FACTORING POLYNOMIALS USING FEWER RANDOM BITS

Eric Bach

Victor Shoup

Computer Sciences Technical Report #757

March 1988

Factoring Polynomials Using Fewer Random Bits

Eric Bach

Victor Shoup

Computer Sciences Department

University of Wisconsin–Madison

Madison, WI 53706

March 16, 1988

Abstract. Let F be a field of $q = p^n$ elements, where p is prime. We present two new probabilistic algorithms for factoring polynomials in $F[X]$ that make particularly efficient use of random bits. They are easy to implement, and require no randomness beyond an initial seed whose length is proportional to the input size. The first algorithm is based on a procedure of Berlekamp; on input f in $F[X]$ of degree d, it uses $d \log_2 p$ random bits and produces in polynomial time a complete factorization of f with a failure probability of no more than $1/p^{(1-\epsilon)\frac{1}{2}d}$. (Here ϵ denotes a fixed parameter between 0 and 1 that can be chosen by the implementor.) The second algorithm is based on a method of Cantor and Zassenhaus; it uses $d \log_2 q$ random bits and fails to find a complete factorization with probability no more than $1/q^{(1-\epsilon)\frac{1}{4}d}$. For both of these algorithms, the failure probability is exponentially small in the number of random bits used.

Submitted to Journal of Symbolic Computation.

This research was sponsored by the National Science Foundation, via grants DCR-8504485 and DCR-8552596.

1. Introduction

Let F be a finite field with q elements, where $q = p^n$ and p is prime. Consider the problem of factoring polynomials $f \in F[X]$ into irreducible factors. There are no known deterministic polynomial-time procedures for this problem, though there are efficient methods that use random numbers. We will present two algorithms that are easy to implement and make particularly efficient use of these random numbers. In particular, they require no randomness beyond an initial seed about as long as the input, and the probability of their not obtaining complete factorizations is exponentially small.

Berlekamp's factoring algorithm [Berlekamp] runs in time polynomial in $\deg f$ and q. Thus, for bounded values of q, this is a polynomial-time algorithm. If q is arbitrary, however, Berlekamp's algorithm requires an exhaustive search through the field F and this takes exponential time. To avoid this, various *probabilistic* polynomial-time factoring algorithms have been invented [Berlekamp, Cantor/Zassenhaus, Rabin, Ben-Or]. When endowed with the ability to flip coins, i.e. access to a source of independent, unbiased random bits, they are efficient and reliable.

In this paper we adopt the view that random bits are a scarce resource, and seek factoring algorithms that use this resource efficiently. There are two reasons for our point of view. First, a careful analysis of exactly where and how much randomness is required to solve a particular problem may produce valuable new insight into the problem. Second, whether polynomial-time computation benefits from randomness is an open question. One could conceivably resolve this in the negative by reducing the randomness requirement to zero. Our results may lead in this direction.

To make our view precise, we introduce a formal definition of random bit usage that was proposed in [Shoup].

First, we adopt the following idealization of probabilistic computation. On input x, an algorithm A computes its random bit requirement $b(x)$ (the time to compute $b(x)$ and its value are polynomially bounded in $|x|$, the length of x). A then is supplied with a random bit string of length $b(x)$, and after time bounded by a polynomial in $|x|$, it outputs an answer or the special symbol "?", signifying failure. We assume the failure probability $\epsilon(x)$ is less than a fixed constant (1/2, say), and that if A succeeds in producing an answer, it is correct. Following Babai, we call such an algorithm a *Las Vegas* algorithm (see [Johnson]).

The measure we shall use to estimate random bit usage is the function $h(x)$ defined by

$$h(x) = \frac{b(x)}{\log_2(1/\epsilon(x))}$$

if $\epsilon(x)$ is nonzero and zero otherwise. $h(x)$ is called the *half-cost* of A on input x; it measures the relationship between the random bit consumption and the failure probability in an invariant fashion. In particular, it does not change if we iterate algorithm A to reduce the failure probability. Intuitively, the half-cost measures the number of random bits required to cut the failure probability in half, provided we buy random bits "in bulk." Since $b(x)$ is polynomial in $|x|$, so is $h(x)$. If $\epsilon(x) > 0$, then $\epsilon(x) \geq 2^{-b(x)}$, so $h(x) \geq 1$. Therefore, the best non-zero half-cost we could hope for is $h(x) = O(1)$. If $h(x) \leq H$ for all x, then $\epsilon(x) \leq 2^{-b(x)/H}$, i.e., the failure probability is exponentially small in the number of random bits used.

Previously, one of the present authors described a constant half-cost Las Vegas algorithm for finding square roots modulo primes [Bach]. In this paper we extend this result and give two constant half-cost Las Vegas algorithms for polynomial factoring. The first is based on Berlekamp's algorithm and an extension described in [Cantor/Zassenhaus]. Given a polynomial f of degree d in $F[X]$, it produces a *complete* factorization of f using $d \log_2 p$ random bits, obtaining a failure probability bound of $1/p^{(1-\epsilon)\frac{1}{2}d}$. This implies a half-cost bound of $2/(1-\epsilon)$ (here ϵ, $0 < \epsilon < 1$, controls the tradeoff between running time and failure rate). The second algorithm is based on the distinct degree factorization method (a so-called "folk" method) and improvements to this method described in [Cantor/Zassenhaus]. It uses $d \log_2 q$ random bits, obtaining a failure probability bound of $1/q^{(1-\epsilon)\frac{1}{4}d}$. This implies a half-cost bound of $4/(1-\epsilon)$. We do not claim that the first algorithm is necessarily superior to the second in practice, since the half-cost bounds contain an arbitrary parameter ϵ and randomness may not be the only resource of interest.

Both of these algorithms are efficient and easy to implement. The usual approach searches for a "splitting polynomial" h for which $\gcd(F(h), f) \neq 1, f$ (F is an easily computable function) and tries many independently chosen values of h. Our approach is to choose a seed at random and then deterministically generate a simple sequence that with very high probability contains a splitting polynomial. In particular, no sophisticated pseudo-random number generator is required.

We describe these algorithms in sections 2 and 3, respectively. In section 4, we briefly discuss some implementation details.

2. Modifications of the Berlekamp Algorithm

Recall that F is a finite field with $q = p^n$ elements, where p is prime. We assume that we are given a concrete representation $F = \mathbb{Z}_p(\alpha)$, where α is a root of an irreducible polynomial over $\mathbb{Z}_p$ of degree n.

Let f be a polynomial of degree d in $F[X]$ that we wish to factor. The first step of Berlekamp's algorithm is to obtain a polynomial f^* that has the same irreducible factors as f, but with each factor occurring only once, i.e. f^* is squarefree. This is easily done (see, e.g., [Knuth], p. 421). So without loss of generality we will assume that the polynomial f is squarefree to begin with. Let $f = f_1 \cdots f_r$ be the complete factorization of f.

F contains $\mathbb{Z}_p$ as a subfield. Furthermore, F is an n-dimensional $\mathbb{Z}_p$-vector space with $1, \alpha, \ldots, \alpha^{n-1}$ forming a basis. The ring $F[X]/(f)$ is an F-vector space of dimension d with the residues mod f of $1, \ldots, X^{d-1}$ forming a basis. Therefore, $F[X]/(f)$ is an nd-dimensional $\mathbb{Z}_p$-vector space with the residues mod f of $\alpha^i X^j$ $(i = 0, \ldots, n-1; j = 0, \ldots, d-1)$ forming a basis. Throughout the rest of this section, we purposefully blur the distinction between a polynomial and its residue mod f.

Now, by the Chinese Remainder Theorem, we have an isomorphism

$$F[X]/(f) \cong F[X]/(f_1) \times \cdots \times F[X]/(f_r).$$

For simplicity, we identify $F[X]/(f)$ with its isomorphic image, and call the residues

$$(a_1, a_2, \ldots, a_r)$$

the *components* of f. Since f_i is irreducible, $F[X]/(f_i)$ is a field containing $\mathbb{Z}_p$, the fixed field of the Frobenius automorphism $x \longmapsto x^p$ on $F[X]/(f_i)$. Thus, $R = \prod_{i=1}^{r} \mathbb{Z}_p$ is an r-dimensional $\mathbb{Z}_p$-vector space, and is the kernel of the $\mathbb{Z}_p$-linear map on $F[X]/(f)$ given by $x \longmapsto x^p - x$. Since we have an explicit basis for $F[X]/(f)$ over $\mathbb{Z}_p$, we can construct the matrix of this linear map, and by Gaussian elimination find a basis $\omega^{(1)}, \ldots, \omega^{(r)}$ for R over $\mathbb{Z}_p$.

If p is small, we can quickly factor f in the following manner. First, we compute

$$h_{ij} = \omega^{(i)} + j \quad (i = 1, \ldots, r; j = 0, \ldots, p-1).$$

Then, we initialize a set $S = \{f\}$, and iteratively refine it as follows: for each h_{ij} we consider each $\hat{f}$ in S in turn, and compute $\gcd(h_{ij}, \hat{f})$; if this is a proper divisor v of $\hat{f}$, we replace $\hat{f}$ in S by v and $\hat{f}/v$. Any pair of factors f_s, f_t must be separated by this procedure, since there must be some basis element

$$\omega^{(i)} = (w_1, \ldots, w_r) \quad (w_i \in \mathbb{Z}_p)$$

for which $w_s \neq w_t$ (otherwise, all elements of R would agree in their s and t components, which is not the case). For this $\omega^{(i)}$,

$$\omega^{(i)} - w_s \equiv 0 \pmod{f_s}$$

$$\omega^{(i)} - w_s \not\equiv 0 \pmod{f_t}.$$

Thus, $\omega^{(i)} - w_s$ will serve to separate f_s and f_t.

For large p, we can use a probabilistic method described in [Cantor/Zassenhaus] that goes as follows. Choose a random element h of R (this is easy as we have a $\mathbb{Z}_p$-basis for R). In component notation

$$h = (a_1, \ldots, a_r) \quad (a_i \in \mathbb{Z}_p).$$

Then compute

$$u = h^{(p-1)/2} = (a_1^{(p-1)/2}, \ldots, a_r^{(p-1)/2})$$

(note p is odd) and try to split f with $u - 1$ and u. Let χ be the quadratic character on $\mathbb{Z}_p$. The probability that neither $\gcd(u-1, f)$ nor $\gcd(u, f)$ split f is the probability that $\chi(a_1) = \cdots = \chi(a_r)$, since each component of u is 0, 1, or -1. This probability is asymptotic to $1/2^{r-1}$ as $p \to \infty$. Using this method, we can construct a Las Vegas algorithm to completely factor f, but it will not have a constant half-cost. Indeed, if $f = f_1 f_2$, the probability that u splits f is about $1/2$, leading to a half-cost near $2\log_2 p$.

We now give a modification of this algorithm that *does* have a constant half-cost for sufficiently large p.

Algorithm 2.1. Input: f. Output: The set of irreducible factors of f.

(1) Construct the basis $\omega^{(1)}, \ldots, \omega^{(r)}$ for R over $\mathbb{Z}_p$ as described in paragraph 3 of this section.

(2) If $r = 1$ then output $\{f\}$ and quit; otherwise, let $k = \lceil \frac{1}{2}\log_2 p \rceil$ and $S = \{f\}$. Repeat steps 3a-3d d times.

(3a) Choose a random $x \in \mathbb{Z}_p$.

(3b) Let $h_{ij} = x\omega^{(i)} + j \quad (i = 1, \ldots, r; j = 0, \ldots, k)$.

(3c) For each h_{ij}, compute $u_{ij} = h_{ij}^{(p-1)/2}$.

(3d) For each u_{ij}, consider each $\hat{f}$ in S, and compute $\gcd(u_{ij}, \hat{f})$ and $\gcd(u_{ij} - 1, \hat{f})$. If either of these is a proper divisor v of $\hat{f}$, then replace $\hat{f}$ in S by v and $\hat{f}/v$.

(4) If $|S| = r$, then output S; otherwise, report failure.

We now analyze the failure probability and half-cost of this algorithm. For fixed s and t, $1 \leq s < t \leq r$, let P_{st} be the probability that one iteration of step 3 of algorithm 2.1 fails to separate f_s and f_t. We state the principal result of this section.

Theorem 2.2. We have

$$P_{st} \leq \frac{\log_2 p}{p^{1/2}}.$$

From this a half-cost bound (Corollary 2.5, below) easily follows. Before proving this we will need two lemmas, the first of which is proved in [Schmidt].

Lemma 2.3. Let χ be a multiplicative character of order $d > 1$ on a finite field K with q elements. Suppose that $g(X) \in K[X]$ has m distinct roots (in the algebraic closure of K) and is not a d-th power, i.e. not of the type $g(X) = c(h(X))^d$, where $c \in K$, and $h(X) \in K[X]$. Then

$$\left|\sum_{x \in K} \chi(g(x))\right| \leq (m-1)q^{1/2}.$$

Lemma 2.4. If β, γ are distinct, nonzero elements of F, then none of the polynomials

$$\prod_{1 \leq j \leq k} (X - j\beta)^{e_j}(X - j\gamma)^{e_j} \qquad (0 \leq e_j \leq 1, \text{not all } e_j \text{ zero})$$

are squares.

Proof. Since β and γ are distinct, if such a polynomial were a square, then for distinct $j_1, \ldots, j_w$ between 1 and k,

$$\begin{aligned} j_1\beta &= j_2\gamma \\ j_2\beta &= j_3\gamma \\ &\vdots \\ j_{w-1}\beta &= j_w\gamma \\ j_w\beta &= j_1\gamma. \end{aligned}$$

But this implies that $\beta \sum j_\nu = \gamma \sum j_\nu$, so either $\beta = \gamma$ or $\sum j_\nu = 0$. The first is impossible, and the last cannot be true because

$$0 < \sum_{\nu=1}^{w} j_\nu \leq \sum_{j=1}^{k} j = k(k+1)/2 < p.$$

The last inequality follows from the fact that $k = \lceil \frac{1}{2} \log_2 p \rceil$. ■

Proof of Theorem 2.2. Without loss of generality assume that $s = 1, t = 2$. There must be some basis element $\omega^{(i)} = (a, b, \ldots)$ for which $a \neq b$. If either a or b are zero, then h_{i0} will separate f_1 and f_2, provided $x \neq 0$. Otherwise, if $\chi(a) \neq \chi(b)$, where χ is the quadratic character, $h_{i0} - 1$ will separate f_1 and f_1, again, provided $x \neq 0$. In either case, we have $P_{12} \leq 1/p$, so we can assume that a and b are nonzero and that $\chi(ab) = 1$.

For any $1 \leq j \leq k$, we cannot have $0 = ax + j = bx + j$, and if either $ax + j$ or $bx + j$ are zero, then h_{ij} will separate f_1 and f_2. So if we fail to separate f_1 and f_2, there must exist nonzero $y_1, \ldots, y_k$ in $\mathbb{Z}_p$ such that

$$\begin{aligned}(ax+1)(bx+1) &= y_1^2\\ (ax+2)(bx+2) &= y_2^2\\ &\vdots\\ (ax+k)(bx+k) &= y_k^2.\end{aligned}$$

Since $ab = c^2$, we can set $z_i = y_i/c$ and rewrite this system of equations as

$$\begin{aligned}(x+1/a)(x+1/b) &= z_1^2\\ (x+2/a)(x+2/b) &= z_2^2\\ &\vdots\\ (x+k/a)(x+k/b) &= z_k^2.\end{aligned} \tag{$*$}$$

Let $N = |\{(x, z_1, \ldots, z_k) \in \mathbb{Z}_p^{k+1} : (x, z_1, \ldots, z_k) \text{ satisfies } (*)\}|$. We want to get a good upper bound on N. Now, for fixed c, the number of z satisfying $z^2 = c$ is $1 + \chi(c)$. Therefore,

$$\begin{aligned}N &= \sum_{x\in\mathbb{Z}_p} \prod_{j=1}^{k} (1 + \chi((x+j/a)(x+j/b)))\\ &= \sum_{0\leq e_1,\ldots,e_k\leq 1} \sum_{x\in\mathbb{Z}_p} \chi\Big(\prod_{j=1}^{k} (x+j/a)^{e_j}(x+j/b)^{e_j}\Big).\end{aligned}$$

In this last expression, the term corresponding to $e_1 = \cdots = e_k = 0$ is p. For the other terms, we can get an upper bound on the magnitude of the character sums appearing therein using lemmas 2.3 and 2.4. These lemmas imply

$$\begin{aligned}N &\leq p + p^{1/2} \sum_{l=1}^{k} \binom{k}{l} (2l-1)\\ &= p + p^{1/2}(2^k(k-1)+1).\end{aligned}$$

We divide this by 2^k to obtain a bound on the number of x for which there exist nonzero $z_1, \ldots, z_k$ satisfying $(*)$, and then divide by p to obtain a bound on the probability P_{12}. This produces

$$P_{12} \leq 1/2^k + (k-1)/p^{1/2} + 1/(2^k p^{1/2}).$$

The right hand side of this inequality is asymptotic to $\frac{1}{2}\log_2 p/p^{1/2}$ as $p \to \infty$, and some calculations show that it is less than $\log_2 p/p^{1/2}$ for all $p \geq 3$. So the theorem is proved. ■

Corollary 2.5. The failure probability of algorithm 2.1 is bounded by

$$\left(\frac{3^{4/3}(\log_2 p)^2}{p}\right)^{\frac{1}{2}d} < 1/p^{(1-\epsilon)\frac{1}{2}d}.$$

(the inequality holds for any $\epsilon > 0$ provided p is sufficiently large).

Proof. Since the iterations of step 3 are independent, the probability of not separating any fixed pair of factors is at most

$$\left(\frac{(\log_2 p)^2}{p}\right)^{\frac{1}{2}d}.$$

Summing over all pairs, we have a failure probability bound of

$$\begin{aligned}\binom{r}{2}\left(\frac{(\log_2 p)^2}{p}\right)^{\frac{1}{2}d} &\leq d^2\left(\frac{(\log_2 p)^2}{p}\right)^{\frac{1}{2}d}\\ &= \left(\frac{d^{4/d}(\log_2 p)^2}{p}\right)^{\frac{1}{2}d}\\ &\leq \left(\frac{3^{4/3}(\log_2 p)^2}{p}\right)^{\frac{1}{2}d}.\end{aligned}$$

The last inequality follows from the fact that $d^{1/d}$ is maximized at $d = 3$.

This gives the first bound. To get the second, note that for any $\epsilon > 0$ and p sufficiently large, we have $3^{4/3}(\log_2 p)^2 < p^\epsilon$. ■

The number of random bits used by algorithm 2.1 is essentially $d\log_2 p$, so Corollary 2.5 implies that the half-cost is no more than $2/(1-\epsilon)$, since for small p, we can resort to the deterministic version of Berlekamp's algorithm. This bound is valid even using the approximate uniform distribution described in section 4.

3. Modifications of the Cantor/Zassenhaus Algorithm

Again, F is a finite field with $q = p^n$ elements, and we want to factor $f \in F[X]$ of degree d. As in section 2, we assume that f is square free. The first step of the Cantor/Zassenhaus algorithm is to perform "distinct degree factorization," that is, obtain factors $f^{(1)}, \ldots, f^{(m)}$ of f such that (1) each $f^{(i)}$ is the product of r_i distinct irreducible polynomials of degree e_i, and (2) each irreducible factor of f appears in some $f^{(i)}$. This can be done in polynomial time by using the fact that $X^{q^a} - X$ is the product of all monic irreducible polynomials whose degrees divide a. See [Cantor/Zassenhaus] for details.

In this section, our analysis will involve several rings and fields, and it will be important to be explicit about the algebraic structures and homomorphisms under discussion. In particular, for $g, h \in F[X]$, we will use the notation $[g]_h$ to denote the image of g in $F[X]/(h)$.

We first discuss factoring $f = f_1 \cdots f_r$ where the f_i's are distinct irreducible polynomials of degree e. We consider the case where p is odd in detail, and then sketch the differences for the case where $p = 2$.

By the Chinese Remainder Theorem, we have an isomorphism

$$F[X]/(f) \cong F[X]/(f_1) \times \cdots \times F[X]/(f_r).$$

Let $Q = q^e$. Each $F[X]/(f_i)$ is a field with Q elements containing F as a subfield. Fix some arbitrary field E of size Q that contains F. Then there is an isomorphism σ_i of $F[X]/(f_i)$ onto E that fixes F. Thus, we have an isomorphism of $F[X]/(f)$ onto E^r that sends $[h]_f$ to $(\sigma_1([h]_{f_1}), \ldots, \sigma_r([h]_{f_r}))$. We identify $F[X]/(f)$ with its isomorphic image.

The Cantor/Zassenhaus algorithm first chooses a polynomial h at random of degree $< re$. In component notation this will be

$$[h]_f = (a_1, \ldots, a_r)$$

where a_i is a randomly selected element of E. We compute $u = h^{(Q-1)/2} \bmod f$. Then

$$[u]_f = (a_1^{(Q-1)/2}, \ldots, a_r^{(Q-1)/2}).$$

Let χ be the quadratic character on E. The probability that neither $\gcd(u-1, f)$ nor $\gcd(u, f)$ split f is the probability that $\chi(a_1) = \cdots = \chi(a_r)$, which is asymptotic to $1/2^{r-1}$. Using this approach, we can construct a Las Vegas algorithm for factoring f, but it will certainly not have a constant half-cost.

We now give a new algorithm for factoring a polynomial f of this restricted type. Along with f, it is given as input an "iteration parameter" λ. We assume a canonical enumeration $C_1, C_2, \ldots$ of polynomials over F of degree $< e$.

Algorithm 3.1. Input: f, λ. Output: the set of irreducible factors of f.

```
(1) Let $k = \lceil \log_2 Q \rceil$, and $S = \{f\}$.
(2) Repeat steps 3a-3c $\lambda$ times.
(3a) Generate a random polynomial $h$ in $F[X]$ of degree $< 2e$.
(3b) Compute $u_i = (h + C_i)^{(Q-1)/2} \bmod f \quad (i = 1, \ldots, k)$.
(3c) For each $u_i$, consider each $\hat{f}$ in $S$, and compute $\gcd(u_i, \hat{f})$ and $\gcd(u_i - 1, \hat{f})$. If either of these
     is a proper divisor $v$ of $\hat{f}$, then replace $\hat{f}$ in $S$ by $v$ and $\hat{f}/v$.
(4) If $|S| = r$, then output $S$; otherwise, report failure.
```

We want to get a bound on the probability that algorithm 3.1 fails to completely factor f. For fixed s and t, $1 \leq s < t \leq r$, let P_{st} denote the probability that one iteration of step 3 fails to separate f_s and f_t.

Theorem 3.2. We have

$$P_{st} \leq \frac{(\log_2 Q)^2}{Q}.$$

Proof. Without loss of generality assume that $s = 1, t = 2$. We have an isomorphism of $F[X]/(f_1 f_2)$ onto E^2 which sends $[h]_{f_1 f_2}$ to $(\sigma_1([h]_{f_1}), \sigma_2([h]_{f_2}))$. We identify $F[X]/(f_1 f_2)$ with its isomorphic image. As h ranges over all polynomials of degree $< 2e$, it ranges over a complete residue system mod $f_1 f_2$. So if h is chosen at random, then in component notation, $[h]_{f_1 f_2} = (x, y)$, where x and y are randomly chosen elements of E.

For $i = 1, \ldots, k$, let $a_i = \sigma_1([C_i]_{f_1})$ and $b_i = \sigma_2([C_i]_{f_2})$. The a_i's are distinct, and so are the b_i's. Now, $u_i \equiv (h + C_i)^{(Q-1)/2} \pmod{f}$, and so this congruence holds mod $f_1 f_2$ as well. Therefore, in component notation

$$[u_i]_{f_1 f_2} = ((x + a_i)^{(Q-1)/2}, (y + b_i)^{(Q-1)/2}).$$

Thus, P_{12} is no more than the probability that for $i = 1, \ldots, k$, $\chi(x + a_i) = \chi(y + b_i)$, where we choose $x, y \in E$ at random. Let P'_{12} be the probability that for $i = 1, \ldots, k$, $\chi(x + a_i) = \chi(y + b_i) \neq 0$. Then $P_{12} \leq k/Q^2 + P'_{12}$. Now, P'_{12} is the probability that there exist nonzero $z_1, \ldots, z_k$ in E such that

$$\begin{aligned}(x+a_1)(y+b_1) &= z_1^2\\ (x+a_2)(y+b_2) &= z_2^2\\ &\vdots\\ (x+a_k)(y+b_k) &= z_k^2.\end{aligned} \tag{$*$}$$

Let $N = \left|\{(x,y,z_1,\ldots,z_k) \in E^{k+2} : (x,y,z_1,\ldots,z_k) \text{ satisfies } (*)\}\right|$. We want to get a good upper bound on N. We have

$$\begin{aligned}N &= \sum_{x,y\in E} (1+\chi((x+a_1)(y+b_1)))\cdots(1+\chi((x+a_k)(y+b_k)))\\ &= \sum_{0\le i_1,\ldots,i_k\le 1}\ \sum_{x,y\in E} \chi((x+a_1)^{i_1}(y+b_1)^{i_1}\cdots(x+a_k)^{i_k}(y+b_k)^{i_k})\\ &= \sum_{0\le i_1,\ldots,i_k\le 1} \Big(\sum_{x\in E}\chi((x+a_1)^{i_1}\cdots(x+a_k)^{i_k})\Big)\Big(\sum_{y\in E}\chi((y+b_1)^{i_1}\cdots(y+b_k)^{i_k})\Big).\end{aligned}$$

In this last expression, the term corresponding to $i_1 = \cdots = i_k = 0$ is Q^2. We can use lemma 2.3 to bound the magnitude of each of the other terms, obtaining

$$\begin{aligned}N &\le Q^2 + Q\sum_{l=1}^{k}\binom{k}{l}(l-1)^2\\ &= Q^2 + Q\left(k(k-1)2^{k-2} - k2^{k-1} + 2^k - 1\right).\end{aligned}$$

We divide this quantity by 2^k to obtain a bound on the number of x, y for which there exist nonzero $z_1,\ldots,z_k$ satisfying $(*)$, and by Q^2 to obtain a bound on P'_{12}. Using the fact that $P_{12} \le k/Q^2 + P'_{12}$, we have

$$P_{12} \le \frac{1}{Q}\left(\frac{k}{Q} + \frac{Q}{2^k} + \frac{k(k-1)}{4} - \frac{k}{2} + 1 - \frac{1}{2^k}\right).$$

The right hand side of this inequality is asymptotic to $\frac{1}{4}(\log_2 Q)^2/Q$ as $Q \to \infty$, and some calculations show that it is less than $(\log_2 Q)^2/Q$ for all $Q \ge 3$. So the theorem is proved. ■

We now consider the case $q = 2^n$. We will assume that F contains a primitive cube root of unity ω. If this is not the case, we can construct the quadratic extension of F using the polynomial $\omega^2 + \omega + 1$, factor f in this extension field, and then, if necessary, multiply conjugates together to obtain the factorization of f over the original field.

We now modify algorithm 3.1 to handle this case. Step (1) is changed so that $k = \lceil \log_3 Q \rceil$. Step (3b) is changed so that

$$u_i = (h + C_i)^{(Q-1)/3} \bmod f.$$

In step (3c), we compute $\gcd(u_i, \hat{f})$, $\gcd(u_i + 1, \hat{f})$ and $\gcd(u_i + \omega, \hat{f})$.

We can prove an analog of theorem 3.2 for this modified algorithm. Let $\hat{P}_{st}$ be the probability that one iteration of step 3 of this modified algorithm fails to separate f_s and f_t.

Theorem 3.3. We have

$$\hat{P}_{st} \leq \frac{(\log_3 Q)^2}{Q}.$$

Sketch of Proof. This is very similar to the proof of theorem 3.2. The relevant system of equations is

$$(x + a_i)(y + b_i)^2 = z_i^3 \quad (i = 1, \ldots k).$$

We can estimate the number of solutions by using the fact that for fixed $c \in E$, the number of $z \in E$ satisfying $z^3 = c$ is $1 + \chi(c) + \chi(c^2)$, where χ is a multiplicative character of order 3 on E. ∎

Using algorithm 3.1, we now describe an algorithm for factoring any polynomial f in $F[X]$ of degree d. The basic idea is to perform distinct degree factorization, and then apply algorithm 3.1 to each of the resulting factors.

Algorithm 3.4. Input: f. Output: the set of irreducible factors of f.

(1) Perform distinct degree factorization, obtaining $f^{(1)}, \ldots, f^{(m)}$, where $f^{(i)}$ has r_i factors, each of degree e_i. Initialize $S = \{\}$. Generate a list ρ of d random elements from F.

(2) For $i = 1, \ldots, m$, do the following. If $r_i = 1$, then add $f^{(i)}$ to S. Otherwise, let $\lambda_i = \lfloor d/(2e_i) \rfloor$. Run algorithm 3.1 on $f^{(i)}$ with $\lambda = \lambda_i$, using ρ as the source of random field elements. If this succeeds in completely factoring $f^{(i)}$, then add the set of factors to S; otherwise, report failure.

(3) Output S.

To analyze this algorithm, we will ignore those $f^{(i)}$ with $r_i = 1$, since they are completely factored already. Therefore, to simplify the discussion, we will assume that $r_i \geq 2$ for each $i = 1, \ldots, m$. We also assume that $e_1 < e_2 < \cdots < e_m$.

Lemma 3.5. Let $0 < \epsilon < 1$ be a constant. Then there exists another constant C, depending on ϵ, such that for $i = 1, \ldots, m$, the probability that algorithm 3.4 fails to completely factor $f^{(i)}$ is less than

$$\frac{1}{q^{(1-\epsilon)\frac{1}{4}d}},$$

provided $q^{e_i} > C$.

Proof. The probability that $f^{(i)}$ is not completely factored is no more than the sum over all s and t, $1 \leq s < t \leq r_i$, of P_{st}. By theorems 3.2 and 3.3, this is at most

$$r_i^2 \left(\frac{(\log_2 Q_i)^2}{Q_i} \right)^{\lambda_i},$$

where $Q_i = q^{e_i}$ (note $\log_3 Q_i < \log_2 Q_i$). Since $\lambda_i \geq \lfloor r_i/2 \rfloor \geq (r_i - 1)/2$, we can bound this by

$$\left(\frac{r_i^{4/(r_i-1)}(\log_2 Q_i)^2}{Q_i} \right)^{\lambda_i} \leq \left(\frac{9(\log_2 Q_i)^2}{Q_i} \right)^{\lambda_i}.$$

Now, let $0 < \epsilon < 1$ be a constant. Then there exists a constant C such that $Q_i > C$ implies $9(\log_2 Q_i)^2 \leq Q_i^{\epsilon}$. In this case, the failure probability is no more than $1/q^{(1-\epsilon)e_i\lambda_i}$. We have $d = (2e_i)\lambda_i + \kappa_i$, where $0 \leq \kappa_i < 2e_i$. Since $d \geq 2e_i$, it follows that $(2e_i)\lambda_i > d/2$, and hence $e_i\lambda_i > d/4$. The lemma follows immediately. ∎

For fixed ϵ, we modify algorithm 3.4 as follows. If $q^{e_i} \leq C$, then we factor $f^{(i)}$ deterministically by brute force examination of all monic polynomials of degree e_i, of which there are no more than C. Otherwise, we use algorithm 3.1. The failure probability bound in lemma 3.5 now holds unconditionally. Note that this bound holds even using the approximate distribution for ρ described in section 4.

Theorem 3.6. Let $0 < \delta < 1$ be a constant. Then there exists another constant D, depending on δ, such that the failure probability of algorithm 3.4 is less than

$$\frac{1}{q^{(1-\delta)(1-\epsilon)\frac{1}{4}d}},$$

provided $q^{e_m} > D$.

Proof. Summing over all $i = 1, \ldots, m$, we see that the probability of failing to factor some $f^{(i)}$ is at most $m/q^{(1-\epsilon)\frac{1}{4}d}$. For fixed $0 < \delta < 1$, we want to bound this quantity by $1/q^{(1-\delta)(1-\epsilon)\frac{1}{4}d}$. Suppose this bound fails. Then we have $q^d < m^\gamma$, where $\gamma = 4/(\delta(1-\epsilon))$. We know that $d \geq 2e_1 + \cdots + 2e_m \geq m(m-1) + 2e_m$. So we have $q^{e_m} < (m^\gamma/q^{m(m-1)})^{1/2}$. Now, the quantity on the right hand side goes to zero as $m \to \infty$, and so it is bounded by some constant D. ∎

We modify algorithm 3.4 so that if $q^{e_m} \leq D$, then we factor f by brute force. The failure probability bound in lemma 3.6 now holds unconditionally. The number of random bits used by algorithm 3.4 is essentially $d \log_2 q$, implying a half-cost of no more than $4/((1-\delta)(1-\epsilon))$.

4. Implementation Details

Throughout our discussion, we have tacitly assumed that by using about $e \log_2 p$ random bits, we can generate a random list of numbers

$$x = (x_1, \dots, x_e) \quad (0 \le x_i < p)$$

with a uniform distribution. We now justify this assumption.

We first point out that the "standard" method of generating a random number between 0 and p does not work. This method is to generate a random number using $\lceil \log_2 p \rceil$ random bits, and then throw this number away if it is too large. This process is repeated until a number in range is generated. This method uses far too many random bits for our purposes.

We now describe a method that does work. Let $b = \lceil \log_2 p^k \rceil$. Generate b random bits, and view the result as a number $0 \le y < 2^b$. Compute $z = y \bmod p^e$. Note that $p^e \le 2^b < 2p^e$. So for any $0 \le z_0 < p^e$, the number of $0 \le y_0 < 2^b$ such that $z_0 = y_0 \bmod p^e$ is at least 1 and at most 2. Therefore, for any z_0, $\Pr[z = z_0] \le 2/2^b \le 2/p^e$. Furthermore, $\Pr[z = z_0] \ge 1/2^b > 1/(2p^e)$.

Now, compute $x = (x_1, \dots, x_e)$ where the x_i's are the digits of z when z is written in base p. This gives a one-one correspondence between the possible values of z and the possible values of x. Therefore, for any e-tuple x_0, we have

$$\frac{1}{2}\left(\frac{1}{p^e}\right) < \Pr[x = x_0] \le 2\left(\frac{1}{p^e}\right).$$

Suppose that an algorithm has a failure probability of α assuming a true uniform distribution. Let α' be the failure probability assuming our approximate uniform distribution. Let S be the set of e-tuples which cause failure. Then $\alpha' = \sum_{x_0 \in S} \Pr[x = x_0] \le \sum_{x_0 \in S} 2/p^e = 2\alpha$. In all of our examples, α is sufficiently small that this increase by a factor of 2 is negligible.

5. References

[Bach] E. Bach, "Realistic analysis of some randomized algorithms," *ACM Symposium on the Theory of Computing*, 1987, pp. 453-461.

[Ben-Or] M. Ben-Or, "Probabilistic algorithms in finite fields," *IEEE Symposium on Foundations of Computer Science*, 1981, pp. 394-398.

[Berlekamp] E. Berlekamp, "Factoring polynomials over large finite fields," *Mathematics of Computation*, Vol. 24, 1970, pp. 713-735.

[Cantor/Zassenhaus] D. Cantor and H. Zassenhaus, "A new algorithm for factoring polynomials over finite fields," *Mathematics of Computation,* Vol. 36, No. 154, April 1981, pp. 587-592.

[Johnson] D. Johnson, "The *NP*-completeness column: an ongoing guide," *Journal of Algorithms,* Vol. 5, 1984, pp. 433-447.

[Knuth] D. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms,* Addison-Wesley, second edition (1981).

[Rabin] M. Rabin, "Probabilistic algorithms in finite fields," *SIAM Journal on Computing,* Vol. 9, No. 2, May 1980, pp. 273-280.

[Schmidt] W. Schmidt, *Equations over Finite Fields,* Springer-Verlag, Lecture Notes in Mathematics 536 (1976).

[Shoup] V. Shoup, "Finding witnesses using fewer random bits," University of Wisconsin–Madison, Computer Sciences Technical Report #725.