 \equiv a \pmod{p}$.

1. write $p - 1 = 2^e r$ with r odd [see the beginning of Section 10.5, p. 185]
2. choose n at random such that $\left(\frac{n}{p}\right) = -1$
3. $z \leftarrow n^r \bmod p$, $y \leftarrow z$, $s \leftarrow e$ **and** $x \leftarrow a^{(r-1)/2} \bmod p$
4. $b \leftarrow (ax^2) \bmod p$ **and** $x \leftarrow (ax) \bmod p$
5. **while** $b \not\equiv 1 \pmod{p}$
6. $\quad m \leftarrow 1$
7. $\quad$ **while** $b^{2^m} \not\equiv 1 \pmod{p}$ **do** $m \leftarrow m + 1$
8. $\quad t \leftarrow y^{2^{s-m-1}} \bmod p$, $y \leftarrow t^2 \bmod p$ **and** $s \leftarrow m$
9. $\quad x \leftarrow (tx) \bmod p$ **and** $b \leftarrow (yb) \bmod p$
10. **return** x

Remarks 11.24

(i) Algorithm 11.23 works in the maximal 2-group of order 2^e of $\mathbb{F}_p^*$ generated by some element z. If $m = s$ after Line 7, this implies that a is not a quadratic residue modulo p. Otherwise a^r is a square in this subgroup and there is an even k less than e such that $a^r z^k \equiv 1 \pmod{p}$. The square root is then given by $x \equiv a^{(r+1)/2} z^{k/2} \pmod{p}$. A variant of Algorithm 11.23 finds $k/2$ by a bit by bit approach [KOB 1994].

(ii) The number of loops performed within the while loop beginning in Line 5 is bounded by e since s is strictly decreasing at each loop.

(iii) The expected running time of Algorithm 11.23 is $O(e^2 \lg^2 p)$.

Example 11.25 Let us compute the square root of 109608 modulo $p = 163841$ with Algorithm 11.23. First one sees that $e = 15$ and $r = 5$. The quadratic nonresidue n found at random in Line 2 is 6558. In the following, we state the values of the principal variables before the while loop in Line 4 and at the end of it Line 9. One can see also that $ab - x^2$, $y^{2^{s-1}}$ and $b^{2^{s-1}}$ are invariant throughout the execution of the algorithm.

m	z	y	x	b	t	ab	x^2	$y^{2^{s-1}}$	$b^{2^{s-1}}$
—	12002	12002	13640	100808	—	90065	90065	-1	1
13	—	82347	78996	68270	31765	155849	155849	-1	1
12	—	140942	104389	56092	82347	162252	162252	-1	1
6	—	81165	18205	57313	52992	135523	135523	-1	1
5	—	38297	90687	101925	81165	132974	132974	-1	1
3	—	101925	97748	39338	119418	119748	119748	-1	1
2	—	39338	121372	163840	101925	54233	54233	-1	1
1	—	163840	41155	1	39338	109608	109608	-1	1

One checks that $41155^2 \equiv 109608 \pmod{p}$.

When the 2-adic valuation e of $p-1$ is large, as in the previous example, it is better to use another algorithm that works in the quadratic extension $\mathbb{F}_{p^2}$.

Algorithm 11.26 Square root computation

INPUT: A prime p and an integer a such that $\left(\frac{a}{p}\right) = 1$.
OUTPUT: An integer x such that $x^2 \equiv a \pmod{p}$.

1. choose b at random such that $\left(\frac{b^2-4a}{p}\right) = -1$
2. $f(X) \leftarrow X^2 - bX + a$ [$f(X)$ is irreducible over $\mathbb{F}_p$]
3. $x \leftarrow X^{(p+1)/2} \pmod{f(X)}$
4. **return** x

Remarks 11.27

(i) If θ is a root of $f(X)$ then θ^p is the other one and therefore $\theta^{p+1} = a$. So x as defined in Line 3 satisfies $x^2 \equiv a \pmod{f(X)}$. It remains to show that x is in fact an element of $\mathbb{F}_p$. As $a^{(p-1)/2} = 1$ we have $X^{(p^2-1)/2} \equiv 1 \pmod{f(X)}$ so that $x^p \equiv x \pmod{f(X)}$.

(ii) The expected running time of Algorithm 11.26 is $O(\lg^3 p)$.

Example 11.28 With the same initial values as in Example 11.25, Algorithm 11.26 first finds at random an irreducible polynomial over $\mathbb{F}_p$, in this case, for instance, $f(X) = X^2 + 27249X + 109608$. Then it computes $X^{(p+1)/2}$, which is equivalent to 41155 modulo $f(X)$.

11.2 Finite fields of characteristic 2

See Section 2.3.2 for an introduction to algebraic extension of fields. Arithmetic in extension fields of $\mathbb{F}_q$ where q is some power of 2 relies on elementary computer operations like exclusive disjunction and shifts. Note that in general q is simply equal to 2. This allows very efficient implementations, especially in hardware, and gives finite fields of characteristic 2 a great importance in cryptography.

11.2.1 Representation

See Section 2.3.3 for a presentation of the different finite field representation systems. In the following we shall focus on efficient implementation techniques used in cryptography. As $\mathbb{F}_{2^d}$ is a vector space of dimension d over $\mathbb{F}_2$, an element can be viewed as a sequence of d coefficients equal to 0 or 1. Therefore it is internally stored as a sequence of bits and the techniques introduced for multiprecision integers apply with some slight modifications. Two kinds of basis are commonly used. In polynomial representation, it is $(1, X, \ldots, X^{d-1})$, whereas with a normal basis it is $(\alpha, \alpha^2, \ldots, \alpha^{2^{d-1}})$, cf. Section 2.3.3. Let us first describe polynomial representation.

11.2.1.a Irreducible polynomial representation

Let $m(X) \in \mathbb{F}_q[X]$ be an irreducible polynomial of degree d and $\big(m(X)\big)$ the principal ideal generated by $m(X)$. Then $\mathbb{F}_q[X]/\big(m(X)\big)$ is the finite field with q^d elements. Formula (2.4)

proves that there exists an irreducible polynomial of degree d for each positive d but the proof is not constructive. To find such a polynomial, we can consider a random polynomial and test its irreducibility. Since (2.4) shows that the probability for a monic polynomial of degree d to be irreducible is close to $1/d$, we should find one after d attempts on average. There is a variety of polynomial irreducibility tests. For example Rabin [RAB 1980] proved the following.

Lemma 11.29 Let $m(X) \in \mathbb{F}_q[X]$ of degree d and let $p_1, \ldots, p_k$ be the prime divisors of d. Then $m(X)$ is irreducible over $\mathbb{F}_q$ if and only if

- $\gcd\big(m(X), X^{q^{d/p_i}} - X\big) = 1$, for $i = 1, \ldots, k$
- $m(X)$ divides $X^{q^d} - X$.

For a deterministic method to find an irreducible polynomial see [SHO 1994b].

Once $m(X)$ has been found, computations are done modulo this irreducible polynomial and reduction is a key operation. For this we need to divide two polynomials with coefficients in a field. Every irreducible polynomial of degree d can be used to build $\mathbb{F}_{q^d}$; however, some special polynomials offer better performance, e.g., monic sparse polynomials are proposed in [SCOR$^+$ 1995].

Usually, one uses *trinomials* or *pentanomials* since binomials and quadrinomials are always divisible by $X + 1$ and so, except for $X + 1$ itself, are never irreducible in $\mathbb{F}_q[X]$. The existence for every d of an irreducible degree d trinomial or pentanomial is still an open question, but this is the case at least for all $d \leqslant 10000$ [SER 1998].

A trinomial $X^d + X^k + 1$ is reducible if both d and k are even as then $X^d + X^k + 1 = (X^{d/2} + X^{k/2} + 1)^2$. Eliminating this trivial case, Swan [SWA 1962] proves the following.

Lemma 11.30 The trinomial $X^d + X^k + 1$, where at least one of d and k is odd, has an even number of factors if and only if one of the following holds

- d is even, k is odd, $d \neq 2k$ and $\frac{dk}{2} \equiv 0$ or $1 \pmod 4$
- d is odd, $d \equiv \pm 3 \pmod 8$, k is even and k does not divide $2d$
- d is odd, $d \equiv \pm 1 \pmod 8$, k is even and k divides $2d$.

It follows that irreducible trinomials do not exist when $d \equiv 0 \pmod 8$ and are rather scarce for $d \equiv 3$ or $5 \pmod 8$. In Table 11.1, we give irreducible polynomials over $\mathbb{F}_2$ of degree less than or equal to 500. More precisely, the coefficients d, k_1 in the table stand for the trinomial $X^d + X^{k_1} + 1$. In case there is no trinomial of degree d, the sequence d, k_1, k_2, k_3 is given for the pentanomial $X^d + X^{k_1} + X^{k_2} + X^{k_3} + 1$. For each d the coefficient k_1 is chosen to be minimal, then k_2 and so on.

For these sparse polynomials there is a specific reduction algorithm [GANÖ 2005]. The nonrecursive version is given hereafter.

Algorithm 11.31 Division by a sparse polynomial

INPUT: Two polynomials $m(X)$ and $f(X)$ with coefficients in a commutative ring, where $m(X)$ is the sparse polynomial $X^d + \sum_{i=1}^{t} a_i X^{b_i}$ with $b_i < b_{i+1}$ and $b_1 = 0$.
OUTPUT: The polynomials u and v such that $f = um + v$ with $\deg v < d$.

1. $v \leftarrow f$ and $u \leftarrow 0$
2. **while** $\deg(v) \geqslant d$ **do**
3. $\quad k \leftarrow \max\{d, \deg v - d + b_t + 1\}$
4. $\quad$ write $v(X)$ as $u_1(X)X^k + w(X)$ $\qquad [\deg w < k]$
5. $\quad v(X) \leftarrow w(X) - u_1(x)\big(m(X) - X^d\big)X^{k-d}$

6. $u(X) \leftarrow u_1(X)X^{k-d} + u(X)$
7. **return** (u, v)

Remarks 11.32

(i) If $\deg f = d'$ then Algorithm 11.31 needs at most $2t(d' - d + 1)$ field additions to compute u and v such that $f = um + v$. If $d' \leqslant 2d - 2$, as is the case when performing arithmetic modulo m, then one needs $4(d - 1)$ additions for a reduction modulo a trinomial and $8(d-1)$ additions modulo a pentanomial. The number of loops is at most $\lceil (d' - d + 1)/(d - b_t - 1) \rceil$. Again if $d' \leqslant 2d - 2$, then the number of loops is at most equal to 2 whatever the value of b_t, as long as $1 \leqslant b_t \leqslant d/2$.

(ii) To avoid computing the quotient u when it is not required, simply discard Line 6 of Algorithm 11.31.

(iii) When the modulus is fixed, there is in general an even faster algorithm that exploits the form of the polynomial. This is the case for NIST irreducible polynomials [FIPS 186-2], cf. for example [HAMEˆ+ 2003, pp. 55–56]

Example 11.33 Take $m(X) = X^{11} + X^2 + 1$ and $f(X) = X^{20} + X^{16} + X^{15} + X^{12} + X^5 + X^3 + X + 1$, and let us find the quotient and remainder of the division of f by m with Algorithm 11.31.

- First $k = 12$, $u_1(X) = X^8 + X^4 + X^3 + 1$ and $w(X) = X^5 + X^3 + X + 1$.
- The new value of $v(X)$ is $X^{11} + X^9 + X^7 + X^6 + X^4 + 1$ and $u(X) = X^9 + X^5 + X^4 + X$.
- For the next and last loop, $k = 11$, $u_1(X) = 1$ and $w(X) = X^9 + X^7 + X^6 + X^4 + 1$.

Finally, $v(X) = X^9 + X^7 + X^6 + X^4 + X^2$ and $u(X) = X^9 + X^5 + X^4 + X + 1$ and one checks that $f(X) = u(X)m(X) + v(X)$.

Instead of trying to minimize the number of nonzero coefficients of the modulus, another option is to do arithmetic modulo a *sedimentary polynomial* [COP 1984, ODL 1985], that is, a polynomial of the form $X^d + h(X)$ irreducible over $\mathbb{F}_q$ such that the degree of $h(X)$ is minimal. For $q = 2$, it has been shown that for all $d \leqslant 600$ the degree of h is at most 11 [GOMC 1993]. Algorithm 11.31 can be slightly modified to perform reduction modulo a sedimentary polynomial. Namely, replace the statement $k \leftarrow \max\{d, \deg v - d + b_t + 1\}$ by $k \leftarrow \max\{d, \deg v - \deg h\}$.

Tests performed in [GANÖ 2005] indicate that sedimentary polynomials are slightly less efficient than trinomials or pentanomials.

11.2.1.b Redundant polynomial representation

For some extensions of even degree there is a better choice, namely *all one polynomials*. They are of the form

$$m(X) = X^d + X^{d-1} + \cdots + X + 1.$$

For $d > 1$, such a polynomial is irreducible if and only if $d+1$ is prime and 2 is a primitive element of $\mathbb{F}_{d+1}$. Now it is clear from the definition of $m(X)$ that $m(X)(X + 1) = X^{d+1} + 1$. Thus an element of $\mathbb{F}_{2^d}$ can be represented on the basis $(\alpha, \alpha^2, \ldots, \alpha^d)$ where α is a root of $m(X)$. In other words, an element of $\mathbb{F}_{2^d}$ is represented by a polynomial of degree at most d without constant coefficient, 1 being replaced by $X + X^2 + \cdots + X^d$. Alternatively, if the representation does not need to be unique, elements can directly be written on $(1, X, X^2, \ldots, X^d)$. In any case, reductions

Table 11.1 Irreducible trinomials and pentanomials over $\mathbb{F}_2$.

	2,1	3,1	4,1	5,2	6,1	7,1	8,4,3,1	9,1	10,3
11,2	12,3	13,4,3,1	14,5	15,1	16,5,3,1	17,3	18,3	19,5,2,1	20,3
21,2	22,1	23,5	24,4,3,1	25,3	26,4,3,1	27,5,2,1	28,1	29,2	30,1
31,3	32,7,3,2	33,10	34,7	35,2	36,9	37,6,4,1	38,6,5,1	39,4	40,5,4,3
41,3	42,7	43,6,4,3	44,5	45,4,3,1	46,1	47,5	48,5,3,2	49,9	50,4,3,2
51,6,3,1	52,3	53,6,2,1	54,9	55,7	56,7,4,2	57,4	58,19	59,7,4,2	60,1
61,5,2,1	62,29	63,1	64,4,3,1	65,18	66,3	67,5,2,1	68,9	69,6,5,2	70,5,3,1
71,6	72,10,9,3	73,25	74,35	75,6,3,1	76,21	77,6,5,2	78,6,5,3	79,9	80,9,4,2
81,4	82,8,3,1	83,7,4,2	84,5	85,8,2,1	86,21	87,13	88,7,6,2	89,38	90,27
91,8,5,1	92,21	93,2	94,21	95,11	96,10,9,6	97,6	98,11	99,6,3,1	100,15
101,7,6,1	102,29	103,9	104,4,3,1	105,4	106,15	107,9,7,4	108,17	109,5,4,2	110,33
111,10	112,5,4,3	113,9	114,5,3,2	115,8,7,5	116,4,2,1	117,5,2,1	118,33	119,8	120,4,3,1
121,18	122,6,2,1	123,2	124,19	125,7,6,5	126,21	127,1	128,7,2,1	129,5	130,3
131,8,3,2	132,17	133,9,8,2	134,57	135,11	136,5,3,2	137,21	138,8,7,1	139,8,5,3	140,15
141,10,4,1	142,21	143,5,3,2	144,7,4,2	145,52	146,71	147,14	148,27	149,10,9,7	150,53
151,3	152,6,3,2	153,1	154,15	155,62	156,9	157,6,5,2	158,8,6,5	159,31	160,5,3,2
161,18	162,27	163,7,6,3	164,10,8,7	165,9,8,3	166,37	167,6	168,15,3,2	169,34	170,11
171,6,5,2	172,1	173,8,5,2	174,13	175,6	176,11,3,2	177,8	178,31	179,4,2,1	180,3
181,7,6,1	182,81	183,56	184,9,8,7	185,24	186,11	187,7,6,5	188,6,5,2	189,6,5,2	190,8,7,6
191,9	192,7,2,1	193,15	194,87	195,8,3,2	196,3	197,9,4,2	198,9	199,34	200,5,3,2
201,14	202,55	203,8,7,1	204,27	205,9,5,2	206,10,9,5	207,43	208,9,3,1	209,6	210,7
211,11,10,8	212,105	213,6,5,2	214,73	215,23	216,7,3,1	217,45	218,11	219,8,4,1	220,7
221,8,6,2	222,5,4,2	223,33	224,9,8,3	225,32	226,10,7,3	227,10,9,4	228,113	229,10,4,1	230,8,7,6
231,26	232,9,4,2	233,74	234,31	235,9,6,1	236,5	237,7,4,1	238,73	239,36	240,8,5,3
241,70	242,95	243,8,5,1	244,111	245,6,4,1	246,11,2,1	247,82	248,15,14,10	249,35	250,103
251,7,4,2	252,15	253,46	254,7,2,1	255,52	256,10,5,2	257,12	258,71	259,10,6,2	260,15
261,7,6,4	262,9,8,4	263,93	264,9,6,2	265,42	266,47	267,8,6,3	268,25	269,7,6,1	270,53
271,58	272,9,3,2	273,23	274,67	275,11,10,9	276,63	277,12,6,3	278,5	279,5	280,9,5,2
281,93	282,35	283,12,7,5	284,53	285,10,7,5	286,69	287,71	288,11,10,1	289,21	290,5,3,2
291,12,11,5	292,37	293,11,6,1	294,33	295,48	296,7,3,2	297,5	298,11,8,4	299,11,6,4	300,5
301,9,5,2	302,41	303,1	304,11,2,1	305,102	306,7,3,1	307,8,4,2	308,15	309,10,6,4	310,93
311,7,5,3	312,9,7,4	313,79	314,15	315,10,9,1	316,63	317,7,4,2	318,45	319,36	320,4,3,1
321,31	322,67	323,10,3,1	324,51	325,10,5,2	326,10,3,1	327,34	328,8,3,1	329,50	330,99
331,10,6,2	332,89	333,2	334,5,2,1	335,10,7,2	336,7,4,1	337,55	338,4,3,1	339,16,10,7	340,45
341,10,8,6	342,125	343,75	344,7,2,1	345,22	346,63	347,11,10,3	348,103	349,6,5,2	350,53
351,34	352,13,11,6	353,69	354,99	355,6,5,1	356,10,9,7	357,11,10,2	358,57	359,68	360,5,3,2
361,7,4,1	362,63	363,8,5,3	364,9	365,9,6,5	366,29	367,21	368,7,3,2	369,91	370,139
371,8,3,2	372,111	373,8,7,2	374,8,6,5	375,16	376,8,7,5	377,41	378,43	379,10,8,5	380,47
381,5,2,1	382,81	383,90	384,12,3,2	385,6	386,83	387,8,7,1	388,159	389,10,9,5	390,9
391,28	392,13,10,6	393,7	394,135	395,11,6,5	396,25	397,12,7,6	398,7,6,2	399,26	400,5,3,2
401,152	402,171	403,9,8,5	404,65	405,13,8,2	406,141	407,71	408,5,3,2	409,87	410,10,4,3
411,12,10,3	412,147	413,10,7,6	414,13	415,102	416,9,5,2	417,107	418,199	419,15,5,4	420,7
421,5,4,2	422,149	423,25	424,9,7,2	425,12	426,63	427,11,6,5	428,105	429,10,8,7	430,14,6,1
431,120	432,13,4,3	433,33	434,12,11,5	435,12,9,5	436,165	437,6,2,1	438,65	439,49	440,4,3,1
441,7	442,7,5,2	443,10,6,1	444,81	445,7,6,4	446,105	447,73	448,11,6,4	449,134	450,47
451,16,10,1	452,6,5,4	453,15,6,4	454,8,6,1	455,38	456,18,9,6	457,16	458,203	459,12,5,2	460,19
461,7,6,1	462,73	463,93	464,19,18,13	465,31	466,14,11,6	467,11,6,1	468,27	469,9,5,2	470,9
471,1	472,11,3,2	473,200	474,191	475,9,8,4	476,9	477,16,15,7	478,121	479,104	480,15,9,6
481,138	482,9,6,5	483,9,6,4	484,105	485,17,16,6	486,81	487,94	488,4,3,1	489,83	490,219
491,11,6,3	492,7	493,10,5,3	494,17	495,76	496,16,5,2	497,78	498,155	499,11,6,5	500,27

are made modulo $X^{d+1}+1$ and a squaring is simply a permutation of the coordinates. This idea, first proposed in [ITTS 1989] and rediscovered in [SIL 1999], is known as the *anomalous basis* or the *ghost bit basis* technique.

When $d>1$ is odd, one can always embed $\mathbb{F}_{2^d}$ into some cyclotomic ring $\mathbb{F}_2[X]/(X^n+1)$ but only for some $n \geqslant 2d+1$. So the benefits obtained from a cheap reduction are partially offset by a more expensive multiplication [WUHA+ 2002]. For elliptic and hyperelliptic curve cryptography only extensions of prime degree are relevant, cf. Chapter 22, so the best we can hope for with this idea is $n=2d+1$.

Adopting the idea of using sparse reducible polynomials with an appropriate irreducible factor, one can use reducible trinomials in case only an irreducible pentanomial exists for some degree d. First, we have to find a trinomial $T(X)=X^n+X^k+1$ with n slightly bigger than d and such that $T(X)$ admits an irreducible factor $m(X)$ of degree d. Such a trinomial is called a *redundant trinomial* and the idea is then to embed $\mathbb{F}_{2^d} \sim \mathbb{F}_2[X]/\big(m(X)\big)$ into $\mathbb{F}_{2^d} \sim \mathbb{F}_2[X]/\big(T(X)\big)$. In the range $[2, 10000]$, there is no irreducible trinomial in about 50% of the cases (precisely 4853 out of 9999 [SER 1998]) but an exhaustive search has shown that there are redundant trinomials for all the corresponding degrees, see [DOCHE] for a table. In general $n-d$ is small and in more than 85% of the cases the number of 32-bit words required to represent an element of $\mathbb{F}_{2^d}$ are the same with a redundant trinomial of degree n and with an irreducible pentanomial of degree d. This implies also that the multiplication has the same cost with both representations, since this operation is usually performed at a word level, cf. Section 11.2.2.a.

From a practical point of view an element of $\mathbb{F}_{2^d}$ is represented by a polynomial of degree less than n and the computations are done modulo $T(X)$. At the end of the whole computation, one can reduce modulo $m(X)$ and this can be done with only $T(X)$ and $\delta(X)=T(X)/m(X)$, since for any polynomial $f(X)$ one has

$$f(X) \bmod m(X) = \frac{f(X)\delta(X) \bmod T(X)}{\delta(X)},$$

as in (11.1). Redundant trinomials can speed up an exponentiation by a factor up to 30%, when compared to irreducible pentanomials, cf. [DOC 2005].

Note that this concept is in fact similar to *almost irreducible trinomials* introduced by Brent and Zimmermann in the context of random number generators in [BRZI 2003]. Similar ideas were also explored by Blake et al. [BLGA+ 1994a, BLGA+ 1996], and Tromp et al. [TRZH+ 1997].

11.2.1.c Normal and optimal normal bases

Another popular way to represent an element of $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$ is to use a normal basis. This is especially true when $q=2$, since in this case the squaring of an element is just a cyclic shift of its coordinates. However, multiplications are more complicated. As a result only special normal bases, called *optimal normal bases*, ONB for short, are used in practice; see Section 11.2.2.b.

Gauß periods of type $(n,1)$ and $(n,2)$, generate optimal normal bases (cf. Section 2.3.3.b and [MUON+ 1989]), and it has been proved that all the optimal normal bases can be produced by this construction [GALE 1992].

For $q=2$, this occurs

1. when $d+1$ is prime and 2 is a primitive element of $\mathbb{F}_{d+1}$. Then the nontrivial $(d+1)$-th roots of unity form an optimal normal basis of $\mathbb{F}_{2^d}$, called a *Type I ONB*.
2. when $2d+1$ is prime and either
   - 2 is primitive in $\mathbb{F}_{2d+1}$ or
   - 2 generates the quadratic residues in $\mathbb{F}_{2d+1}$, that is $2d+1 \equiv 3 \pmod 4$ and the order of 2 in $\mathbb{F}_{2d+1}$ is d.

Then there is a primitive $(2d+1)$-th root of unity ζ in $\mathbb{F}_{2^d}$ and $\zeta+\zeta^{-1}$ is a normal element generating a *Type II ONB*. Such a basis can be written $(\zeta+\zeta^{-1}, \zeta^2+\zeta^{-2}, \ldots, \zeta^d+\zeta^{-d})$ as shown in [BLRO+ 1998].

Note that Type I ONB and anomalous bases are equal up to suitable permutations. So it is possible to enjoy a cheap multiplication and a cheap squaring at the same time. However, as said previously, there is no Type I ONB for an extension of prime degree. The situation is slightly better for Type II ONB. Indeed, in the range $[50, 500]$ there are 80 extension degrees that are prime and among them only 18 have a Type II ONB, namely $53, 83, 89, 113, 131, 173, 179, 191, 233, 239, 251, 281, 293, 359, 419, 431, 443$, and 491. As a consequence, the use of optimal normal bases for cryptographic purposes is quite constrained in practice.

In the remainder of this section one details the arithmetic itself. First it is clear that addition and subtraction are the same operations in a field of characteristic two. Using polynomial representation or a normal basis one sees that an addition in $\mathbb{F}_{q^d}$ can be carried out with at most d additions in $\mathbb{F}_q$. Ultimately, an addition in $\mathbb{F}_{q^d}$ reduces to a bitwise-XOR hardware operation, which can be performed at a word level. Multiplications are also processed using a word-by-word approach.

11.2.2 Multiplication

Again this part mainly deals with software oriented solutions. For a discussion focused on hardware, see Chapter 26.

Montgomery representation for prime fields (see Section 10.4.2) can be easily generalized to extension fields of characteristic 2; see for instance [KOAC 1998]. We shall not investigate this option further but limit ourselves to multiplications using a polynomial basis and a normal basis.

11.2.2.a Polynomial basis

The internal representation of a polynomial is similar to multiprecision integers. Indeed, let ℓ be the word size used by the processor. Then a polynomial $u(X)$ of degree less than d will be represented as the r-word vector $(u_{r-1} \ldots u_0)$ and the j-th bit of the word u_i, that is the coefficient of $u(X)$ of degree $i\ell + j$, will be denoted by $u_i[j]$. Many operations on polynomials are strongly related to integer multiprecision arithmetic. For example, polynomials can be multiplied with a slightly modified version of Algorithm 10.8. However in general, we do not have the equivalent of single precision operations. For example, on computers there is usually no hardware multiplication of polynomials in $\mathbb{F}_2[X]$ of bounded degree, even if this operation is simpler than integer multiplication, since there is no carry to handle. Nevertheless, it is possible to perform computations at a word level doing XOR and shifts. Indeed, if $v(X)X^j$ has been already computed then it is easy to deduce $v(X)X^{i\ell+j}$. This is the principle of Algorithm 11.34 introduced in [LÓDA 2000a].

Algorithm 11.34 Multiplication of polynomials in $\mathbb{F}_2[X]$

INPUT: The polynomials $u(X), v(X) \in \mathbb{F}_2[X]$ of degree at most $d-1$ represented as words of size ℓ bits.

OUTPUT: The product $w(X) = u(X)v(X)$ of degree at most $2d-2$.

1. $w(X) \leftarrow 0$ and $r \leftarrow \lceil \deg u/\ell \rceil$
2. **for** $j = 0$ **to** $\ell - 1$ **do**
3. **for** $i = 0$ **to** $r - 1$
4. **if** $u_i[j] = 1$ **then** $w(X) \leftarrow w(X) + v(X)X^{i\ell}$

5. **if** $j \neq \ell - 1$ **then** $v(X) \leftarrow v(X)X$
6. **return** w

Remark 11.35 Algorithm 11.34 proceeds the bits of the word u_i from the right to the left. A left-to-right version exists as well, but it is reported to be a bit less efficient [HAME+ 2003].

Example 11.36 Let $u(X) = X^5+X^4+X^2+X$, $v(X) = X^{10}+X^9+X^7+X^6+X^5+X^4+X^3+1$ and $\ell = 4$. So $u = (0011\ 0110)$, $v = (0110\ 1111\ 1001)$ and $r = 2$. Here are the values of $v(X)$ and $w(X)$ at the end of Line 5 when Algorithm 11.34 executes.

j	0	1	2	3
v	(1101 1111 0010)	(0001 1011 1110 0100)	(0011 0111 1100 1000)	(0110 1111 1001 0000)
w	(0110 1111 1001 0000)	(1011 1101 0100 0010)	(1010 0110 1010 0110)	(1010 0110 1010 0110)

Finally $w(X) = X^{15} + X^{13} + X^{10} + X^9 + X^7 + X^5 + X^2 + X$.

Just as for exponentiation algorithms, precomputations and windowing techniques can be very helpful. The next algorithm scans k bits at a time from left to right and accesses intermediate products by table lookup. Usually a good compromise between the speedup and the number of precomputations is to take $k = 4$.

Algorithm 11.37 Multiplication of polynomials in $\mathbb{F}_2[X]$ using window technique

INPUT: The polynomials $u(X), v(X) \in \mathbb{F}_2[X]$ of degree at most $d-1$ represented as words of size ℓ bits. The precomputed products $t(X)v(X)$ for all $t(X)$ of degree less than k.
OUTPUT: The product $w(X) = u(X)v(X)$ of degree at most $2d-2$.

1. $w(X) \leftarrow 0$ and $r \leftarrow \lceil \deg u/\ell \rceil$
2. **for** $j = \ell/k - 1$ **down to** 0 **do**
3. **for** $i = 0$ **to** $r - 1$
4. $t(X) \leftarrow t_{k-1}X^{k-1} + \cdots + t_0$ where $t_m = u_i[jk + m]$
5. $w(X) \leftarrow w(X) + t(X)v(X)X^{i\ell}$ $[t(X)v(X)$ is precomputed$]$
6. **if** $j \neq 0$ **then** $w(X) \leftarrow w(X)X^k$
7. **return** w

Remark 11.38 As for prime fields, cf. Algorithm 11.1, it is possible to modify Algorithm 11.37 and interleave polynomial reductions with elementary multiplications in order to get the result in $\mathbb{F}_{2^d}$ directly at the end.

Example 11.39 To illustrate the way Algorithm 11.37 works, let us take $k = 2$, $u = (0011\ 0110)$, $v = (0110\ 1111\ 1001)$ and $\ell = 4$ as for Example 11.36. The successive values of t come from the bits of u in the following way $(0011\ \underline{01}10)$, $(\underline{00}11\ 0110)$, $(0011\ 01\underline{10})$, and $(00\underline{11}\ 0110)$.

j	1	1	0	0
i	0	1	0	1
t	(01)	(00)	(10)	(11)
w	(0110 1111 1001)	(0110 1111 1001)	(0001 0110 0001 0110)	(1010 0110 1010 0110)

The result is of course the same, i.e., $w(X) = X^{15} + X^{13} + X^{10} + X^9 + X^7 + X^5 + X^2 + X$.

Another idea is to emulate single precision multiplications by storing all the elementary products. However, for 32-bit words the number of precomputed values is far too big. That is why an intermediate approach involving Karatsuba method is often considered instead. In this case, the product of two single precision polynomials of degree less than 32 is computed with 9 multiplications of 8-bit blocks, each elementary product being obtained by table lookup [GAGE 1996].

Karatsuba method can also be applied to perform the whole product directly. In [GANÖ 2005] the crossover degree between *à la* schoolbook and Karatsuba multiplications is reported to be equal to 576. Other more sophisticated techniques like the FFT or Cantor multiplication based on evaluation/interpolation are useful only for even larger degrees. For example, the crossover between Karatsuba and Cantor multiplication is for degree 35840 [GANÖ 2005].

11.2.2.b Optimal normal bases

Unlike additions, multiplications are rather involved with normal bases. The standard way to multiply two elements in $\mathbb{F}_{q^d}$ within a normal basis is to introduce the so-called *multiplication matrix* $T_{\mathcal{N}}$ whose entries $t_{i,h}$ satisfy

$$\alpha^{q^i} \times \alpha = \sum_{h=0}^{d-1} t_{i,h} \alpha^{q^h} \quad \text{so that} \quad \alpha^{q^i} \times \alpha^{q^j} = \sum_{h=0}^{d-1} t_{i-j,h-j} \alpha^{q^h}.$$

So if $u = (u_0, \ldots, u_{d-1})$ and $v = (v_0, \ldots, v_{d-1})$ then the general term w_h of $w = uv$ is

$$w_h = \sum_{0 \leqslant i,j < d} u_i v_j t_{i-j,h-j}.$$

Example 11.40 The following is taken directly from [OMMA 1986]. Let α be a zero of $m(X) = X^7 + X^6 + 1$. The next equalities are computed $\bmod\ m(X)$.

$$\begin{aligned} \alpha &= X & \alpha^2 &= X^2 \\ \alpha^{2^2} &= X^4 & \alpha^{2^3} &= X^6 + X + 1 \\ \alpha^{2^4} &= X^6 + X^5 + X^4 + X^3 + X & \alpha^{2^5} &= X^5 + X^4 + X^2 + X + 1 \\ \alpha^{2^6} &= X^4 + X^3 + 1 & \alpha^{2^7} &= \alpha. \end{aligned}$$

The products $\alpha^{q^i} \times \alpha$ are

$$\begin{aligned} \alpha \times \alpha &= X^2 & \alpha^2 \times \alpha &= X^3 \\ \alpha^{2^2} \times \alpha &= X^5 & \alpha^{2^3} \times \alpha &= X^6 + X^2 + X + 1 \\ \alpha^{2^4} \times \alpha &= X^5 + X^4 + X^2 + 1 & \alpha^{2^5} \times \alpha &= X^6 + X^5 + X^3 + X^2 + X \\ \alpha^{2^6} \times \alpha &= X^5 + X^4 + X. \end{aligned}$$

From this and in order to obtain $T_{\mathcal{N}}$, one introduces the matrix

$$\mathcal{M} = \left[\begin{array}{ccccccc|ccccccc} 0&0&0&1&0&1&1&0&0&0&1&1&0&0\\ 1&0&0&1&1&1&0&0&0&0&1&0&1&1\\ 0&1&0&0&0&1&0&1&0&0&1&1&1&0\\ 0&0&0&0&1&0&1&0&1&0&0&0&1&0\\ 0&0&1&0&1&1&1&0&0&0&0&1&0&1\\ 0&0&0&0&1&1&0&0&0&1&0&1&1&1\\ 0&0&0&1&1&0&0&0&0&0&1&0&1&0 \end{array}\right]$$

where the first and last seven columns give respectively the expression of α^{q^i} and of $\alpha^{q^i} \times \alpha$ on the basis $1, X, \ldots, X^6$. To get the identity matrix in the left part of $\mathcal{M}$ one performs a Gaussian elimination, which gives at the same time the transposed matrix of $T_{\mathcal{N}}$ in the right part of $\mathcal{M}$. Hence

$$T_{\mathcal{N}} = \left[\begin{array}{ccccccc} 0&1&0&0&0&0&0\\ 1&1&0&1&1&1&0\\ 0&0&0&1&1&0&1\\ 0&1&0&1&0&0&0\\ 1&0&0&0&0&1&0\\ 0&1&1&0&1&0&0\\ 1&0&1&1&1&0&1 \end{array}\right].$$

From this matrix, one deduces for instance that $\alpha^2 \times \alpha = \alpha + \alpha^2 + \alpha^{2^3} + \alpha^{2^4} + \alpha^{2^5}$.

The number of nonzero coefficients of the matrix $T_{\mathcal{N}}$ is denoted by $\delta_{\mathcal{N}}$ and called the *density of* $T_{\mathcal{N}}$. It is a crucial parameter for the speed of the system since the multiplication of two elements in $\mathbb{F}_{q^d}$ can be computed with at most $2d\,\delta_{\mathcal{N}}$ multiplications and $d(\delta_{\mathcal{N}} - 1)$ additions in $\mathbb{F}_q$. On average the density is about $(q-1)d^2/q$ [BEGE$^+$ 1991] but in fact $\delta_{\mathcal{N}} \geqslant 2d - 1$ [MUON$^+$ 1989] and this bound is sharp. By definition, an optimal normal basis, cf. Section 11.2.1.c, has such a minimal density.

Concerning $\mathbb{F}_{2^d}$, recall that a multiplication in a Type I ONB can be in fact performed in the corresponding anomalous basis. There is a simple way to transform an element into a polynomial and computations are made modulo $X^{d+1} - 1$. For Type II ONB, there is a similar idea called *palindromic representation* [BLRO$^+$ 1998]. The situation is not as favorable as for Type I ONB since in this case computations must be made modulo $X^{2d+1} - 1$. Optimal normal bases of Type I and II appear as special cases of Gauß periods, cf. Section 2.3.3.b.

11.2.3 Squaring

Squaring is a trivial operation for extensions of $\mathbb{F}_2$ in normal basis representation and it is very simple in polynomial representation. The absolute Frobenius $X \mapsto X^2$ being a linear map, one sees that if $u(X) = \sum u_i X^i$ then $u^2(X) = \sum u_i X^{2i}$. Thus, this operation is nothing but inserting 0 bits in the internal representation of u and reducing the result modulo $m(X)$. Precomputing a table of 256 values containing the squares of each byte allows us to speed up the 0-bit insertion process. However, the reduction remains the most time-consuming part of the whole process.

To speed up this process a bit, it is possible to split the square $\sum u_i X^{2i}$ into an even and an odd part so that the number of required bitwise-XOR operations to actually perform the reduction is halved. See [KIN 2001] for details.

11.2.4 Inversion and division

There are mainly two ways to compute the inverse of an element $\alpha \in \mathbb{F}_{q^d}$. The first method is to perform an extended gcd computation of the polynomial representing α and the irreducible polynomial defining $\mathbb{F}_{q^d}$. Alternatively, one can exploit the multiplicative structure of the group $\mathbb{F}_{q^d}^*$ with Lagrange's theorem. This is especially useful for normal bases.

11.2.4.a Euclid extended gcd

Given two nonzero polynomials f and m in $\mathbb{F}_q[X]$, there are unique polynomials u, v, and g such that $fu + mv = g$ where $g = \gcd(f, m)$, $\deg u < \deg m$, and $\deg v < \deg f$. In case m is irreducible and $\deg f < \deg m$ we have $g = 1$ so that u is the inverse of f modulo m. The following algorithm returns u, v and g.

Algorithm 11.41 Euclid extended polynomial gcd

INPUT: Two nonzero polynomials $f, m \in \mathbb{F}_q[X]$.
OUTPUT: The polynomials u, v, g in $\mathbb{F}_q[X]$ such that $fu + mv = g$ with $g = \gcd(f, m)$.

1. $u \leftarrow 1, v \leftarrow 0, s \leftarrow m$ and $g \leftarrow f$
2. **while** $s \neq 0$ **do**
3. compute Euclid division of g by s $[g = qs + r]$
4. $t \leftarrow u - vq, u \leftarrow v, g \leftarrow s, v \leftarrow t$ and $s \leftarrow r$
5. $v \leftarrow (g - fu)/m$
6. **return** (u, v, g)

Remark 11.42 Assuming $\deg f \leqslant d$ and $\deg m \leqslant d$, Algorithm 11.41 requires $O(d^2)$ elementary operations in $\mathbb{F}_q$.

Example 11.43 Take $m(X) = X^{11} + X^2 + 1$ and $f(X) = X^8 + X^6 + X^5 + X^4 + X + 1$. Algorithm 11.41 proceeds as follows

q	r	u	v	g
(0000)	(0001 0111 0011)	(0000)	(0001)	(1000 0000 0101)
(1011)	(1000)	(1011)	(0001 0111 0011)	(0001 0111 0011)
(0010 1110)	(0011)	(1011)	(0001 0000 0011)	(1000)
(0111)	(0001)	(0001 0000 0011)	(0111 0000 0010)	(0011)
(0011)	(0000)	(0111 0000 0010)	(1000 0000 0101)	(0001)
—	—	(0111 0000 0010)	(1100 1011)	(0001)

One deduces that $(X^{10} + X^9 + X^8 + X)f(X) + (X^7 + X^6 + X^3 + X + 1)m(X) = 1$ which implies that $X^{10} + X^9 + X^8 + X$ is the inverse of $f(X)$ in $\mathbb{F}_2[X]/\big(m(X)\big)$.

11.2.4.b Binary inversion

For extensions of $\mathbb{F}_2$, there is also a dedicated algorithm inspired by the binary integer version [BrCu+ 1993].

Algorithm 11.44 Inverse of an element of $\mathbb{F}_{2^d}^*$ in polynomial representation

INPUT: An irreducible polynomial $m(X) \in \mathbb{F}_2[X]$ of degree d and a nonzero polynomial $f(X) \in \mathbb{F}_2[X]$ such that $\deg f < d$.
OUTPUT: The polynomial $u(X) \in \mathbb{F}_2[X]$ such that $fu \equiv 1 \pmod m$.

1. $u \leftarrow 1, v \leftarrow 0, s \leftarrow m$ and $\delta \leftarrow 0$
2. **for** $i = 1$ **to** $2d$ **do**
3. **if** $f_d = 0$ **then** $\big[f(X) = f_d X^d + \cdots + f_0\big]$
4. $f(X) \leftarrow Xf(X), u(X) \leftarrow \big(Xu(X)\big) \bmod m(X)$ and $\delta \leftarrow \delta + 1$
5. **else**
6. **if** $s_d = 1$ **then** $\big[s(X) = s_d X^d + \cdots + s_0\big]$
7. $s(X) \leftarrow s(X) - f(X)$ and $v(X) \leftarrow \big(v(X) - u(X)\big) \bmod m(X)$
8. $s(X) \leftarrow Xs(X)$
9. **if** $\delta = 0$ **then**
10. $t(X) \leftarrow f(X), f(X) \leftarrow s(X)$ and $s(X) \leftarrow t(X)$
11. $t(X) \leftarrow u(X), u(X) \leftarrow v(X)$ and $v(X) \leftarrow t(X)$
12. $u(X) \leftarrow \big(Xu(X)\big) \bmod m(X)$
13. $\delta \leftarrow 1$
14. **else**
15. $u(X) \leftarrow \big(u(X)/X\big) \bmod m(X)$ and $\delta \leftarrow \delta - 1$
16. **return** u

Remarks 11.45

(i) The operations $\big(Xu(X)\big) \bmod m(X)$ and $\big(u(X)/X\big) \bmod m(X)$ can be very efficiently performed if $X^d \bmod m(X)$ and $X^{-1} \bmod m(X)$ are precomputed.

(ii) Algorithm 11.44, unlike other binary gcd versions (see [HaMe+ 2003] for instance) does not require any degree comparison thanks to the use of the counter δ. This idea was first suggested by Brent and Kung for modular inversion (see [BrKu 1983] and Section 11.1.3.a) and gives good performances in both software and hardware.

(iii) A similar algorithm testing least significant bits instead of most significant bits has been recently proposed [WuWu+ 2004].

(iv) It is possible to directly obtain $\big(h(X)/f(X)\big) \bmod m(X)$ by setting $u(X) \leftarrow h(X)$ instead of $u \leftarrow 1$ in the first line of Algorithms 11.41 and 11.44. In this case, a reduction is almost always needed at the end when the first algorithm is used, whereas the result is already reduced with the second one.

Example 11.46 With the values of Example 11.43, the successive steps of Algorithm 11.44 are

i	u	v	s	δ
1	(0010)	(0000)	(1000 0000 0101)	1
2	(0100)	(0000)	(1000 0000 0101)	2
3	(1000)	(0000)	(1000 0000 0101)	3
4	(0100)	(1000)	(1110 011 1010)	2
5	(0010)	(1000)	(1110 0111 0100)	1
6	(0001)	(1010)	(1011 1101 1000)	0
7	(0001 0110)	(0001)	(1011 1001 1000)	1
8	(0010 1100)	(0001)	(1011 1001 1000)	2
9	(0101 1000)	(0001)	(1011 1001 1000)	3
10	(1011 0000)	(0001)	(1011 1001 1000)	4
11	(0001 0110 0000)	(0001)	(1011 1001 1000)	5
12	(1011 0000)	(0001 0110 0001)	(0111 0011 0000)	4
13	(0101 1000)	(0001 0110 0001)	(1110 0110 0000)	3
14	(0010 1100)	(0001 0011 1001)	(1100 1100 0000)	2
15	(0001 0110)	(0001 0001 0101)	(1001 1000 0000)	1
16	(1011)	(0001 0000 0011)	(0011 0000 0000)	0
17	(0010 0000 0110)	(1011)	(1000 0000 0000)	1
18	(0100 0000 1100)	(1011)	(1000 0000 0000)	2
19	(0010 0000 0110)	(0100 0000 0111)	(1000 0000 0000)	1
20	(0001 0000 0011)	(0110 0000 0001)	(1000 0000 0000)	0
21	(0110 0000 0001)	(0001 0000 0011)	(1100 0000 0000)	1
22	(0111 0000 0010)	(0111 0000 0010)	(1000 0000 0000)	0

and the final result is the same, that is, the inverse of $f(X)$ is $X^{10} + X^9 + X^8 + X$.

11.2.4.c Inversion based on Lagrange's theorem

It is also possible to use the group structure of $\mathbb{F}_{q^d}^*$ to get the inverse of an element α. This method has the same asymptotic complexity as the extended Euclidean one but is reported to be a little faster [NÖC 1996] when a squaring is for free. We know that $|\mathbb{F}_{q^d}^*| = q^d - 1$ with q some power of 2, say $q = 2^k$. So $\alpha^{q^d-2} = 1/\alpha$. Now

$$q^d - 2 = (q^{d-1} - 1)q + q - 2,$$

and we can take advantage of the special expression of $q^{d-1}-1$ in base q and of the Frobenius, which makes the computation of q-th powers easier. For better performance, addition chains, presented in Section 9.2.3, are used as well.

Algorithm 11.47 Inverse of an element of $\mathbb{F}_{q^d}^*$ using Lagrange's theorem

INPUT: An element $\alpha \in \mathbb{F}_{q^d}^*$, two addition chains, namely $(a_0, a_1, \ldots, a_{s_1})$ for $q - 2$ and $(b_0, b_1, \ldots, b_{s_2})$ for $d - 1$.
OUTPUT: The inverse of α i.e., $\alpha^{q^d-2} = 1/\alpha$.

1. $y \leftarrow \alpha^{q-2}$ [using $(a_0, a_1, \ldots, a_{s_1})$ and Algorithm 9.41]

2. $T[0] \leftarrow \alpha \times y$ and $i \leftarrow 1$
3. **while** $i \leqslant s$ **do**
4. $\quad t \leftarrow T[k]^{q^j}$ where $b_i = b_k + b_j$
5. $\quad T[i] \leftarrow t \times T[j]$ $\qquad \left[T[i] = \alpha^{q^{b_i}-1} \text{ for all } i\right]$
6. $\quad i \leftarrow i + 1$
7. $t \leftarrow T[s_2]$ $\qquad [b_{s_2} = d - 1]$
8. **return** yt^q

Remarks 11.48

(i) Note that exchanging b_k and b_j, in Line 4, does not alter the correctness of the algorithm. In fact it is better to force b_k to be the maximum of b_k and b_j so that the exponentiation $T[b_k]^{q^{b_j}}$ is simpler.

(ii) One can obtain the inverse of $\alpha \in \mathbb{F}_{q^d}$ with $s_1 + s_2 + 2$ multiplications in $\mathbb{F}_{q^d}$ and $(1 + \sum_i b_j)$ q-th power computations where b_j is the integer in $b_i = b_k + b_j$. This last number is equal to $d-1$ when $(b_0, b_1, \ldots, b_{s_2})$ is a star addition chain, cf. Section 9.2.1.

(iii) One of the three methods proposed by Itoh and Tsujii [ITTS 1988] is a special case of Algorithm 11.47 when $q = 2$ and the addition chain computing $d - 1$ is derived from the square and multiply method.

(iv) When q is bigger than 2, another option suggested by Itoh and Tsujii is to write α^{-1} as $\alpha^{-r} \times \alpha^{r-1}$ where $r = (q^d - 1)/(q - 1) = q^{d-1} + \cdots + q + 1$. As $\alpha^r \in \mathbb{F}_q$, it can be easily inverted. It is the standard way to compute an inverse in an OEF, cf. Section 11.3.

Example 11.49 Suppose that one wants the inverse of $\alpha \in \mathbb{F}_{2^{19}}$, that is $\alpha^{2^{19}-2}$. Obviously, one has $2^{19} - 2 = 2(2^{18} - 1)$ and an addition chain for 18 is $(1, 2, 3, 6, 12, 18)$.

i	$b_i = b_k + b_j$	$T[k]^{q^{b_j}} \times T[j]$	$T[i]$
0	1	—	α
1	$2 = 1 + 1$	$T[0]^{2^1} \times T[0]$	$\alpha^2 \times \alpha = \alpha^3$
2	$3 = 2 + 1$	$T[1]^{2^1} \times T[0]$	$\alpha^6 \times \alpha = \alpha^7$
3	$6 = 3 + 3$	$T[2]^{2^3} \times T[2]$	$\alpha^{7\times 8} \times \alpha^7 = \alpha^{63}$
4	$12 = 6 + 6$	$T[3]^{2^6} \times T[3]$	$\alpha^{63\times 64} \times \alpha^{64} = \alpha^{4095}$
5	$18 = 12 + 6$	$T[4]^{2^6} \times T[3]$	$\alpha^{4095\times 64} \times \alpha^{63} = \alpha^{2^{18}-1}$

Finally $T[5]^2 = \alpha^{-1}$.

11.2.5 Exponentiation

In polynomial representation, a simple trick can greatly speed up exponentiation. Namely, let $f(X)$, $m(X)$ be polynomials in $\mathbb{F}_q[X]$ and $g(X) = X^{q^r}$. Because of the Frobenius action, it is obvious that $f^{q^r} \equiv f(g) \pmod m$. At this point one uses a fast algorithm for modular composition designed by Brent and Kung [BRKU 1978].

The idea, *à la* baby-step giant-step, is to write

$$f(X) = \sum_{0 \leqslant i < k} X^{ki} F_i(X) \text{ with } k = \lceil \deg f \rceil \text{ and } F_i(X) = \sum_{0 \leqslant j < k} f_{ik+j} X^j$$

and to precompute and store $1, g, g^2, \ldots, g^{k-1}$ and $1, g^k, g^{2k}, \ldots, g^{k(k-1)}$ modulo m. Here is the complete algorithm:

Algorithm 11.50 Modular composition of Brent and Kung

INPUT: The polynomials $m, f, g \in \mathbb{F}_q[X]$ with $\deg m = d$ and $\deg f, g < d$.
OUTPUT: The polynomial $f(g) \bmod m$.

1. $k \leftarrow \lceil \sqrt{d} \rceil$
2. $G[0] \leftarrow 1$
3. **for** $i = 1$ **to** k **do** $G[i] \leftarrow (gG[i-1]) \bmod m$ $\quad [G[i] = g^i \bmod m]$
4. $P[0] \leftarrow 1$
5. **for** $i = 1$ **to** $k-1$ **do** $P[i] \leftarrow (G[k]P[i-1]) \bmod m$ $\quad [P[i] = g^{ki} \bmod m]$
6. **for** $i = 0$ **to** $k-1$ **do** $F[i] \leftarrow \sum_{j=0}^{k-1} f_{ik+j} G[j]$ $\quad [F[i] = F_i(g)]$
7. $R \leftarrow \left(\sum_{i=0}^{k-1} F[i]P[i]\right) \bmod m$
8. **return** R

Remark 11.51 With classical arithmetic the complexity of Algorithm 11.50 is $O(d^{5/2})$, but it can be reduced to $O(d^{1/2+\lg 3})$. Indeed, as shown in [NÖC 1996], the loop in Line 6 can be computed with fast matrix multiplication *à la* Strassen [KNU 1997] and the other multiplications with the Karatsuba method.

Example 11.52 Let $m(X) = X^{15} + X + 1$ irreducible over $\mathbb{F}_2$, $f(X) = X^{14} + X^{13} + X^8 + X^6 + X^4 + X^3 + 1$ and $g(X) = X^{10} + X^3 + 1$. One has $k = 4$ and

$$f(X) = F_0(X) + X^4 F_1(X) + X^8 F_2(X) + X^{12} F_3(X)$$

with

$$F_0(X) = X^3 + 1, F_1(X) = X^2 + 1, F_2(X) = 1 \text{ and } F_3(X) = X^2 + X.$$

The precomputed values g^i and g^{ki} for $0 \leqslant i \leqslant k$ are respectively stored in the arrays G and P whereas $F[i]$ contains $F_i(g)$.

i	$G[i]$	$P[i]$	$F[i]$
0	(0001)	(0001)	(0101 0100 1011)
1	(0100 0000 1001)	(0100 0000 0001)	(0010 0000)
2	(0010 0001)	(0110 0001)	(0001)
3	(0101 0010 1010)	(0100 0110 0100)	(0100 0100 1000)

Finally $R = X^{13} + X^{12} + X^{11} + X^9 + X^7 + X^5 + X^3 + X^2 + X + 1$ which is equivalent to $f(g)$ modulo m.

Now we present Shoup's algorithm [SHO 1994a, GAGA$^+$ 2000] which is mainly based on the q^r-ary method for a well chosen r.

Algorithm 11.53 Shoup exponentiation algorithm

INPUT: The polynomials $f, m \in \mathbb{F}_q[X]$ with $\deg m = d$ and $\deg f < d$. A parameter r and an exponent $n = (n_{\ell-1} \dots n_0)_{q^r}$ such that $0 < n < q^d$.
OUTPUT: The polynomial $f^n \bmod m$.

1. **for** $i = 0$ **to** $\ell - 1$ precompute and store $f^{n_i} \bmod m$
2. $g(X) \leftarrow X^{q^r} \bmod m$ and $y \leftarrow 1$
3. **for** $i = \ell - 1$ **down to** 0 **do**
4. $\quad y \leftarrow y(g)$ [use Algorithm 11.50]
5. $\quad y \leftarrow \left(y \times f^{n_i}\right) \bmod m$
6. **return** y

Remarks 11.54

(i) The parameter r is usually set to $\lceil d/\log_q d \rceil$ and the precomputations can be done with Yao's method, cf. Algorithm 9.44, as proposed by Gao et al. [GAGA$^+$ 2000].
(ii) Neglecting precomputations, the number of multiplications needed is $O(d/\lg d)$. Its complexity, including the cost of precomputations, is $O(d^3/\lg d + d^2 \lg d)$ with classical arithmetic and $O(d^{1+\lg 3}/\lg d + d^{(1+\lg 7)/2} \lg d)$ with Karatsuba method and *à la* Strassen matrix multiplication techniques for modular composition.
(iii) The number of stored values is $O(d/\lg d)$.
(iv) The for loop starting Line 3 is a *Horner-like scheme*.

Example 11.55 Take $q = 2, m(X) = X^{15}+X+1, f(X) = X^{14}+X^{13}+X^8+X^6+X^4+X^3+1$ and $n = 23801$. Let us compute $f^n \bmod m$ with Algorithm 11.53. One has $r = \lceil 15/\lg 15 \rceil = 4$, $23801 = (5\,12\,15\,9)_{16}$, and $g(X) \equiv X^2+X \pmod{m(X)}$. Then for each i, $y(g) \equiv y^{16} \pmod m$ and $\left(yf^{n_i}\right) \bmod m$ are successively computed. In the following table, we give the corresponding values of y after the execution of Lines 4 and 5 of Shoup's algorithm, as well as the precomputed values f^{n_i} used at each step.

i	y	$f^{n_i} \bmod m$
3	1	(0001 1001 0011 1000)
—	(0001 1001 0011 1000)	—
2	(0110 1001 0111 1000)	(1001 1101 1100)
—	(0010 1001 1001 0000)	—
1	(0111 0011 1101 0010)	(0111 0100 0000 0011)
—	(0100 0101 0111 1110)	—
0	(0011 1000 1101 1010)	(0100 0001 0111 0011)
—	(0001 0111 0001 0101)	—

11.2.6 Square roots and quadratic equations

Every element $\alpha \in \mathbb{F}_{2^d}$ is a square. The square root of α can be easily obtained thanks to the multiplicative structure of $\mathbb{F}_{2^d}^*$, which implies that $\sqrt{\alpha} = \alpha^{2^{d-1}}$. In a normal basis the computation of a square root is therefore immediate. If α is represented by $f(X) = \sum_{i=0}^{d-1} f_i X^i$ on a polynomial basis, it is better to write

$$\sqrt{f(X)} = \sum_{i \text{ even}} f_i X^{i/2} + \sqrt{X} \sum_{i \text{ odd}} f_i X^{\frac{i-1}{2}}$$

where $\sqrt{X}$ has been precomputed modulo $m(X)$. When $m(X)$ is the irreducible trinomial $X^d + X^k + 1$ with d odd, note that $\sqrt{X}$ can be obtained directly. Indeed

$$\sqrt{X} \equiv X^{\frac{d+1}{2}} + X^{\frac{k+1}{2}} \pmod{m(X)}$$

if k is odd and $\sqrt{X} \equiv X^{-\frac{d-1}{2}}(X^{\frac{k}{2}} + 1) \pmod{m(X)}$ otherwise. This technique applies to redundant trinomials as well; see Section 11.2.1.b.

Solving quadratic equations in $\mathbb{F}_{2^d}$ is not as straightforward as computing square roots. Indeed, let us solve the equation $T^2 + aT + b = 0$ in $\mathbb{F}_{2^d}$ where, by the above, a is assumed to be nonzero. The change of variable $T \leftarrow T/a$ yields the simpler equation

$$T^2 + T = c \text{ with } c = b/a^2. \tag{11.2}$$

Lemma 11.56 Equation (11.2) has a solution in $\mathbb{F}_{2^d}$ if and only if $\mathrm{Tr}(c) = 0$. If x is a solution then $x + 1$ is the other one.

When d is odd, such a solution is given by

$$x = \sum_{i=0}^{(d-3)/2} c^{2^{2i+1}}. \tag{11.3}$$

When d is even, set

$$x = \sum_{i=0}^{d-1} \left(\sum_{j=0}^{i} c^{2^j} \right) y^{2^i} \tag{11.4}$$

where $y \in \mathbb{F}_{2^d}$ is any element of trace 1.

Proof. Let x be a solution of (11.2). Then $\mathrm{Tr}(c) = \mathrm{Tr}(x^2 + x) = \mathrm{Tr}(x) + \mathrm{Tr}(x) = 0$. The opposite direction is proved by showing that the proposed solutions actually work. Computing $x^2 + x$, one has in the first case

$$x^2 + x = c + \mathrm{Tr}(c) = c,$$

and in the second one

$$x^2 + x = y\,\mathrm{Tr}(c) + c\,\mathrm{Tr}(y) = c.$$

Thus x is always a solution of (11.2) as claimed. □

In practice, several improvements can be considered. First, to check for the existence of a solution and then to actually compute such a solution.

Remarks 11.57

(i) There is a unique vector w in $\mathbb{F}_{2^d}$ that is orthogonal to all the elements of trace 0. If w is precomputed, it is enough to compute the scalar product $w \cdot c$ in order to deduce the trace of c [KNU 1999].

(ii) When the field $\mathbb{F}_{2^d}$ is defined by an irreducible polynomial of the form

$$X^d + a_{d-1}X^{d-1} + \cdots + a_1 X + a_0 \text{ with } a_j = 0 \text{ for all } j > d/2 \tag{11.5}$$

the trace of an element can also be obtained very efficiently.
Let θ be a root of this polynomial. Then using the Newton–Girard formula giving the sum of the conjugates of θ^k in terms of the a_i's and the linearity of the trace map it is immediate that

$$\text{if } c = \sum_{k=0}^{d-1} c_k \theta^k \text{ then } \mathrm{Tr}(c) = c_0 + \sum_{k=1}^{d-1} k c_k a_{d-k}.$$

As we have seen, moderately large extension fields of characteristic 2 can always be defined by trinomials or pentanomials of the form (11.5), so that the computation of the trace is always simple in practice.

Example 11.58 In $\mathbb{F}_{2^{233}}$ defined by $X^{233} + X^{74} + 1$ we have

$$\mathrm{Tr}\big(c_{232}\theta^{232} + c_{231}\theta^{231} + \cdots + c_1\theta + c_0\big) = c_0 + c_{159}.$$

Remark 11.59 Rather than computing a solution using (11.3) or (11.4), it can be faster to use the linearity of the map $\lambda \mapsto \lambda^2 + \lambda$ defined from $\mathbb{F}_{2^d}$ to $\mathbb{F}_{2^d}$. Indeed, precomputing the inverse matrix of this operator gives the result in a straightforward way. Additional tricks can be used to reduce the storage and the amount of computations [KNU 1999].

11.3 Optimal extension fields

On the one hand, multiplications in extension fields of characteristic 2 are usually performed less efficiently than in prime fields, due to the lack of a single precision polynomial multiplication on most processors. On the other hand, inversion in prime fields can be a very expensive operation, especially in hardware. To overcome these two difficulties, optimal extension fields have been recently investigated [MIH 1997, BAPA 1998]. They seem to be particularly interesting for smart cards [WOBA$^+$ 2000].

First, we shall briefly introduce optimal extension fields, and give existence criterions and some examples before addressing the arithmetic itself. We conclude with the special cases of extensions of degree 3 and 5.

11.3.1 Introduction

Let us take an extension field $\mathbb{F}_{p^d}$ such that

- the characteristic p fits in a machine word and allows a fast reduction in $\mathbb{F}_p$
- the irreducible polynomial defining $\mathbb{F}_{p^d}$ allows a fast polynomial reduction.

This choice leads to the following concept.

Definition 11.60 An *optimal extension field*, OEF for short, is an extension field $\mathbb{F}_{p^d}$ where

- p is a *pseudo-Mersenne prime*, that is $p = 2^n + c$ with $|c| \leqslant 2^{\lfloor n/2 \rfloor}$
- there is an irreducible binomial $m(X) = X^d - \omega$ over $\mathbb{F}_p$.

If $c = \pm 1$ then the field is said to be of *Type I* and it is of *Type II* when $\omega = 2$.

Remark 11.61 Generalizing Definition 11.60, cf. [AVMI 2004], it is possible to consider a prime p of another form provided a fast reduction algorithm exists; see Section 10.4.3 for examples.

The cardinality of $\mathbb{F}_{p^d}$ is approximately equal to 2^{nd} and in practice, an element $\alpha \in \mathbb{F}_{p^d}$ is represented by the polynomial $a_{d-1}X^{d-1} + \cdots + a_1X + a_0$ where $a_i \in \mathbb{F}_p$. As suggested before, this implies that computations in OEFs require two kinds of reduction. Intermediate results have to be reduced modulo the binomial $m(X)$, and for this task Algorithm 11.31 is not even required since a reduction modulo m consists simply of replacing X^d by ω. Coefficients of the polynomial also have to be reduced modulo p. For Type I OEF, this operation needs one addition in $\mathbb{F}_p$, cf. Section 10.4.3. Otherwise reduction is obtained by Algorithm 10.25 and is more expensive.

OEFs are rather easy to find and their search is simplified by the results below on the irreducibility of $X^d - \omega$ over $\mathbb{F}_p$.

Theorem 11.62 Let $d \geqslant 2$ be an integer and $\omega \in \mathbb{F}_p^*$. The binomial $m(X) = X^d - \omega$ is irreducible in $\mathbb{F}_p[X]$ if and only if the two following conditions hold

- each prime factor of d divides the order e of ω but does not divide $(p-1)/e$
- $p \equiv 1 \pmod 4$ if $d \equiv 0 \pmod 4$.

As shown in [JUN 1993] one has the sufficient condition

Corollary 11.63 If $\omega \in \mathbb{F}_p^*$ is a primitive element and $d \mid (p-1)$ then the polynomial $X^d - \omega$ is irreducible over $\mathbb{F}_p$.

If d is squarefree and $X^d - \omega$ irreducible over $\mathbb{F}_p$ then Theorem 11.62 implies that $p \equiv 1 \pmod d$. This remark is also useful to speed up the search of OEFs.

In Table 11.2 are given all OEFs of Type I, of cryptographic interest sorted with respect to nd.

Table 11.2 Type I OEFs.

n	c	d	ω	nd	n	c	d	ω	nd	n	c	d	ω	nd	n	c	d	ω	nd
13	−1	6	7	78	17	−1	5	3	85	13	−1	7	3	91	31	−1	3	5	93
7	−1	14	3	98	17	−1	6	3	102	19	−1	6	3	114	13	−1	9	7	117
7	−1	18	3	126	8	1	16	3	128	16	1	8	3	128	13	−1	10	3	130
19	−1	7	3	133	7	−1	21	3	147	17	−1	9	3	153	13	−1	13	2	169
17	−1	10	3	170	19	−1	9	3	171	13	−1	14	3	182	61	−1	3	5	183
31	−1	6	5	186	7	−1	27	3	189	13	−1	15	11	195	31	−1	7	3	217
13	−1	18	7	234	17	−1	15	3	255	8	1	32	3	256	16	1	16	3	256
19	−1	14	3	266	13	−1	21	7	273	31	−1	9	5	279	17	−1	17	2	289

Table 11.3 contains examples of OEFs of Type II. More precisely, given a size s between 135 and 300, for each $n \geqslant 7$ dividing s, the unique parameters $c \in \left[-2^{\lfloor n/2 \rfloor}, 2^{\lfloor n/2 \rfloor}\right]$ and d, if any, are given, such that

- $p = 2^n + c$ is prime
- c is minimal in absolute value
- $d = s/n$ and $X^d - 2$ is irreducible over $\mathbb{F}_p$.

Note that when $n = 8, 16, 32$, or 64 only negative values of c are reported so that elements of $\mathbb{F}_p$ can be represented with a single word on the corresponding commonly used architectures. Since these parameters are of great importance in practice, they appear distinctly in the table.

Concerning arithmetic, additions and subtractions are straightforward and do not enjoy special improvements, unlike other basic operations we shall describe now.

11.3.2 Multiplication

Let two elements $\alpha, \beta \in \mathbb{F}_{p^d}$ be represented by $\alpha = \sum_{i=0}^{d-1} a_i X^i$ and $\beta = \sum_{i=0}^{d-1} b_i X^i$ where a_i and b_i are in $\mathbb{F}_p$. Then using the relation $X^d \equiv \omega \pmod{m(X)}$, one has

$$\alpha\beta = c_{d-1} + \sum_{k=0}^{d-2} (c_k + \omega c_{d+k}) X^k \quad \text{with} \quad c_k = \sum_{i=0}^{k} a_i b_{k-i}.$$

Instead of reducing $a_i b_{k-i}$ modulo p at each step, it can be faster, especially for OEFs that are not of Type I, to compute $c_k + \omega c_{d+k}$ as a multiprecision integer and to reduce it only once. As shown in [HAME$^+$ 2003], if $p = 2^n + c$ is such that $\lg\big(1 + \omega(d-1)\big) + 2\lg|c| \leqslant n$, then $c_k + \omega c_{d+k}$ can be reduced at once with only two multiplications by c.

As suggested in [MIH 2000], one can also use convolutions methods, like the FFT, to multiply α and β. This is particularly effective when d is close to a power of 2, or close to the product of small primes.

As usual, a squaring should be considered independently and computed with a specific procedure.

11.3.3 Exponentiation

The action of the absolute Frobenius ϕ_p can be computed very efficiently in OEFs [MIH 2000]. Indeed, since the coefficients of $\alpha = \sum_{j=0}^{d-1} a_j X^j$ are in $\mathbb{F}_p$, one has

$$\alpha^{p^i} = \sum_{j=0}^{d-1} a_j \omega^{\lfloor jp^i/d \rfloor} X^{((jp^i) \bmod d)}.$$

Recall that when d is squarefree, $p \equiv 1 \pmod{d}$ so that $X^{((jp)^i \bmod d)}$ is simply X^j. Thus an exponentiation to the power p^i only requires us to multiply each coefficient a_j by some power of ω, which can be precomputed.

Another interesting choice is to take $p = kd + 1$ for a given d. In this case, $X^{((jp^i) \bmod d)} = X^j$ as well, and $\omega^{\lfloor jp/d \rfloor} = \zeta^j$ where $\zeta = \omega^{\frac{p-1}{d}} \in \mathbb{F}_p$ is a d-th root of unity.

Example 11.64 Let $p = 2^{16} - 165$, $\mathbb{F}_{p^6} \simeq \mathbb{F}_p[X]/(X^6 - 2)$ and take the random element

$$\alpha = 44048X^5 + 24430X^4 + 54937X^3 + 18304X^2 + 46713X + 63559.$$

One checks that $p - 1 \equiv 0 \pmod{6}$ so that $\zeta = 2^{\lfloor p/d \rfloor}$ is a 6-th primitive root of unity. Precomputing $\zeta, \zeta^2, \zeta^3, \zeta^4$ and ζ^5 modulo p, and multiplying a_j by ζ^j, one obtains

$$\alpha^p = 23814X^5 + 34492X^4 + 10602X^3 + 7340X^2 + 40911X + 63559.$$

Using the same set of precomputations, the product of α by the ζ^{4j}'s, componentwise gives

$$\alpha^{p^4} = 41725X^5 + 34492X^4 + 54937X^3 + 7340X^2 + 24628X + 63559.$$

Table 11.3 Examples of Type II OEFs.

n	c	d	nd	n	c	d	nd	n	c	d	nd	n	c	d	nd	n	c	d	nd	n	c	d	nd
15	−19	9	135	27	203	5	135	45	−55	3	135	17	29	8	136	34	85	4	136	23	11	6	138	46	−21	3	138
10	27	14	140	14	−3	10	140	20	−3	7	140	28	−95	5	140	35	53	4	140	47	5	3	141	11	−19	13	143
9	−3	16	144	12	−3	12	144	16	−15	9	144	18	−11	8	144	24	75	6	144	36	117	4	144	48	75	3	144
29	39	5	145	21	−21	7	147	49	−139	3	147	37	29	4	148	15	3	10	150	25	35	6	150	30	7	5	150
50	−51	3	150	19	−19	8	152	38	13	4	152	17	−13	9	153	51	65	3	153	14	−15	11	154	22	−57	7	154
31	413	5	155	12	−39	13	156	13	29	12	156	26	−45	6	156	39	−19	4	156	52	21	3	156	53	41	3	159
10	−3	16	160	16	−165	10	160	20	−3	8	160	32	−5	5	160	40	141	4	160	23	11	7	161	9	11	18	162
18	3	9	162	27	53	6	162	54	−33	3	162	41	−75	4	164	15	35	11	165	33	29	5	165	55	11	3	165
14	−3	12	168	21	−19	8	168	24	−63	7	168	28	3	6	168	42	−11	4	168	56	−57	3	168	13	−1	13	169
10	−3	17	170	17	−61	10	170	34	−113	5	170	19	−19	9	171	57	−13	3	171	43	29	4	172	29	−43	6	174
58	−63	3	174	7	3	25	175	25	41	7	175	35	53	5	175	11	5	16	176	22	−3	8	176	44	21	4	176
59	−55	3	177	12	15	15	180	15	−19	12	180	18	−93	10	180	20	−3	9	180	30	3	6	180	36	−5	5	180
45	−139	4	180	60	33	3	180	7	3	26	182	13	27	14	182	14	−3	13	182	26	−45	7	182	61	−31	3	183
23	−27	8	184	46	165	4	184	37	9	5	185	31	11	6	186	62	−57	3	186	17	−61	11	187	47	5	4	188
21	−19	9	189	27	203	7	189	63	−25	3	189	19	−27	10	190	38	7	5	190	12	−3	16	192	16	−243	12	192
24	−3	8	192	32	−387	6	192	48	21	4	192	64	−189	3	192	13	29	15	195	15	71	13	195	39	23	5	195
14	−3	14	196	28	−57	7	196	49	69	4	196	11	5	18	198	18	9	11	198	22	−3	9	198	33	29	6	198
10	−3	20	200	20	−5	10	200	25	69	8	200	40	15	5	200	50	−27	4	200	29	−3	7	203	17	29	12	204
34	−165	6	204	51	21	4	204	41	−21	5	205	23	11	9	207	13	29	16	208	16	−15	13	208	26	−27	8	208
52	21	4	208	11	5	19	209	19	−27	11	209	10	−15	21	210	14	−3	15	210	15	21	14	210	21	−21	10	210
30	7	7	210	35	53	6	210	42	−33	5	210	53	5	4	212	43	−67	5	215	8	−15	27	216	12	3	18	216
18	117	12	216	24	−33	9	216	27	29	8	216	36	117	6	216	54	−131	4	216	31	−85	7	217	20	33	11	220
22	67	10	220	44	55	5	220	55	−67	4	220	13	−31	17	221	17	−31	13	221	37	269	6	222	14	−3	16	224
16	−155	14	224	28	37	8	224	32	−17	7	224	56	−27	4	224	9	9	25	225	25	35	9	225	45	59	5	225
19	−19	12	228	38	−45	6	228	57	141	4	228	10	−11	23	230	23	−27	10	230	46	127	5	230	21	−111	11	231
33	35	7	231	29	−3	8	232	58	−27	4	232	13	−13	18	234	18	−11	13	234	26	15	9	234	39	−91	6	234
47	−127	5	235	59	−99	4	236	17	−115	14	238	34	−113	7	238	15	−19	16	240	16	−15	15	240	20	−3	12	240
24	75	10	240	30	−35	8	240	40	141	6	240	48	−165	5	240	60	−107	4	240	11	21	22	242	22	85	11	242
9	11	27	243	27	53	9	243	61	21	4	244	35	−31	7	245	49	69	5	245	41	−133	6	246	13	17	19	247
19	81	13	247	31	−19	8	248	62	−171	4	248	10	−3	25	250	25	−61	10	250	50	−113	5	250	12	63	21	252
14	−3	18	252	18	−35	14	252	21	−19	12	252	28	3	9	252	36	175	7	252	42	75	6	252	63	29	4	252
23	15	11	253	17	−61	15	255	51	−237	5	255	16	−99	16	256	32	−99	8	256	64	−59	4	256	43	−691	6	258
37	41	7	259	13	29	20	260	20	57	13	260	26	117	10	260	52	55	5	260	9	11	29	261	29	−43	9	261
11	5	24	264	12	−3	22	264	22	−3	12	264	24	73	11	264	33	29	8	264	44	21	6	264	53	−111	5	265
14	33	19	266	19	−85	14	266	38	−45	7	266	10	9	27	270	15	−19	18	270	18	87	15	270	27	203	10	270
30	3	9	270	45	−139	6	270	54	−33	5	270	16	−17	17	272	17	29	16	272	34	85	8	272	13	41	21	273
21	−69	13	273	39	−7	7	273	25	35	11	275	55	3	5	275	12	−47	23	276	23	29	12	276	46	−21	6	276
31	11	9	279	14	−3	20	280	20	−3	14	280	28	−125	10	280	35	53	8	280	40	27	7	280	56	175	5	280
47	5	6	282	15	−49	19	285	19	−67	15	285	57	−111	5	285	11	−19	26	286	22	−87	13	286	26	69	11	286
41	−31	7	287	9	−3	32	288	12	−3	24	288	16	−165	18	288	18	−11	16	288	24	117	12	288	32	−153	9	288
36	117	8	288	48	75	6	288	17	−1	17	289	29	149	10	290	58	−63	5	290	14	−3	21	294	21	−21	14	294
42	−161	7	294	49	−139	6	294	59	273	5	295	37	29	8	296	11	5	27	297	27	−39	11	297	33	17	9	297
23	293	13	299	12	−5	25	300	20	435	15	300	25	77	12	300	30	−83	10	300	50	−51	6	300	60	105	5	300

Accordingly, the use of the Frobenius speeds up an exponentiation to a generic power n. Indeed, a traditional approach would require $d \lg p$ squarings to get α^n, but writing n in basis p, i.e., $n = (n_{\ell-1} \dots n_0)_p$, it is clear that

$$\alpha^n = \prod_{i=0}^{\ell-1} \phi_p^i(\alpha^{n_i}).$$

Combined with a right-to-left strategy to compute the α^{n_i}'s, cf. Section 9.1.1, this idea shows that only $(\lg p - 1)$ squarings are needed, namely $\alpha^2, \alpha^4, \dots, \alpha^{2^{\lg p - 1}}$. In addition, each term α^{n_i} can be computed in parallel.

Example 11.65 Let $n = 270718518656895471173938628 89$ and let us compute α^n. First remark that $n = (22388\,12209\,20770\,63238\,8078\,10838)_p$. Using the precomputed values $\alpha^2, \alpha^4, \dots, \alpha^{2^{15}}$ Algorithm 9.2 gives

$$\begin{aligned}
\alpha^{n_0} &= 13812X^5 + 61164X^4 + 49159X^3 + 1927X^2 + 1781X + 31944 \\
\alpha^{n_1} &= 4807X^5 + 57203X^4 + 62178X^3 + 3283X^2 + 4690X + 33266 \\
\alpha^{n_2} &= 49155X^5 + 5527X^4 + 47396X^3 + 13274X^2 + 13828X + 60304 \\
\alpha^{n_3} &= 21607X^5 + 11848X^4 + 23310X^3 + 30303X^2 + 31752X + 44845 \\
\alpha^{n_4} &= 29730X^5 + 12285X^4 + 27469X^3 + 798X^2 + 9947X + 47295 \\
\alpha^{n_5} &= 54710X^5 + 18029X^4 + 18950X^3 + 23518X^2 + 10120X + 34955
\end{aligned}$$

and with the technique explained above one obtains

$$\begin{aligned}
\alpha^{n_0} &= 13812X^5 + 61164X^4 + 49159X^3 + 1927X^2 + 1781X + 31944 \\
\alpha^{pn_1} &= 60618X^5 + 51323X^4 + 3361X^3 + 4138X^2 + 57415X + 33266 \\
\alpha^{p^2n_2} &= 8288X^5 + 43479X^4 + 47396X^3 + 53960X^2 + 58194X + 60304 \\
\alpha^{p^3n_3} &= 43932X^5 + 11848X^4 + 42229X^3 + 30303X^2 + 33787X + 44845 \\
\alpha^{p^4n_4} &= 46496X^5 + 26171X^4 + 27469X^3 + 28535X^2 + 55771X + 47295 \\
\alpha^{p^5n_5} &= 37148X^5 + 9908X^4 + 46589X^3 + 49290X^2 + 55179X + 34955
\end{aligned}$$

so that the product of all these values is

$$\alpha^n = 42336X^5 + 42804X^4 + 21557X^3 + 49577X^2 + 22038X + 4278.$$

11.3.4 Inversion

Although an inverse could be computed with an extended gcd computation in $\mathbb{F}_{p^d}$, it is much faster to use the Frobenius action and an inversion in $\mathbb{F}_p$ to compute it.

Namely, take

$$r = \frac{p^d - 1}{p - 1} = p^{d-1} + p^{d-2} + \cdots + p + 1.$$

Then α^{r-1} and α^r are easily obtained using the Frobenius and in addition $\alpha^r \in \mathbb{F}_p$ since it is the norm of α. So α^r can be easily inverted to obtain

$$\alpha^{-1} = \alpha^{r-1} \times \alpha^{-r}.$$

Further improvements, reminiscent of an addition chain approach, can be applied to compute the term α^{r-1}. As an example, let us consider the extension degree $d = 6$, often used in practice with 32-bit architectures. In this case, the successive steps to compute α^{r-1} are

$$\alpha^p, \alpha^{p+1}, \alpha^{p^3+p^2}, \alpha^{p^5+p^4}, \alpha^{p^5+p^4+p^3+p^2} \text{ and } \alpha^{p^5+p^4+p^3+p^2+p}.$$

The entire algorithm is as follows.

Algorithm 11.66 OEF inversion

INPUT: A nonzero element $\alpha \in \mathbb{F}_{p^d}$.
OUTPUT: The inverse of α in $\mathbb{F}_{p^d}$.

1. $r \leftarrow (p^d - 1)/(p-1)$
2. $s \leftarrow \alpha^{r-1}$ [use an addition-chain-like approach]
3. $t \leftarrow s\alpha$ $[t = \alpha^r \in \mathbb{F}_p]$
4. $u \leftarrow t^{-1}$ [compute the inverse of t in $\mathbb{F}_p$]
5. **return** su

Remarks 11.67

(i) Algorithm 11.66 is in fact a generalization of a method proposed by Itoh and Tsujii [ITTS 1988] for characteristic 2 fields. See also Remark 11.48 (iv).

(ii) Since t belongs to $\mathbb{F}_p$ it is equal to the constant coefficient of the product $s\alpha$. Thus this multiplication needs only d multiplications in $\mathbb{F}_p$ as does the product su.

(iii) Let $\nu(k)$ be the Hamming weight of k, then α^{r-1} can be computed [HAMEˆ+ 2003] with $N_{\mathrm{M}} = \lfloor \lg d - 1 \rfloor + \nu(d-1) - 1$ products in $\mathbb{F}_{p^d}$ and at most N_{ϕ_p} Frobenius computations where

$$N_{\phi_p} = \begin{cases} N_{\mathrm{M}} + 1 & \text{if } d \text{ is odd,} \\ \lfloor \lg(d-1) \rfloor + \nu(d) & \text{otherwise.} \end{cases}$$

Example 11.68 Take p, $\mathbb{F}_{p^6}$ and α as defined in Example 11.64 and let us compute the inverse of α by Algorithm 11.66. First $r = p^5 + p^4 + p^3 + p^2 + p + 1$ and α^{r-1} is obtained by computing successively

$$\begin{aligned}
\alpha^{p} &= 23814X^5 + 34492X^4 + 10602X^3 + 7340X^2 + 40911X + 63559 \\
\alpha^{p+1} &= 27871X^5 + 42246X^4 + 20450X^3 + 8624X^2 + 26549X + 28414 \\
\alpha^{p^3+p^2} &= 47216X^5 + 11126X^4 + 20450X^3 + 50936X^2 + 29251X + 28414 \\
\alpha^{p^5+p^4} &= 55991X^5 + 12167X^4 + 20450X^3 + 5979X^2 + 9739X + 28414 \\
\alpha^{p^5+p^4+p^3+p^2} &= 26086X^5 + 2404X^4 + 35019X^3 + 45382X^2 + 45825X + 22132 \\
\alpha^{r-1} &= 28310X^5 + 14778X^4 + 7889X^3 + 29498X^2 + 2991X + 44851
\end{aligned}$$

with 3 multiplications in $\mathbb{F}_{p^6}$ and 3 applications of ϕ_p or ϕ_p^2. Finally, $t = \alpha^{r-1}\alpha \equiv 42318 \pmod p$, $u = t^{-1} \equiv 27541 \pmod p$ and

$$\alpha^{-1} = \alpha^{r-1}u = 33766X^5 + 3708X^4 + 9164X^3 + 48513X^2 + 58147X + 27858.$$

11.3.5 Squares and square roots

For any extension field $\mathbb{F}_{p^d}$ of odd characteristic, and not only for OEFs, there is a simple method relying on Theorem 2.104 and very similar to Algorithm 11.19 to decide if an element of $\mathbb{F}_{p^d}$ is a square or not.

Algorithm 11.69 Legendre–Kronecker–Jacobi symbol

INPUT: A polynomial $f(X) \in \mathbb{F}_p[X]$ and an irreducible polynomial $m(X) \in \mathbb{F}_p[X]$.
OUTPUT: The Legendre–Kronecker–Jacobi symbol $\left(\frac{f(X)}{m(X)}\right)$.

1. $k \leftarrow 1$
2. **repeat**
3. **if** $f(X) = 0$ **then return** 0
4. $a \leftarrow$ the leading coefficient of $f(X)$
5. $f(X) \leftarrow f(X)/a$
6. **if** $\deg m \equiv 1 \pmod 2$ **then** $k \leftarrow k\left(\frac{a}{p}\right)$
7. **if** $p^{\deg m} \equiv 3 \pmod 4$ **and** $\deg m \deg f \equiv 1 \pmod 2$ **then** $k \leftarrow -k$
8. $r(X) \leftarrow f(X)$, $f(X) \leftarrow m(X) \bmod r(X)$ and $m(X) \leftarrow r(X)$
9. **until** $\deg m = 0$
10. **return** k

Remark 11.70 Algorithm 11.69 relies on the law (2.7). Since $f(X)$ is not necessarily monic it is first divided by its leading coefficient a. Now we remark that $a \in \mathbb{F}_p$ is always a square in an extension of even degree. When the degree is odd a is a quadratic residue if and only if $a = 0$ or $\left(\frac{a}{p}\right) = 1$.

Example 11.71 Take $p = 7$, let $m(X)$ be the irreducible polynomial $X^9 + 2X^8 + X^7 + 2X^6 + 2X^5 + 4X^2 + 6X + 6$ and $f(X) = X^6 + X^5 + 6X^4 + 2X^3 + 2X^2 + 4X + 1$, both being elements of $\mathbb{F}_7[X]$.

After Line 8 the values of r, f, a and k are as follows

r	f	a	k
$X^6 + 4X^5 + 3X^4 + X^3 + X^2 + 2X + 4$	$4X^5 + 3X^4 + 4X^3 + 3X^2 + 2X + 4$	1	1
$X^5 + 6X^4 + X^3 + 6X^2 + 4X + 1$	$4X^3 + 2X^2 + 2X + 6$	4	1
$X^3 + 4X^2 + 4X + 5$	$2X^2 + 3X$	4	-1
$X^2 + 5X$	$2X + 5$	2	-1
$X + 6$	6	2	-1
1	0	6	1

So $f(X)$ is a square modulo $m(X)$. Using a trivial generalization of Algorithm 11.26, one finds that $(3X^3 + 6X^2 + 2X + 1)^2 \equiv f(X) \pmod{m(X)}$.

To conclude this part, let us remark that the computation of the trace of an element in an OEF enjoys the same kind of improvements as in characteristic 2, cf. Remark 11.57 (ii).

11.3.6 Specific improvements for degrees 3 and 5

For some applications, like the implementation of trace zero varieties, cf. Section 15.3, one needs to work in an extension field $\mathbb{F}_{p^d}$ of small degree d. In particular, $d = 3$ and $d = 5$ are interesting there. Some specific tricks can be used to make multiplication and inversion more efficient.

We use an explicit description of the field extension as $\mathbb{F}_{p^d} = \mathbb{F}_p[\theta]$, where θ is a root of an irreducible binomial $X^d - \omega$. Since $d = 3$ or 5 is prime, $X^d - \omega$ is irreducible whenever $p \equiv 1 \pmod{d}$ and ω is not a d-th power in $\mathbb{F}_p$. As $\mathbb{F}_p$ contains a d-th root of unity ζ, the roots of $X^d - \omega$ are $\theta, \zeta\theta, \ldots, \zeta^{d-1}\theta$.

Remark 11.72 It is very likely that there exists ω of small integer value, which is not a d-th power. In fact, by Čebotarev's density theorem, we have with probability $1/d$ that $p \equiv 1 \pmod{d}$ and $X^d - 2$ is irreducible over $\mathbb{F}_p$. With even larger probability, one can find some small ω such that $X^d - \omega$ is irreducible over $\mathbb{F}_p$ and the multiplication by ω, i.e., the reduction modulo the irreducible binomial, can be computed by additions only.

We shall write all elements of $\mathbb{F}_{p^3}$, respectively $\mathbb{F}_{p^5}$, as polynomials in θ of degrees at most 2, respectively 4, over $\mathbb{F}_p$. Addition, subtraction, and negation of elements of $\mathbb{F}_{p^d}$ are performed component-wise. If ω is small we can ignore the costs of reducing modulo $X^d - \omega$.

11.3.6.a Multiplication and squaring

Multiplication of elements of $\mathbb{F}_{p^d}$ is split into multiplication of the corresponding polynomials in θ and then reduction of the result using the fact that $\theta^d = \omega$.

Multiplication for $d = 3$

Multiplication in degree 3 extensions is done using Karatsuba's method, which we detail here to have the exact operation count. Let us multiply $\alpha = \sum_{i=0}^{2} a_i\theta^i$ with $\beta = \sum_{i=0}^{2} b_i\theta^i$. We have

$$\begin{aligned}\alpha\beta &= a_0b_0 + \big((a_0+a_1)(b_0+b_1) - a_0b_0 - a_1b_1\big)\theta \\ &\quad + \big((a_0+a_2)(b_0+b_2) - a_0b_0 - a_2b_2 + a_1b_1\big)\theta^2 \\ &\quad + \big((a_1+a_2)(b_1+b_2) - a_1b_1 - a_2b_2\big)\theta^3 + (a_2b_2)\theta^4.\end{aligned} \tag{11.6}$$

It enables us to multiply two degree 2 polynomials by 6 multiplications. By delaying all modular reductions and using incomplete reduction (see [AvMi 2004]), we need to perform 3 modular reductions modulo p.

Multiplication for $d = 5$

In the degree 5 extension case, to multiply $\alpha = \sum_{i=0}^{4} a_i\theta^i$ with $\beta = \sum_{i=0}^{4} b_i\theta^i$ we put $\xi = \theta^3$ and let $A_0 = \sum_{i=0}^{2} a_i\theta^i$, $A_1 = a_3 + a_4\theta$, $B_0 = \sum_{i=0}^{2} b_i\theta^i$, and $B_1 = b_3 + b_4\theta$. Karatsuba's method is then used to obtain

$$\begin{aligned}\alpha\beta &= (A_0 + A_1\xi)(B_0 + B_1\xi) \\ &= A_0B_0 + \big((A_0+A_1)(B_0+B_1) - A_0B_0 - A_1B_1\big)\xi + A_1B_1\xi^2.\end{aligned}$$

The product A_1B_1 is computed using Karatsuba's method, and A_0B_0 and $(A_0 + A_1)(B_0 + B_1)$ are both computed using (11.6). Note, however, that having A_0, B_0 of degree 2 and A_1, B_1 of degree 1, the coefficients of θ^4 in A_0B_0 and $(A_0 + A_1)(B_0 + B_1)$ are the same, so we can save one $\mathbb{F}_p$-multiplication. The amount of $\mathbb{F}_p$-multiplications needed to multiply two degree 4 polynomials is thus $3 + 2 \times 6 - 1 = 14$. By delaying all modular reductions and using incomplete reduction, we need to compute just 5 modular reductions modulo p.

Squaring

The squarings are more efficiently carried out using the schoolbook method, since this reduces the number of additions significantly and in several libraries, squarings in $\mathbb{F}_p$ are no cheaper than

ordinary multiplications. If this is not the case one should implement both versions (the schoolbook version and the Karatsuba one) and compare their running time.

For $d = 3$, we need 3 squarings and 3 multiplications in $\mathbb{F}_p$ by

$$(a_0 + a_1\theta + a_2\theta^2)^2 = a_0^2 + \omega a_1 a_2 + (\omega a_2^2 + a_0 a_1)\theta + (a_1^2 + a_0 a_2)\theta^2.$$

Likewise, for $d = 5$ we have 5 squarings and 10 multiplications. The number of modular reductions are again 3 and 5 as in the case of multiplications.

11.3.6.b Inversion

For the inversion, the difference from the general OEF approach becomes obvious. To compute the inverse of $\alpha \in \mathbb{F}_{p^d}$, we can consider the multiplication as a linear map and determine a preimage. This method is faster for $d = 3$ and also for $d = 2$ but we do not investigate that case any further. Note that for $d = 5$, the general method made explicit is faster.

Inversion for $d = 3$

Let $\beta = b_0 + b_1\theta + b_2\theta^2$, with $b_0, b_1, b_2 \in \mathbb{F}_p$, be the inverse of $\alpha = a_0 + a_1\theta + a_2\theta^2 \in \mathbb{F}_{p^3}$ and using $\theta^3 = \omega$, the relation $\alpha\beta = 1$ can be written as:

$$\begin{bmatrix} a_0 & a_2\omega & a_1\omega \\ a_1 & a_0 & a_2\omega \\ a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Hence

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_0 & a_2\omega & a_1\omega \\ a_1 & a_0 & a_2\omega \\ a_2 & a_1 & a_0 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = (a_0^3 + \omega\, a_1^3 + \omega^2 a_2^3 - 3\,\omega\, a_0 a_1 a_2)^{-1} \begin{bmatrix} a_0^2 - \omega\, a_1 a_2 \\ \omega\, a_2^2 - a_0 a_1 \\ a_1^2 - a_0 a_2 \end{bmatrix}.$$

From this formula we obtain a method for inverting elements in $\mathbb{F}_{p^3}$, which requires (ignoring multiplications by 3 and by ω) just one inversion, 3 squarings, and 9 multiplications in $\mathbb{F}_p$.

This method can be generalized to very small extensions. It is described e.g., in [KOMO+ 1999].

Inversion for $d = 5$

In this case we use the inversion technique described in Algorithm 11.66 but can save a bit by combining the powers of the Frobenius automorphism, i.e., making the addition chain explicit.

Thus we compute the inversion in $\mathbb{F}_{p^5}$ as:

$$\alpha^{-1} = \frac{(\alpha^p \alpha^{p^2} (\alpha^p \alpha^{p^2})^{p^2}}{\alpha \big(\alpha^p \alpha^{p^2} (\alpha^p \alpha^{p^2})^{p^2}\big)}.$$

Note that if the result of a $\mathbb{F}_{p^5}$-multiplication is known in advance to be in $\mathbb{F}_p$ (such as the norm), its computation requires just 5 $\mathbb{F}_p$-multiplications. This way we compute inverses in $\mathbb{F}_{p^5}$ by one inversion and 50 multiplications in $\mathbb{F}_p$. This strategy is optimal for $d = 5$ and needs less multiplications than the generalization of the linear algebra approach used for $d = 3$.