Subquadratic-Time Factoring of Polynomials over Finite Fields*

Erich Kaltofen[1] and Victor Shoup[2]

[1]Department of Mathematics, North Carolina State University
Raleigh, North Carolina 27695-8205; Internet: `kaltofen@eos.ncsu.edu`
URL: `http://www4.ncsu.edu/~kaltofen`

[2]Bellcore, 445 South Str.
Morristown, New Jersey 07960-6438; Internet: `shoup@bellcore.com`

Abstract

New probabilistic algorithms are presented for factoring univariate polynomials over finite fields. The algorithms factor a polynomial of degree n over a finite field of constant cardinality in time $O(n^{1.815})$. Previous algorithms required time $\Theta(n^{2+o(1)})$. The new algorithms rely on fast matrix multiplication techniques. More generally, to factor a polynomial of degree n over the finite field $\mathbb{F}_q$ with q elements, the algorithms use $O(n^{1.815} \log q)$ arithmetic operations in $\mathbb{F}_q$.

The new "baby step/giant step" techniques used in our algorithms also yield new fast practical algorithms at super-quadratic asymptotic running time, and subquadratic-time methods for manipulating normal bases of finite fields.

1 Introduction

In this paper, we present a new probabilistic approach for factoring univariate polynomials over finite fields. The resulting algorithms factor a polynomial of

*This material is based on work supported in part by the National Science Foundation under Grant No. CCR-9319776 (first author) and by an Alexander von Humboldt Research Fellowship (second author).

A major part of the work was performed while the first author was at Rensselaer Polytechnic Institute, Department of Computer Science, in Troy, New York and while the second author was at the Universität des Saarlandes, FB 14–Informatik, in Saarbrücken, Germany.

Appears in *Mathematics of Computation*, vol. 67(223), pp. 1179-1197, July 1998; a preliminary version of this paper appears in the Proc. 27th Annual ACM Symp. Theory of Computing, ACM Press, pp. 398–406 (1995).

degree n over a finite field $\mathbb{F}_q$ whose cardinality q is constant in time $O(n^{1.815})$. The best previous algorithms required time $\Theta(n^{2+o(1)})$.

This running-time bound relies on fast matrix multiplication algorithms. Let ω be an exponent of matrix multiplication; that is, ω is chosen so that we can multiply two $n \times n$ matrices using $O(n^\omega)$ arithmetic operations (we assume that $2 < \omega \leq 3$). Using the result of Coppersmith & Winograd (1990), we can take $\omega < 2.375477$.

More generally, we prove the following:

Theorem 1 *For any $0 \leq \beta \leq 1$, there exists a probabilistic algorithm for factoring a univariate polynomial of degree n over a finite field $\mathbb{F}_q$ that uses an expected number of*

$$O(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2} + n^{1+\beta+o(1)} \log q)$$

arithmetic operations in $\mathbb{F}_q$. In particular, choosing $\omega < 2.375477$ and minimizing the exponent of n, we get $O(n^{1.815} \log q)$ operations in $\mathbb{F}_q$.

Relation to Previous Work

The first random polynomial-time algorithm for this problem is due to Berlekamp (1970). Berlekamp's algorithm reduces the problem to that of finding elements in the null space of an $n \times n$ matrix over $\mathbb{F}_q$. Using standard techniques from linear algebra, Berlekamp's algorithm can be implemented so as to use an expected number of $O(n^\omega + n^{1+o(1)} \log q)$ operations in $\mathbb{F}_q$. Note that the algorithm by Rabin (1980) has an inferior running time, but Rabin completes the mathematical justification for the expected running time of the probabilistic Berlekamp method.

A very different algorithm is described by Cantor and Zassenhaus (1981) (see also Ben-Or (1981), especially for the case where the characteristic is 2). Starting with a square-free polynomial, that algorithm first separates the irreducible factors of distinct degree (distinct-degree factorization), and then completely factors each of the resulting factors (equal-degree factorization). The Cantor/Zassenhaus algorithm can be implemented so as to use an expected number of $O(n^{2+o(1)} \log q)$ operations in $\mathbb{F}_q$.

Von zur Gathen & Shoup (1992) developed new algorithmic techniques that essentially allow one to implement the Cantor/Zassenhaus algorithm so that it uses an expected number of $O(n^{2+o(1)} + n^{1+o(1)} \log q)$ operations in

$\mathbb{F}_q$. Their techniques allow one to solve the special problem of equal-degree factorization using an expected number of $O(n^{(\omega+1)/2+o(1)} + n^{1+o(1)} \log q)$ operations in $\mathbb{F}_q$.

Niederreiter (1993) developed an alternate approach to factoring polynomials over finite fields. However, from a complexity point of view this method is closely related to Berlekamp's original algorithm (Fleischmann 1993, Niederreiter and Göttfert 1994).

Kaltofen & Lobo (1994) adapted the linear system solver of Wiedemann (1986) to Berlekamp's algorithm. Utilizing techniques from von zur Gathen & Shoup, they show how their Black Box Berlekamp algorithm can be implemented so as to use an expected number of $O(n^{2+o(1)} + n^{1+o(1)} \log q)$ in $\mathbb{F}_q$.

Notice that at $\beta = 0$, the running-time of our algorithm matches that of Berlekamp's, and at $\beta = 1$ it matches that of Cantor/Zassenhaus, so that in some sense it interpolates between these two algorithms.

When $\log q$ is not too large in relation to n, then our new algorithm is asymptotically faster than previous algorithms. This is certainly clear if q is a constant and $\omega < 3$. Also, for $\omega < 2.375477$, as n and q tend to infinity with $\log q = O(n^{0.454})$, our new algorithm uses $O(n^{2-\Omega(1)})$ operations in $\mathbb{F}_q$, whereas the best previous algorithms require $\Theta(n^{2+o(1)})$ operations. Figure 1 plots the running times of selected algorithms in dependence of both n and $\log q$ when a fast matrix multiplication algorithm is used. The method by von zur Gathen & Shoup to-date remains the champion when $\log q = \Omega(n^{0.454})$ and simultaneously $\log q = O(n^{1.375477})$. At larger fields the term $n \log q$ in the Berlekamp algorithm dominates all running times.

Overview

Our Theorem 1 is proved using the Cantor/Zassenhaus strategy. The main technical contribution here is a subquadratic distinct-degree factorization algorithm, which is based on a "baby step/giant step" strategy. Our Fast Cantor/Zassenhaus algorithm is described in §2.

We also show how to modify the Black Box Berlekamp algorithm, using a very similar baby step/giant step technique, to get a subquadratic-time algorithm as well. This algorithm is described in §3. Interestingly, our techniques for the Black Box Berlekamp algorithm lead to subquadratic algorithms for finding a normal element in a finite field and for converting to and from normal coordinates. We present those algorithms in §4.

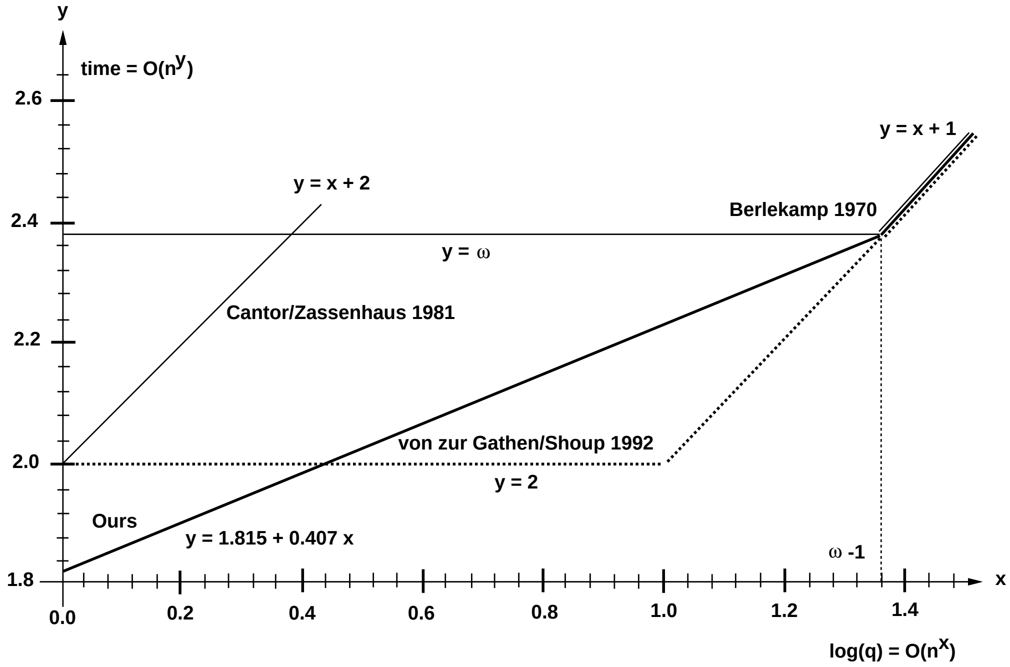

Figure 1: Running time comparisons for variable field cardinality

At the heart of our algorithms is the following problem. Given polynomials f, g, and h in $\mathbb{F}_q[x]$ of degree bounded by n, compute $g(h) \bmod f \in \mathbb{F}_q[x]$. Recently, this so-called *modular polynomial composition* problem has arisen in many contexts (von zur Gathen & Shoup 1992, Shoup 1994a). The algorithm of Brent & Kung (1978) solves this problem using $O(n^{(\omega+1)/2})$ operations in $\mathbb{F}_q$.

Any improvement in the complexity of this problem would yield an improvement in the complexity of factoring. Indeed, if this problem could be solved using $O(n^{1+o(1)})$ operations in $\mathbb{F}_q$, then our Fast Cantor/Zassenhaus algorithm could be implemented so as to use $O(n^{1.5+o(1)} + n^{1+o(1)} \log q)$ operations in $\mathbb{F}_q$.

Our algorithms rely on fast multiplication of matrices, indeed of $n^{1/2} \times n^{1/2}$ matrices, and therefore are not particularly practical. Interestingly, however, the techniques themselves can be adapted so as to give a quite practical factoring algorithm that uses

$$O(n^{2.5} + n^{1+o(1)} \log q)$$

operations in $\mathbb{F}_q$ and space for $O(n^{1.5})$ elements in $\mathbb{F}_q$, where the implied "big-O" constants are quite reasonable. From practical experience, we have found that when q is a large prime, this new algorithm allows much larger polynomials to be factored using a reasonable amount of space and time than was previously possible using other algorithms. This is briefly discussed in §5; a more complete discussion, including a description of an implementation of this algorithm as well as the results of empirical tests, is given in Shoup (1994b).

To attain a subquadratic running time, our algorithms rely on randomization. Even if we restrict ourselves to the field $\mathbb{F}_2$, the asymptotically fastest known deterministic algorithm (Shoup 1990) runs in time $O(n^{2+o(1)})$, and it remains an open problem to find a subquadratic deterministic algorithm.

2 The Fast Cantor/Zassenhaus Algorithm

Like the original Cantor/Zassenhaus algorithm, our algorithm splits the problem of factoring into three sub-problems:

Square-free factorization The input is a polynomial $f \in \mathbb{F}_q[x]$ of degree n. The output is $f_1, \ldots, f_n \in \mathbb{F}_q[x]$ such that

$$f = f_1 \cdot f_2^2 \cdot \cdots \cdot f_n^n.$$

Distinct-degree factorization The input is a square-free polynomial $f \in \mathbb{F}_q[x]$ of degree n. The output is $f^{[1]}, \ldots, f^{[n]} \in \mathbb{F}_q[x]$ such that for $1 \leq d \leq n$, $f^{[d]}$ is the product of the monic irreducible factors of f of degree d.

Equal-degree factorization The input is a polynomial $f \in \mathbb{F}_q[x]$ of degree n and an integer d such that f is the product of distinct monic irreducible polynomials, each of degree d. The output is the set of irreducible factors of f.

The factoring algorithm proceeds in three stages as follows. In the first stage, the input polynomial is fed into a square-free factorizer. In the second stage, the non-trivial outputs from the first stage are fed into distinct-degree factorizers. In the last stage, the non-trivial outputs from the second stage are fed into equal-degree factorizers.

The square-free factorization problem can be solved on degree n inputs using $O(n^{1+o(1)} + n \log q)$ operations in $\mathbb{F}_q$, using the algorithm of Yun (see Knuth 1981).

The equal-degree factorization problem can be solved on degree n inputs with the probabilistic algorithm of von zur Gathen & Shoup (1992) using an expected number of

$$O(n^{(\omega+1)/2+o(1)} + n^{1+o(1)} \log q),$$

or

$$O(n^{1.688} + n^{1+o(1)} \log q)$$

operations in $\mathbb{F}_q$.

We shall now present a family of (deterministic) algorithms for the distinct-degree factorization problem, parameterized by β with $0 \leq \beta \leq 1$, that uses

$$O(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2} + n^{1+\beta+o(1)} \log q)$$

operations in $\mathbb{F}_q$. For any choice of β, this dominates the time required for square-free factorization and equal-degree factorization, establishing Theorem 1.

Our distinct-degree factorization algorithm uses a "baby step/giant step" strategy that exploits the following fact.

Lemma 1 *For nonnegative integers i and j, the polynomial $x^{q^i} - x^{q^j} \in \mathbb{F}_q[x]$ is divisible by precisely those irreducible polynomials in $\mathbb{F}_q[x]$ whose degree divides $i - j$.*

Proof. Assume without loss of generality that $i \geq j$. Then $x^{q^i} - x^{q^j} = (x^{q^{i-j}} - x)^{q^j}$, and the result follows at once from the factorization of $x^{q^k} - x$, which consists of all irreducible factors whose degree is a divisor of k (see Lidl & Niederreiter 1983, Theorem 3.20). □

We first present a high-level description of our distinct-degree factorization algorithm. The details of how each step is to be implemented are deferred until later.

Algorithm D This algorithm takes as input a square-free polynomial $f \in \mathbb{F}_q[x]$ of degree n. The output is $f^{[1]}, \ldots, f^{[n]} \in \mathbb{F}_q[x]$ such that for $1 \leq d \leq n$, $f^{[d]}$ is the product of the monic irreducible factors of f of degree d. The algorithm is parameterized by a constant β, with $0 \leq \beta \leq 1$.

Step D1 (compute baby steps) Let $l = \lceil n^{\beta} \rceil$. For $0 \leq i \leq l$, compute $h_i = x^{q^i} \bmod f \in \mathbb{F}_q[x]$.

Step D2 (compute giant steps) Let $m = \lceil n/2l \rceil$. For $1 \leq j \leq m$, compute $H_j = x^{q^{lj}} \bmod f \in \mathbb{F}_q[x]$.

Step D3 (compute interval polynomials) For $1 \leq j \leq m$, compute

$$I_j = \prod_{0 \leq i < l} (H_j - h_i) \bmod f \in \mathbb{F}_q[x].$$

Note that by Lemma 1, the polynomial I_j is divisible by those irreducible factors of f whose degree divides an integer k with $(j-1)l < k \leq jl$.

Step D4 (compute coarse DDF) In this step, we compute polynomials $F_1, \ldots, F_m$, where $F_j = f^{[(j-1)l+1]} f^{[(j-1)l+2]} \cdots f^{[jl]}$. This is done as follows.

```
$f^* \leftarrow f$;
for $j \leftarrow 1$ to $m$ do
        $\{F_j \leftarrow \gcd(f^*, I_j);\ f^* \leftarrow f^*/F_j\ \}$
```

Step D5 (compute fine DDF) In this step, we compute the output polynomials $f^{[1]}, \ldots, f^{[n]}$. First, initialize $f^{[1]}, \ldots, f^{[n]}$ to 1. Then do the following.

```
for $j \leftarrow 1$ to $m$ do
      $\{g \leftarrow F_j$;
        for $i \leftarrow l-1$ down to 0 do
              $\{f^{[lj-i]} \leftarrow \gcd(g, H_j - h_i);\ g \leftarrow g/f^{[lj-i]}\}$
      $\}$
if $f^* \neq 1$ then $f^{[\deg(f^*)]} \leftarrow f^*$;
```

The correctness of this algorithm is clear from the comments contained therein.

Before establishing the running-time bound in Theorem 1, we begin with the following slightly weaker, but simpler, result.

Theorem 2 *Algorithm D can be implemented so as to use*

$$O(n^{(\omega+1)/2+1-\beta} + n^{1+\beta+o(1)} \log q)$$

operations in $\mathbb{F}_q$. In particular, choosing $\omega < 2.375477$ and minimizing the exponent of n, we get $O(n^{1.844} \log q)$ operations in $\mathbb{F}_q$.

The proof of Theorem 2 is based on the observation that for any positive integer r, if we are given $h = x^{q^r} \bmod f \in \mathbb{F}_q[x]$, then for any $g \in \mathbb{F}_q[x]$, we can compute $g^{q^r} \bmod f$ as $g(h) \bmod f \in \mathbb{F}_q[x]$. To solve this so-called "modular composition" problem, we use the following result.

Lemma 2 *Given a polynomial $f \in \mathbb{K}[x]$ of degree n over an arbitrary field $\mathbb{K}$, and polynomials $g, h \in \mathbb{K}[x]$ of degree less than n, we can compute the polynomial $g(h) \bmod f \in \mathbb{K}[x]$ using $O(n^{(\omega+1)/2})$ arithmetic operations in $\mathbb{K}$.*

Proof. This is essentially Algorithm 2.1 in Brent & Kung (1978). □

We now prove Theorem 2.

Step D1 is performed by iterating the standard repeated-squaring algorithm l times. This takes $O(n^{1+\beta+o(1)} \log q)$ operations in $\mathbb{F}_q$.

Step D2 is performed by setting $H_1 = h_l$, and then iterating the algorithm of Lemma 2, computing each H_j as $H_{j-1}(H_1) \bmod f \in \mathbb{F}_q[x]$. This takes $O(n^{(\omega+1)/2+1-\beta})$ operations in $\mathbb{F}_q$.

Step D3 is performed as follows. Let R be the ring $\mathbb{F}_q[x]/(f)$. We first compute the coefficients of the polynomial $H(Y) \in R[Y]$ of degree l, where

$$H(Y) = \prod_{0 \le i < l} (Y - (h_i \bmod f)).$$

Then we evaluate $H(Y)$ at the m points

$$(H_1 \bmod f), \ldots, (H_m \bmod f) \in R.$$

Using fast algorithms for multiplication of polynomials in $R[Y]$ (Cantor & Kaltofen 1991) Step D3 can be implemented so as to use $O(n^{1+\beta+o(1)} + n^{2-\beta+o(1)})$ operations in $\mathbb{F}_q$ (Aho et al. 1974).

In Step D4, we need to compute $O(m)$ GCD's and divisions, requiring $O(n^{2-\beta+o(1)})$ operations in $\mathbb{F}_q$.

To implement Step D5 efficiently, we first reduce each h_i modulo each F_j. Reducing one h_i modulo each F_j takes $O(n^{1+o(1)})$ operations in $\mathbb{F}_q$, using standard "Chinese remaindering" techniques (Aho et al. 1974) Thus, reducing all of the h_i's modulo all of the F_j's takes just $O(n^{1+\beta+o(1)})$ operations in $\mathbb{F}_q$. Also, we compute $H_j \bmod F_j$ for each F_j. This takes $O(n^{2-\beta+o(1)})$ operations in $\mathbb{F}_q$. With these pre-computations, the total cost of computing the GCD's and divisions in the inner loop amounts to $O(n^{1+\beta+o(1)})$ operations in $\mathbb{F}_q$. Thus the total cost of Step D5 is $O(n^{1+\beta+o(1)} + n^{2-\beta+o(1)})$ operations in $\mathbb{F}_q$.

That proves Theorem 2.

We now show how to modify the implementation of Step D2 to obtain the slightly better running-time bound of Theorem 1.

Theorem 3 *Algorithm D can be implemented so as to use*

$$O(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2} + n^{1+\beta+o(1)} \log q)$$

operations in $\mathbb{F}_q$. *In particular, choosing* $\omega < 2.375477$ *and minimizing the exponent of* n, *we get* $O(n^{1.815} \log q)$ *operations in* $\mathbb{F}_q$.

To prove this theorem, it will suffice to show that we can compute the polynomials $H_1, \ldots, H_m$ in Step D2 using $O(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2})$ operations in $\mathbb{F}_q$. This is an immediate consequence of the following two lemmas.

Lemma 3 *Given a polynomial* $f \in \mathbb{K}[x]$ *of degree* n *over an arbitrary field* $\mathbb{K}$, *and polynomials* $g_1, \ldots, g_k, h \in \mathbb{K}[x]$ *of degree less than* n, *where* $k = O(n)$, *we can compute*

$$g_1(h) \bmod f, \ldots, g_k(h) \bmod f \in \mathbb{K}[x]$$

using

$$O(n^{(\omega+1)/2} k^{(\omega-1)/2})$$

arithmetic operations in $\mathbb{K}$.

Proof. Setting $t = \lceil \sqrt{nk} \rceil$, we decompose each of the input polynomials $g_1, \ldots, g_k$ as

$$g_i = \sum_{0 \le j < n/t} g_{i,j} y^j, \quad y = x^t, \tag{1}$$

where the $g_{i,j}$'s are polynomials of degree less than t. We first compute the polynomials $h^{(i)} = h^i \bmod f$ for $0 \le i \le t$. Next, we compute all of the polynomials $g_{i,j}(h) \bmod f$ by computing the following product of an $n \times t$ matrix and a $t \times (k\lceil n/t \rceil)$ matrix:

$$\left[\; \vec{h}^{(0)} \; \middle| \; \ldots \; \middle| \; \vec{h}^{(t-1)} \; \right] \cdot \left[\; \vec{g}_{1,0} \; \middle| \; \ldots \; \middle| \; \vec{g}_{1,\lceil n/t \rceil -1} \; \middle| \; \ldots \; \middle| \; \vec{g}_{k,0} \; \middle| \; \ldots \; \middle| \; \vec{g}_{k,\lceil n/t \rceil -1} \; \right].$$

Here, we use the notation $\vec{\cdot}$ to denote the column vector consisting of the coefficients of a polynomial. This computation is done by performing $O(\sqrt{n/k})$ multiplications of $t \times t$ matrices. Finally, we compute for $1 \le i \le k$ the polynomial $g_i(h) \bmod f(x) \in \mathbb{K}[x]$ by substituting the polynomial $h^{(t)}$ for y in the formula (1), and performing a Horner evaluation scheme. This is done by iteratively performing $\lceil n/t \rceil - 1$ polynomial multiplications mod f and $O(n/t)$ polynomial additions.

It is easily seen that the dominant cost is again the matrix multiplication step, which can be carried out using the stated number of operations. □

We remark that when $k = 1$, the algorithm in the above proof is the same as Brent & Kung's modular composition algorithm.

Lemma 4 *Let $f \in \mathbb{F}_q[x]$ be a polynomial of degree n. Suppose that we are given $x^{q^r} \bmod f \in \mathbb{F}_q[x]$. Then we can compute*

$$x^{q^r} \bmod f,\, x^{q^{2r}} \bmod f,\, \ldots,\, x^{q^{kr}} \bmod f \in \mathbb{F}_q[x],$$

where $k = O(n)$, using

$$O(n^{(\omega+1)/2} k^{(\omega-1)/2})$$

operations in $\mathbb{F}_q$.

Proof. For $i \ge 1$, let $G_i = x^{q^{ir}} \bmod f \in \mathbb{F}_q[x]$. Assume we have computed $G_1, \ldots, G_m$. Then we can compute $G_{m+1}, \ldots, G_{2m}$ by computing $G_1(G_m) \bmod f, \ldots, G_m(G_m) \bmod f$ using the algorithm in the previous lemma.

So to compute $G_1, \ldots, G_k$ given G_1, we simply repeat the above "doubling" step $O(\log k)$ times. The stated running-time estimate then follows easily. □

3 The Fast Black Box Berlekamp Algorithm

In Kaltofen & Lobo (1994), a version of Berlekamp's factoring algorithm was given based on Wiedemann's (1986) sparse linear system solver. In this section, we show how to modify that algorithm to obtain a probabilistic, subquadratic-time algorithm.

We split this section into two parts. In §3.1, we review the ideas behind the Black Box Berlekamp algorithm, presenting a high-level description of that algorithm. Then in §3.2, we describe a subquadratic-time implementation, first proving a running time bound of $O(n^{1.880} + n^{1.808} \log q)$ operations in $\mathbb{F}_q$. We then modify this method to obtain the bound $O(n^{1.852}+n^{1.763} \log q)$. With yet a bit more work, we show how to obtain the bound $O(n^{1.815} \log q)$.

3.1 The Black Box Berlekamp Algorithm

We first recall the main ideas behind the Black Box Berlekamp algorithm. Suppose the coefficient field $\mathbb{F}_q$ has characteristic p. Let $f \in \mathbb{F}_q[x]$ be a monic square-free polynomial of degree n to be factored. Assume that the factorization of f into irreducibles is

$$f = f_1 \cdots f_r.$$

For $1 \leq i \leq r$, let $d_i = \deg(f_i)$, and let p^{e_i} be the highest power of p that divides d_i. Furthermore, let $e = \max\{e_i : 1 \leq i \leq r\}$.

Now consider the q-th power map $\sigma \colon \alpha \mapsto \alpha^q$ for $\alpha \in \mathbb{F}_q[x]/(f)$. Let $\phi \in \mathbb{F}_q[\lambda]$ be the minimum polynomial of σ over $\mathbb{F}_q$, i.e., ϕ is the monic polynomial of least degree such that $\phi(\sigma) = 0$. The polynomial ϕ can easily be described in terms of the degrees of the irreducible factors of f, as follows. By the Chinese remainder theorem we have the $\mathbb{F}_q$-algebra isomorphism

$$\mathbb{F}_q[x]/(f) \cong \mathbb{F}_q[x]/(f_1) \oplus \cdots \oplus \mathbb{F}_q[x]/(f_r).$$

For $1 \leq i \leq r$, let σ_i be the q-th power map on $\mathbb{F}_q[x]/(f_i)$, and let $\phi_i \in \mathbb{F}_q[\lambda]$ be its minimum polynomial. From the basic theory of finite fields, we know that $\phi_i = \lambda^{d_i} - 1$. Moreover, by the Chinese remainder theorem,

$$\phi = \mathrm{lcm}\{\phi_1, \ldots, \phi_r\} = \mathrm{lcm}\{\lambda^{d_1} - 1, \ldots, \lambda^{d_r} - 1\}.$$

Now consider the polynomial $\mu(\lambda) = \phi(\lambda)/(\lambda - 1)$, and the image $I_i \subset \mathbb{F}_q[x]/(f_i)$ of $\mu(\sigma_i)$. Since $(\sigma_i - 1)(\alpha) = \alpha^q - \alpha = 0$ for all $\alpha \in I_i$, it follows that $I_i \subset \mathbb{F}_q$. It is easily seen that $\lambda - 1$ divides $\lambda^{d_i} - 1$ exactly to the power p^{e_i}, which implies that $I_i = \mathbb{F}_q$ if $e_i = e$, and $I_i = \{0\}$ if $e_i < e$ (see Kaltofen & Lobo 1994, §3, for more details).

These considerations motivate the following recursive algorithm. The details of how each step is to be implemented are deferred until later.

Algorithm B The algorithm takes as input a monic square-free polynomial $f \in \mathbb{F}_q[x]$ of degree n, and produces as output the set of irreducible factors of f.

Step B1 (compute minimum polynomial) Probabilistically compute a polynomial $\phi^* \in \mathbb{F}_q[\lambda]$ that with probability at least 1/2 is equal to ϕ, the minimum polynomial of the q-th power map σ on $\mathbb{F}_q[x]/(f)$, and that otherwise divides ϕ.

Step B2 (evaluate polynomial) If $\phi^*(\lambda) = \lambda^n - 1$, then halt, as f is then certified to be irreducible. If $\lambda - 1$ does not divide $\phi^*(\lambda)$, go back to Step B1, as then ϕ^* is clearly erroneous.
Otherwise, set $\mu^*(\lambda) = \phi^*(\lambda)/(\lambda - 1)$, choose a random $\alpha \in \mathbb{F}_q[x]/(f)$, and compute

$$\alpha^* = (\mu^*(\sigma))(\alpha) \in \mathbb{F}_q[x]/(f).$$

Step B3 (split) Let $\alpha^* = (g \bmod f)$. Compute $h_1 = \gcd(g, f)$ and $h_2 = f/h_1$. If $\phi^* = \phi$ then the degrees of all irreducible factors of h_2 are divisible by p^e and the residues of h_2 modulo these factors are random elements in $\mathbb{F}_q \setminus \{0\}$. Compute $h^* \in \mathbb{F}_q[x]/(h_2)$ as

$$h^* = \begin{cases} g^{(q-1)/2} \bmod h_2 & \text{if } p > 2, \\ \sum_{j=0}^{k-1} g^{2^j} \bmod h_2 & \text{if } q = 2^k. \end{cases}$$

Recursively factor h_1, $h_2^* = \gcd(1 + h^* \bmod h_2, h_2)$ and h_2/h_2^*.

Before going into the details of each step, we first calculate a bound on the recursion depth of this algorithm.

Lemma 5 *The expected value of the recursion depth of Algorithm B is $O(\lceil \log_p n \rceil \log r)$, where r is the number of irreducible factors of f.*

Proof. Consider one invocation of the algorithm and recall the notation preceding the algorithm. Each factor f_i with $e_i = e$ will be separated from the factors f_j with $e_j < e$ in Step B3 with probability bounded away from 0 by a constant. If f has several factors with $e_i = e$, then each pair of such factors will be separated in Step B3 with probability bounded away from 0 by a constant. These statements follow easily from the fact that ϕ^* is correctly computed with probability 1/2, and from the discussion preceding the algorithm.

Using a standard argument (see, for example, Lemma 4.1 in von zur Gathen & Shoup 1992), at an expected depth of $O(\log r)$, all irreducible factors f_i with $e_i = e$ will be isolated, and the only reducible factors remaining will have $e_i < e$.

It follows that at an expected depth of $O(\lceil \log_p n \rceil \log r)$, all irreducible factors of f will be isolated. □

Next, we discuss the problem of computing ϕ^* in Step B1. Following Wiedemann (1986), this is done as follows. We choose random $\alpha \in \mathbb{F}_q[x]/(f)$ and a random $\mathbb{F}_q$-linear map $u\colon \mathbb{F}_q[x]/(f) \to \mathbb{F}_q$, and compute the minimum polynomial of the linearly generated sequence $\{a_i : a_i = u(\sigma^i(\alpha)) \text{ and } i \geq 0\}$. Using an asymptotically fast version of the Berlekamp-Massey algorithm (Massey 1969, Dornstetter 1987), given the first $2n$ terms of the sequence $\{a_i : i \geq 0\}$, we can determine the minimum polynomial $\phi_{\alpha,u} \in \mathbb{F}_q[\lambda]$ of this sequence using $O(n^{1+o(1)})$ operations in $\mathbb{F}_q$. In general, $\phi_{\alpha,u}$ divides ϕ, but the probability that $\phi_{\alpha,u} = \phi$ (for random α, u) may be less than 1/2, and indeed not even bounded away from 0 by a constant. To increase this probability, we repeat the above procedure some number $\rho(n,q)$ times, each time choosing a new α and a new u at random, thus obtaining polynomials ϕ_{α_i,u_i}, where $1 \leq i \leq \rho(n,q)$. Then we compute

$$\phi^* = \operatorname{lcm}\{\phi_{\alpha_i,u_i} : 1 \leq i \leq \rho(n,q)\}.$$

The value $\rho(n,q)$ can be chosen as indicated in the next lemma.

Lemma 6 *Let $\rho(n,q)$ be defined as follows. If $q \geq 4n$, then $\rho(n,q) = 1$. Otherwise,*

$$\rho(n,q) = \begin{cases} 6 & \text{if } q = 2, \\ 4 & \text{if } q = 3, \\ 3 & \text{if } 4 \leq q \leq 9, \\ 2 & \text{if } q \geq 11. \end{cases}$$

Then the probability that $\phi^* = \phi$ *is at least* $1/2$.

Proof. If $q \geq 4n$, then the result follows by the analysis of Kaltofen & Pan (1991). Otherwise we argue along the same lines as Wiedemann (1986, §VI). Suppose $\phi = \psi_1^{\eta_1} \cdots \psi_s^{\eta_s}$ is the factorization of ϕ into irreducibles. Suppose $\alpha \in \mathbb{F}_q[x]/(f)$ and $u\colon \mathbb{F}_q[x]/(f) \to \mathbb{F}_q$ are chosen at random. As above, let $\phi_\alpha \in \mathbb{F}_q[\lambda]$ be the minimum polynomial of the sequence $\{\sigma^i(\alpha) : i \geq 0\}$ and let $\phi_{\alpha,u}$ be the minimum polynomial of the sequence $\{u(\sigma^i(\alpha)) : i \geq 0\}$.

Claim. For any single j with $1 \leq j \leq s$, the probability that $\psi_j^{\eta_j}$ does not divide $\phi_{\alpha,u}$ is no more than

$$(2/q - 1/q^2)^\delta \quad \text{where } \delta = \deg(\psi_j).$$

We prove this claim by using a fact established by Wiedemann. He shows that there exists a surjective $\mathbb{F}_q$-linear map $\mathcal{L}\colon V \to W$ depending on α, where V is the linear space of $\mathbb{F}_q$-linear maps from $\mathbb{F}_q[x]/(f)$ to $\mathbb{F}_q$ and W is the linear space of polynomials of degree less than $\deg(\phi_\alpha)$, such that for any $u \in V$ we have $\phi_{\alpha,u} = \phi_\alpha / \gcd(\phi_\alpha, \mathcal{L}(u))$. Suppose now that $\psi_j^{\eta_j}$ divides ϕ_α. Then $\psi_j^{\eta_j}$ divides $\phi_{\alpha,u}$ if ψ_j does not divide $\mathcal{L}(u)$, which for a random u is a random polynomial over $\mathbb{F}_q$ of degree less than $\deg(\phi_\alpha)$. Clearly, of all $q^{\deg(\phi_\alpha)}$ such polynomials only $q^{\deg(\phi_\alpha)-\delta}$ are divisible by ψ_j, so the probability that ψ_j does not divide $\mathcal{L}(u)$ is $1 - 1/q^\delta$. Furthermore, by considering the rational canonical form of the linear transform σ we can show the existence of an element α_0 such that $\phi_{\alpha_0} = \phi$. As $\mathcal{L}$ is surjective, there also must exist a u_0 such that $\phi_{\alpha_0,u_0} = \phi_{\alpha_0} = \phi$. By switching the rôles of u and α, as Wiedemann does in the proof of his Proposition 4, we can obtain that the probability that $\psi_j^{\eta_j}$ divides ϕ_{α,u_0} is $1 - 1/q^\delta$. Thus, the probability that $\psi_j^{\eta_j}$ divides ϕ_α is no less.

Therefore, the probability that $\psi_j^{\eta_j}$ does not divide $\phi_{\alpha,u}$ is no less than $1 - (1 - 1/q^\delta)^2 = 2/q^\delta - 1/q^{2\delta}$. The claim then follows from the inequality $2c^\delta - c^{2\delta} \leq (2c - c^2)^\delta$, which holds for all real numbers c with $0 < c \leq 1/2$ and all integers $\delta \geq 1$.

From this claim, one sees that if this procedure is repeated $k = \rho(n, q)$ times, and we compute ϕ^* as the polynomial least common multiple of all of the ϕ_{α_i,u_i}'s, then the probability that $\psi_j^{\eta_j}$ does not divide ϕ^* is at most $(2/q - 1/q^2)^{k \deg(\psi_j)}$.

Since the factorization of $x^{q^l} - x$ includes each irreducible polynomial of degree l, the number of irreducibles of degree l is at most q^l/l. Hence summing over all irreducible polynomials dividing ϕ, as well as all those irreducible polynomials *not* dividing ϕ, we get an upper bound on the probability that $\phi^* \neq \phi$ of

$$\sum_{l\geq 1} \frac{q^l}{l}\,(2/q-1/q^2)^{k\,l} = -\log(1-q(2/q-1/q^2)^k).$$

The lemma then follows from a simple numerical calculation. $\square$

For sufficiently large q it is possible to improve the expected recursion depth of Algorithm 1 to $O(\log_p n + \log r)$ as follows. First, we make the probability that $\phi^* \neq \phi$ in Step B1 smaller than $1/q$ by computing the least common multiple of ρ' many polynomials ϕ_{α_i,u_i}, where

$$\rho'(n,q) = \begin{cases} 6 & \text{if } q=2, \\ 5 & \text{if } q=3, \\ 4 & \text{if } q=4,5, \\ 3 & \text{if } q\geq 7, \\ 2 & \text{if } q>4n^2. \end{cases}$$

The values for ρ' follow as in the proof of Lemma 6. Suppose now that $\phi^* = \phi$. Then the probability that all r_e irreducible factors f_i of f with $e_i = e$ are separated from the remaining factors in a single gcd in Step B3 is equal to the probability that r_e random elements of $\mathbb{F}_q$ are simultaneously not equal to zero, which is $(1-1/q)^{r_e}$. Overall, we succeed to separate the f_i with $e_i = e$ from the rest in a single try with probability $\geq (1-1/q)\,(1-1/q)^{r_e}$.

Let r_η be the number of irreducible factors f_j of f with $e_j = \eta$. Then with probability no less than

$$\prod_{\eta=0}^{e}\left(1-\frac{1}{q}\right)^{1+r_\eta} \geq \left(1-\frac{1}{q}\right)^{e+r+1} \geq \left(\frac{1}{4}\right)^{(e+r+1)/q}$$

the factors with $e_j = \eta$ are split off for all η by a recursion of depth $e+1$. Now suppose that for a constant $c > 0$ we have

$$q \geq c\,(\lfloor \log_p n\rfloor + r + 1).$$

Then the probability that all pairs f_i and f_j with $e_i \neq e_j$ are separated at recursion depth $O(\log_p n)$ is bounded away from 0 by a constant (namely $4^{-1/c}$). It is then easy to show that the expected depth is $O(\log_p n + \log r)$. We note that for constant q, however, the expected recursion depth can be $\Theta((\log n)^2)$.

3.2 A Subquadratic-Time Implementation

Theorem 4 *For any constant β with $0 \leq \beta \leq 1$, Algorithm B can be implemented so as to use an expected number of*

$$O(n^{(\omega+1)/2+(3-\omega)|\beta-1/2|+o(1)} + n^{(\omega+1)/2+1-\beta+o(1)} + n^{1+\beta+o(1)} \log q) \qquad (2)$$

operations in $\mathbb{F}_q$. In particular, choosing $\omega < 2.375477$ and minimizing the exponent of n, we get $O(n^{1.880} + n^{1.808} \log q)$ operations in $\mathbb{F}_q$.

Remark The first term in (2) is dominated by the second exactly when $\beta < (\omega - 5)/(2(\omega - 4))$, and thus at least when $\beta < 3/4$.

To prove Theorem 4, we first show that one invocation of Algorithm B, not counting the recursive calls, can be implemented so as to satisfy the bound in Theorem 4. By Lemma 5, multiplying this by $O((\log n)^2)$ gives a bound on the total cost of the algorithm, and thus the theorem will follow.

The cost of Step B3 is $O(n^{1+o(1)} \log q)$ operations in $\mathbb{F}_q$, and the cost of the Berlekamp-Massey algorithm in Step B1 is $O(n^{1+o(1)})$ operations in $\mathbb{F}_q$. So to prove our result, we have to solve the following two types of problems within the stated time bounds.

automorphism projection Given $\alpha \in \mathbb{F}_q[x]/(f)$, $u: \mathbb{F}_q[x]/(f) \to \mathbb{F}_q$, and a positive integer $k = O(n)$, compute $u(\sigma^i(\alpha)) \in \mathbb{F}_q$ for all i with $0 \leq i < k$.

automorphism evaluation Given $\alpha \in \mathbb{F}_q[x]/(f)$ and a polynomial $\mu \in \mathbb{F}_q[\lambda]$ of degree less than k, where $k = O(n)$, compute $(\mu(\sigma))(\alpha) \in \mathbb{F}_q[x]/(f)$.

We first claim that these two problems are computationally equivalent, in a very strong sense. Consider the $n \times k$ matrix A whose columns consist of the coordinates with respect to the natural power basis $1, x, x^2, \ldots, x^{n-1}$ for $\mathbb{F}_q[x]/(f)$ of $\alpha, \sigma(\alpha), \ldots, \sigma^{k-1}(\alpha)$. Then the automorphism projection problem consists of multiplying A on the left by a row vector $(u_0, \ldots, u_{n-1}) \in \mathbb{F}_q^{1 \times n}$. The automorphism evaluation problem consists of multiplying A on the right by a column vector $(\mu_0, \ldots, \mu_{k-1})^{\mathrm{T}} \in \mathbb{F}_q^{k \times 1}$. Thus these two problems are merely the transpose of each each other, and by the so-called *transposition principle* a straight-line program of length l for one can be quickly converted (in time $O(l)$) into a straightline program of length $O(l)$ for the other, provided the straight-line program computes linear forms in the input variables $\{u_i\}$ (respectively, $\{\mu_i\}$). It should be noted that this observation applies to the Wiedemann algorithm in general. For example, in Algorithm 1 in Wiedemann (1986) step 4 and step 6 are computationally equivalent within a constant factor. The discovery of the transposition principle goes back to the Ph.D. thesis of Fiduccia (1973), Theorem 2 for multiplications (see also Fiduccia 1972, Theorem 4) and Theorem 5 for additions and subtractions. The additive version with a similar proof is rediscovered in (Kaminski et al. 1988). Furthermore, we remark that the transposition principle is a direct consequence of the so-called reverse mode in automatic differentiation, see Canny et al. (1989); for reverse mode see also Ostrowski et al. (1971), Linnainmaa (1976), Baur & Strassen (1983), and Griewank (1991).

Thus, to prove our theorem, it will suffice to prove the required bound for just one of these problems. We prove it for the automorphism evaluation problem. The following algorithm for automorphism evaluation is based on the same "baby step/giant step" strategy used in Brent & Kung's modular composition algorithm.

Algorithm AE This algorithm takes as input an element $\alpha \in \mathbb{F}_q[x]/(f)$, where $f \in \mathbb{F}_q[x]$ is of degree n, and a polynomial $\mu \in \mathbb{F}_q[\lambda]$ of degree less than k, where $k = O(n)$. The output is $(\mu(\sigma))(\alpha) \in \mathbb{F}_q[x]/(f)$. The algorithm is parameterized by a constant β, with $0 \le \beta \le 1$.

We set $t = \lceil n^\beta \rceil$ and $m = \lceil k/t \rceil$, and we write μ as

$$\mu = \sum_{0 \le j < m} \mu_j(\lambda)\lambda^{tj},$$

where each $\mu_j \in \mathbb{F}_q[\lambda]$ has degree less than t.

Then we have

$$(\mu(\sigma))(\alpha) = \sum_{0 \le j < m} \sigma^{tj}(\,(\mu_j(\sigma))(\alpha)\,).$$

The algorithm proceeds as follows.

Step AE1 Compute $\sigma^i(\alpha) \in \mathbb{F}_q[x]$, for all i with $0 \le i < t$, by iterating a repeated squaring algorithm.

Step AE2 Using the values computed in Step AE1, we next compute $(\mu_j(\sigma))(\alpha) \in \mathbb{F}_q[x]$ for all j with $0 \le j < m$. This is done by multiplying an $m \times t$ matrix by a $t \times n$ matrix.

Step AE3 We compute $x^{q^t} \bmod f$, using the method of Algorithm 5.2 in von zur Gathen & Shoup (1992), which requires the computation of $x^q \bmod f$, plus $O(\log t)$ modular polynomial compositions.

Step AE4 We use the values computed in Steps AE2 and AE3 together with a Horner evaluation scheme to get $(\mu(\sigma))(\alpha)$. This is done iteratively, performing $m-1$ modular compositions.

Lemma 7 *Algorithm AE can be implemented so as to use*

$$O(n^{(\omega+1)/2+(3-\omega)|\beta-1/2|} + n^{(\omega+1)/2+1-\beta} + n^{1+\beta+o(1)} \log q)$$

operations in $\mathbb{F}_q[x]$. Moreover, the algorithm satisfies the conditions of the transposition principle.

Proof.

Step AE1 takes $O(n^{1+\beta+o(1)} \log q)$ operations in $\mathbb{F}_q$.

In Step AE2, if $\beta > 1/2$, we compute $O(n^{1+\beta}/n^{2(1-\beta)})$ multiplications of square matrices of dimension $O(n^{1-\beta})$; otherwise, if $\beta \leq 1/2$, we perform $O(n^{2-\beta}/n^{2\beta})$ multiplications of square matrices of dimension $O(n^{\beta})$. In either case, the number of operations in $\mathbb{F}_q$ is readily calculated as $O(n^{(\omega+1)/2+(3-\omega)|\beta-1/2|})$.

Step AE3 takes $O(n^{(\omega+1)/2} + n^{1+o(1)} \log q)$ operations in $\mathbb{F}_q$.

Step AE4 takes $O(n^{(\omega+1)/2+1-\beta})$ operations in $\mathbb{F}_q$.

To prove the second assertion in the lemma, one easily checks that all of the values computed by the algorithm are linear in the input variables representing the coefficients of μ. □

Although the above discussion implies the *existence* of an algorithm for automorphism projection, it is not too difficult to give an explicit algorithm. We describe one here.

Let Q be the $n \times n$ matrix representing the q-th power map σ on $\mathbb{F}_q[x]/(f)$, with respect to the natural power basis. The matrix Q is the transpose of Petr's matrix (see Schwarz 1956) computed in the classical Berlekamp algorithm. We represent the projection map u as a row vector $\vec{u}^{\mathrm{T}}$, and we let $\vec{\alpha}$ be the column vector consisting of the coordinates of α. We want to compute the values

$$\vec{u}^{\mathrm{T}} Q^i \vec{\alpha} \qquad (0 \leq i < k). \tag{3}$$

Algorithm AP This algorithm takes as input α and u as above and computes the quantities (3). The algorithm is parameterized by a constant β, with $0 \leq \beta \leq 1$.

Set $t = \lceil n^\beta \rceil$ and $m = \lceil k/t \rceil$. We rewrite (3) as

$$(\vec{u}^{\mathrm{T}} Q^{tj}) \cdot (Q^i \vec{\alpha}) \qquad (0 \leq j < m,\ 0 \leq i < t). \tag{4}$$

The algorithm proceeds as follows.

Step AP1 Compute the vectors $Q^i\vec{\alpha}$, for $0 \leq i < t$, by iterating a repeated squaring algorithm $t-1$ times (left multiplication by Q is the same as q-th powering).

Step AP2 Compute x^{q^t} as in Step AE3.

Step AP3 Compute the vectors $\vec{u}^{\mathrm{T}} Q^{tj}$, for $0 \leq j < m$, by iteratively computing $m-1$ "transposed" modular polynomial compositions to carry out the right multiplications by Q^t, each of which (by the transposition principle) has the same cost as an ordinary modular composition (with $x^{q^t} \bmod f$).

Step AP4 Using the values computed in Steps AP1 and AP3, all of the values in (4) are computed by multiplying an $m \times n$ matrix by an $n \times t$ matrix.

It is straightforward to check that Lemma 7 also holds for Algorithm AP. We point out that an explicit algorithm for the "transposed" modular composition problem in Step AP3 is given in Shoup (1994b, §4.1).

Interestingly, Algorithm AP suggests a slightly faster algorithm for automorphism projection. Notice that the term $n^{(\omega+1)/2+1-\beta}$ in the running-time bound comes from Step AP3. Using the transposition principle and the strategy used to prove Theorem 3, we can reduce this term to $n^{(\omega+1)/2+(1-\beta)(\omega-1)/2}$ as follows.

Lemma 8 *Given $x^{q^t} \bmod f$, we can compute $\vec{u}^{\mathrm{T}} Q^{tj}$ for all j with $0 \leq j < m$, where $m = O(n)$, using $O(n^{(\omega+1)/2} m^{(\omega-1)/2})$ operations in $\mathbb{F}_q$.*

Proof. We use the same "doubling" strategy used in the algorithm in the proof of Lemma 4. Assume we have computed the row vectors

$$\vec{u}^{\mathrm{T}}, \vec{u}^{\mathrm{T}} Q^t, \ldots, \vec{u}^{\mathrm{T}} Q^{(k-1)t}, \tag{5}$$

as well as $x^{q^{kt}} \bmod f$ for some $k \geq 1$. Then we multiply each vector in the sequence (5) by Q^{kt} and compute $x^{q^{2kt}} \bmod f$. The problem of applying Q^{kt} to the sequence (5) is precisely the transpose of the problem solved by the algorithm in Lemma 3, and so by the transposition principle, now applied to a block diagonal matrix with Q^{kt} as diagonal blocks, we can do this in $O(n^{(\omega+1)/2}k^{(\omega-1)/2})$ arithmetic operations. Computing $x^{q^{2kt}} \bmod f$ from $x^{q^{kt}} \bmod f$ requires just one modular composition. That completes the description of the doubling step. The running time bound follows easily. □

Again, by the transposition principle, this implies the existence of an algorithm for the automorphism evaluation problem with the same complexity, although it is not entirely clear at the moment how to explicitly describe this algorithm.

Combining all of this with our previous analysis of Algorithm B, we have proved the following.

Theorem 5 *For any constant β with $0 \leq \beta \leq 1$, Algorithm B can be implemented so as to use an expected number of*

$$O(n^{(\omega+1)/2+(3-\omega)|\beta-1/2|+o(1)} + n^{(\omega+1)/2+(1-\beta)(\omega-1)/2+o(1)} + n^{1+\beta+o(1)} \log q) \quad (6)$$

operations in $\mathbb{F}_q$. In particular, choosing $\omega < 2.375477$ and minimizing the exponent of n, we get $O(n^{1.852} + n^{1.763} \log q)$ operations in $\mathbb{F}_q$.

Remark The first term in (6) is dominated by the second exactly when $\beta < 2/(5-\omega)$, and thus at least when $\beta < 2/3$.

For $\omega = 2.375477$, by making use of techniques for fast rectangular matrix multiplication, the operation count (6) in Theorem 5 can be reduced to

$$O(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2+o(1)} + n^{1+\beta+o(1)} \log q),$$

and in particular to $O(n^{1.815} \log q)$ for an appropriate choice of β. We indicate how this is done.

The first term in (6) arises from the rectangular $m \times n$ times $n \times t$ matrix multiplication in Step AP4. By the remark after Theorem 5, we may assume $\beta \geq 2/3$ and in particular that $t > m$.

Techniques for fast rectangular matrix multiplication allow us to multiply a $b^{\delta} \times b$ matrix by a $b \times b$ matrix with $O(b^{2+o(1)})$ operations for some $\delta > 0$

(Coppersmith 1982, Lotti & Romani 1983). With the construction yielding $\omega < 2.375477$ by Coppersmith and Winograd (1990), we may chose $\delta = 0.29$ (Coppersmith, private communication).

The needed $m \times n \times t$ matrix product is done with $O(n/t)$ products of $m \times t$ times $t \times t$ matrices. We shall carry out each of the latter products by multiplying a $(t/b) \times (t/b)$ block matrix with $(mb/t) \times b$ blocks times a $(t/b) \times (t/b)$ block matrix with $b \times b$ blocks. If $mb/t = b^\delta$, i.e., $b = (t/m)^{1/(1-\delta)}$, each block product costs $O(b^{2+o(1)})$ operations, yielding a total of $O((t/b)^\omega b^{2+o(1)})$ operations for the $m \times t \times t$ product. Substituting $m = O(n^{1-\beta})$ and $t = O(n^\beta)$, we get for the entire $m \times n \times t$ product

$$O(n^{1-\beta+\omega\beta-(\omega-2)(2\beta-1)/(1-\delta)+o(1)}) \tag{7}$$

operations.

Now, for $\omega = 2.375477$ and $\delta = 0.29$, one routinely checks that for $2/3 \leq \beta \leq 1$, the quantity (7) is dominated by either the second or the third term of (6).

4 Applications to normal bases

The results of §3 can be used to speed certain operations with so-called normal basis of finite extensions of $\mathbb{F}_q$. In this section we describe those subquadratic algorithms.

A finite field $\mathbb{F}_{q^n}$ of q^n elements can be represented as an n-dimensional vector space over $\mathbb{F}_q$. For instance, if $f(x) \in \mathbb{F}_q[x]$ is an irreducible monic polynomial of degree n over $\mathbb{F}_q$, the powers $1, x, \ldots, x^{n-1}$ form a basis for the Kronecker representation $\mathbb{F}_q[x]/(f(x))$ of the field $\mathbb{F}_{q^n}$. It can be advantageous for performing arithmetic in $\mathbb{F}_{q^n}$, in particular exponentiation, if one finds a normal element $\alpha \in \mathbb{F}_{q^n}$ with the property that

$$\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}}$$

is a $\mathbb{F}_q$-vector space basis for $\mathbb{F}_{q^n}$. Von zur Gathen and Giesbrecht (1990) give a randomized algorithm for finding a normal element $\alpha \in \mathbb{F}_q[x]/(f(x))$ in $O(n^{2+o(1)} \log q)$ arithmetic operations in $\mathbb{F}_q$. The running time of their algorithm is reduced in (von zur Gathen and Shoup 1992) to $O(n^{2+o(1)} + n^{1+o(1)} \log q)$ arithmetic operations in $\mathbb{F}_q$. Here we give $O(n^{1.815} \log q)$ solutions to the following three problems:

basis selection Given $f(x) \in \mathbb{F}_q[x]$ irreducible monic of degree n, compute a normal element $\alpha \in \mathbb{F}_q[x]/(f(x))$.

conversion to power basis coordinates Given f and α as above and $c_0, \ldots, c_{n-1} \in \mathbb{F}_q$, compute $c_0\alpha + \cdots + c_{n-1}\alpha^{q^{n-1}}$ in power basis representation.

conversion to normal coordinates Given f and α as above and $\gamma \in \mathbb{F}_q[x]/(f(x))$, compute $c_0, \ldots, c_{n-1} \in \mathbb{F}_q$ such that

$$c_0\alpha + \cdots + c_{n-1}\alpha^{q^{n-1}} = \gamma.$$

Theorem 6 *We have probabilistic algorithms that can solve the basis selection and conversion to and from power basis coordinates problems in*

$$O(n^{(\omega+1)/2+(1-\beta)(\omega-1)/2+o(1)} + n^{1+\beta+o(1)} \log q), \qquad (8)$$

arithmetic operations in $\mathbb{F}_q$ for any β with $0 \le \beta \le 1$.

Proof. Suppose $\vec{\alpha}$ is the column vector containing the coefficients of the canonical representative of α. Using the notation of §3 we have that $Q^i\vec{\alpha}$ is the coefficient vector of the canonical representative of α^{q^i}, where Q is the matrix representing the q-th power map on $\mathbb{F}_q[x]/(f(x))$. Hence α is normal if

$$\vec{\alpha}, Q\vec{\alpha}, Q^2\vec{\alpha}, \ldots, Q^{n-1}\vec{\alpha}$$

are linearly independent vectors. Since f is irreducible and the minimum polynomial of Q is $\lambda^n - 1$ such an α must exist. Furthermore, for a random row vector $\vec{u}^{\mathrm{T}}$ and for a random column vector $\vec{\alpha}$ the probability that the minimum linear generator of

$$\vec{u}^{\mathrm{T}}Q^i\vec{\alpha} \qquad (0 \le i)$$

remains $\lambda^n - 1$ is no less than $1/(12\max\{\log_q(n), 1\})$ (see Wiedemann 1986, Proposition 3, or Giesbrecht 1993, §6.1). Therefore, a normal element can be found with success probability no less than $1 - 1/e$ by running the automorphism projection algorithm of §3 $12\max\{\log_q(n), 1\}$ times. The stated complexity (8) then follows from our estimates at the end of §3.

Conversion to power basis coordinates is simply the automorphism evaluation problem of §3, so it remains to demonstrate conversion to normal basis coordinates in time (8). By first applying the q-th power map $n-1$ times to

$$\gamma = c_0\alpha + \cdots + c_{n-1}\alpha^{q^{n-1}}$$

and then applying a linear map u from $\mathbb{F}_{q^n}$ to $\mathbb{F}_q$ we obtain

$$u(\gamma^{q^j}) = \sum_{i=0}^{n-1} c_i u(\alpha^{q^{i+j}}) \quad (0 \le j < n). \tag{9}$$

If the linear map u preserves $\lambda^n - 1$ as the minimum linear generator for $u(\alpha^{q^i})$, where $i \ge 0$, then the Hankel matrix on the right side of (9) must be non-singular, because otherwise one could find a second linear generator of degree n. Such a u is a by-product of our basis selection method and can be found in a similar way if only α is given. The same is true for the entries $u(\alpha^{q^{i+j}})$ in the Hankel matrix, while the left side elements $u(\gamma^{q^j})$ are computed again by automorphism projection. The Hankel system is finally solved for the c_i in $O(n^{1+o(1)})$ arithmetic steps (Brent et al. 1980). □

5 Practical Algorithms

In this section, we describe how the methods developed in this paper can be used to obtain practical algorithms, without relying on fast matrix multiplication.

Consider our Fast Cantor/Zassenhaus algorithm. A practical variant of Algorithm D, the distinct-degree factorizer, runs as follows. In Step D1, we set $l \approx \sqrt{n/2}$, so $m \approx \sqrt{n/2}$ as well. We compute $x^q \bmod f$ via repeated squaring. We generate both the baby steps and the giant steps (Steps D1 and D2) by iteratively applying a modular composition algorithm. Steps D3, D4, and D5 are performed by carrying them out quite literally as they are described, without any "tricks."

The cost of each step is then as follows:

Steps D1 and D2: $\approx \sqrt{2n}$ modular compositions;

Step D3: $\approx n/2$ multiplications of polynomials modulo f;

Step D4: $\approx \sqrt{n/2}$ GCD's;

Step D5: $O(n^{3/4})$ polynomial divisions and the equivalent of at most $\approx \sqrt{n/2}$ GCD's.

To appreciate the practical significance of this, one must realize that, in spite of popular prejudice, asymptotically fast polynomial multiplication algorithms are in fact fast in practice, for quite reasonably sized n. This is demonstrated, for example, in Shoup (1994b).

Because of this, the dominant cost in Brent & Kung's modular composition algorithm is indeed the $\approx n^2$ scalar multiplications and additions, although the cost of the $O(n^{1/2})$ polynomial multiplications can not be entirely ignored.

The time spent taking GCD's can be reduced by using a fast "Half-GCD" algorithm (which is in practice faster than Euclid's algorithm, although not spectacularly so), and by "buffering" the GCD's, i.e., computing the product of several polynomials modulo f, and then taking just one GCD. With these techniques, the time spent taking GCD's can be made a small percentage of the total. Also the time spent is Step D5 is in practice a small percentage of the total.

Finally, we note that the space requirement of this algorithm is just $O(n^{1.5})$ elements in $\mathbb{F}_q$.

In summary, this variant of our distinct-degree factorizer uses

$$O(n^{2.5} + n^{1+o(1)} \log q)$$

operations in $\mathbb{F}_q$ and space for $O(n^{1.5})$ elements in $\mathbb{F}_q$. Moreover, both of the implied "big-O" constants are reasonably small.

Of course, in general, we may have to perform one or more equal-degree factorizations as well. The equal-degree factorization algorithm in von zur Gathen & Shoup (1992) can be implemented so as to use

$$O(n^2 \log n + n^{1+o(1)} \log q)$$

operations in $\mathbb{F}_q$ and space for $O(n^{1.5})$ elements in $\mathbb{F}_q$, where again the implied constants are reasonably small.

In Shoup (1994b), this factoring algorithm is developed in further detail, and an implementation as well as the results of empirical tests are described. That paper concludes that if q is a large prime, then this new algorithm allows much larger polynomials to be factored in a reasonable amount of time and space than was previously possible using other algorithms. As an example from that paper, a pseudo-random degree 128 polynomial was factored modulo a 128-bit prime on a SUN SPARC-station ELC, which is rated at about 20 MIPS. The running time was under 2 minutes. To put this in some context, for the same polynomial on the same machine, the built-in Maple factorizer (based on Cantor/Zassenhaus) required about 25 hours. As another example, a pseudo-random degree 1024 polynomial was factored modulo a 1024-bit prime in about 50 hours, using about 11 megabytes of memory.

It is also possible obtain a practical version of the Fast Black Box Berlekamp algorithm using similar techniques, although we have not as yet implemented this. That algorithm would require significantly fewer GCD calculations than the above algorithm, and would also avoid the occasional need to perform a large equal-degree factorization; however, it would require at least twice as many modular compositions as the above algorithm. Empirical analysis of the relative costs of these operations indicate that the disadvantages would outweigh the advantages of Fast Black Box Berlekamp over Fast Cantor/Zassenhaus. Moreover, a Fast Black Box Berlekamp algorithm would require about twice as much space. Therefore, at the moment, for practical purposes, Fast Cantor/Zassenhaus appears preferable to Fast Black Box Berlekamp.

References

Aho, A., Hopcroft, J., and Ullman, J., *The Design and Analysis of Algorithms*; Addison and Wesley, Reading, MA, 1974.

Baur, W. and Strassen, V., “The complexity of partial derivatives,” *Theoretical Comp. Sci.* **22**, pp. 317–330 (1983).

Ben-Or, M., “Probabilistic algorithms in finite fields,” *Proc. 22nd IEEE Symp. Foundations Comp. Sci.*, pp. 394–398 (1981).

Berlekamp, E. R., “Factoring polynomials over large finite fields,” *Math. Comp.* **24**, pp. 713–735 (1970).

Brent, R. P., Gustavson, F. G., and Yun, D. Y. Y., “Fast solution of Toeplitz systems of equations and computation of Padé approximants,” *J. Algorithms* **1**, pp. 259–295 (1980).

Brent, R. P. and Kung, H. T., “Fast algorithms for manipulating formal power series,” *J. ACM* **25**/4, pp. 581–595 (1978).

Canny, J., Kaltofen, E., and Lakshman Yagati, "Solving systems of non-linear polynomial equations faster," *Proc. ACM-SIGSAM 1989 Internat. Symp. Symbolic Algebraic Comput.*, pp. 121–128 (1989).

Cantor, D. G. and Kaltofen, E., "On fast multiplication of polynomials over arbitrary algebras," *Acta Inform.* **28**/7, pp. 693–701 (1991).

Cantor, D. G. and Zassenhaus, H., "A new algorithm for factoring polynomials over finite fields," *Math. Comp.* **36**, pp. 587–592 (1981).

Coppersmith, D., "Rapid multiplication of rectangular matrices," *SIAM J. Comput.* **11**/3, pp. 467–471 (1982).

Coppersmith, D. and Winograd, S., "Matrix multiplication via arithmetic progressions," *J. Symbolic Comput.* **9**/3, pp. 251–280 (1990).

Dornstetter, J. L., "On the equivalence between Berlekamp's and Euclid's algorithms," *IEEE Trans. Inf. Theory* IT-**33**/3, pp. 428–431 (1987).

Fiduccia, C. M., "On obtaining upper bounds on the complexity of matrix multiplication," in *Complexity of Computer Computations*, edited by R. E. Miller and J. W. Thatcher; Plenum Press, New York, pp. 31–40, 1972.

Fiduccia, C. M., "On the Algebraic Complexity of Matrix Multiplication," *Ph.D. Thesis*, Center Comput. Inform. Sci., Div. Engin., Brown Univ., Providence, Rhode Island, June 1973.

Fleischmann, P., "Connections between the algorithms of Berlekamp and Niederreiter for factoring polynomials over $\mathbb{F}_q$," *Linear Algebra and Applications* **192**, pp. 101–108 (1993).

von zur Gathen, J. and Giesbrecht, M., "Constructing normal bases in finite fields," *J. Symbolic Comput.* **10**/6, pp. 547–570 (1990).

von zur Gathen, J. and Shoup, V., "Computing Frobenius maps and factoring polynomials," *Comput. Complexity* **2**, pp. 187–224 (1992).

Giesbrecht, M., "Nearly optimal algorithms for canonical matrix forms," *Ph.D. Thesis*, Dept. Comput. Science, University of Toronto, Toronto, Canada, 1993.

Griewank, A., "Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation," *Optimization Methods & Software* **1**, pp. 35–54 (1992).

Kaltofen, E. and Lobo, A., "Factoring high-degree polynomials by the black box Berlekamp algorithm," in *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '94*, edited by J. von zur Gathen and M. Giesbrecht; ACM Press, New York, N. Y., pp. 90–98, 1994.

Kaltofen, E. and Pan, V., "Processor efficient parallel solution of linear systems over an abstract field," in *Proc. 3rd Ann. ACM Symp. Parallel Algor. Architecture*; ACM Press, pp. 180–191, 1991.

Kaminski, M., Kirkpatrick, D. G., and Bshouty, N. H., "Addition requirements for matrix and transposed matrix products," *J. Algorithms* **9**, pp. 354–364 (1988).

Knuth, D. E., *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Ed. 2*; Addison Wesley, Reading, MA, 1981.

Lidl, R. and Niederreiter, H., *Finite Fields*; Addison-Wesley, Reading, MA, 1983.

Linnainmaa, S., "Taylor expansion of the accumulated rounding error," *BIT* **16**, pp. 146–160 (1976).

Lotti, G. and Romani, F., "On the asymptotic complexity of rectangular matrix multiplication," *Theoretical Comput. Sci.* **23**, pp. 171–185 (1983).

Massey, J. L., "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory* IT-**15**, pp. 122–127 (1969).

Niederreiter, H., "A new efficient factorization algorithm for polynomials over small finite fields," *Applic. Algebra Engin., Commun. Comput.* **4**, pp. 81–87 (1993).

Niederreiter, H. and Göttfert, R., "Factorization of polynomials over finite fields and characteristic sequences," *J. Symbolic Comput.* **16**/5, pp. 401–412 (1994).

Ostrowski, G. M., Wolin, Ju. M., and Borisow, W. W., "Über die Berechnung von Ableitungen," *Wissenschaftliche Zeitschrift Techn. Hochsch. Chem. Leuna-Merseburg* **13**/4, pp. 382–384 (1971). In German.

Rabin, M. O., "Probabilistic algorithms in finite fields," *SIAM J. Comp.* **9**, pp. 273–280 (1980).

Schwarz, Št., "On the reducibility of polynomials over a finite field," *Quart. J. Math. Oxford Ser. (2)* **7**, pp. 110–124 (1956).

Shoup, V., "On the deterministic complexity of factoring polynomials over finite fields," *Inform. Process. Letters* **33**/5, pp. 261–267 (1990).

Shoup, V., "Fast construction of irreducible polynomials over finite fields," *J. Symbolic Comput.* **17**/5, pp. 371–391 (1994a).

Shoup, V., "A new polynomial factorization algorithm and its implementation," *Manuscript*, Univ. d. Saarlandes, Saarbrücken, Germany, August 1994b. To appear in *J. Symbolic Comput.*

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

Note added in Proof

Bürgisser et al. [N1] have traced the transposition principle discussed in §3.2 to Tellegen's theorem of control theory. By use of the new exponents for rectangular matrix multiplication [N2] the asymptotic complexity of modular polynomial composition and hence of our factorization algorithms can be lowered a little bit. For example, the $O(n^{1.815} \log q)$ running time of Theorem 1 can be lowered to $O(n^{1.8054} \log q)$. For large q and small characteristic p it is possible to improve the binary running time of the von zur Gathen/Shoup algorithm. In [N3] we show, for example, that for $q = 2^k$ with $k = \Omega(n^{1.46})$ one may factor a polynomial of degree n with coefficients in $\mathbb{F}_q$ in $O(n(\log q)^{1.67})$ fixed precision operations. Here the field $\mathbb{F}_q$ is represented as a polynomial residue ring, performing ring arithmetic modulo an irreducible polynomial of degree k with coefficients in $\mathbb{F}_2$. Binary running time, that is, fixed precision cost, includes the time for executing each field operation in $\mathbb{F}_q$.

Many of Kaltofen's publications are accessible through links in the online BIBTEX bibliography database at `http://www.math.ncsu.edu/~kaltofen/bibliography/`.

References

[N1] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory.* Springer Verlag, Heidelberg, Germany, 1997.

[N2] X. Huang and V. Pan. Fast rectangular matrix multiplications and improving parallel matrix computations. In M. Hitz and E. Kaltofen, editors, *Proc. Second Internat. Symp. Parallel Symbolic Comput. PASCO '97*, pages 11–23, New York, N. Y., 1997. ACM Press.

[N3] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In W. Küchlin, editor, *Proc. 1997 Internat. Symp. Symbolic Algebraic Comput. (ISSAC'97)*, pages 184–188, New York, N. Y., 1997. ACM Press.