

Computational Mathematics: Handout 12

Curtis Bright

March 7, 2024

1 Factoring Polynomials

In this handout we cover how to factor polynomials—in particular, we give a polynomial time algorithm for factoring polynomials whose coefficients are in a finite field. That is, given an $f \in \mathbb{F}_q[x]$, write it as the decomposition

$$f = f_1 \cdot f_2 \cdots f_n$$

where f_1, \dots, f_n are irreducible polynomials. An *irreducible* polynomial is one that cannot be written as a nontrivial product, i.e., f is irreducible exactly when $f = gh$ implies at least one of g and h are a constant polynomial. Because the coefficients are from a field, all of the leading coefficients of the polynomials f_i are invertible. Thus, one can write the decomposition as

$$f = cf_1 \cdot f_2 \cdots f_n \quad (*)$$

where $c \in \mathbb{F}_q$ and all the f_i are *monic* (have a leading coefficient of 1). We consider the factorization problem as finding the decomposition $(*)$ given f . By dividing both sides by c we also make the leading coefficient on the left-hand side also 1. Thus, without loss of generality we will assume that the f to factor is monic.

Throughout this handout the reader can think of \mathbb{F}_q as being \mathbb{Z}_p (i.e., the field of residues modulo a prime p), as \mathbb{Z}_p is the simplest kind of finite field. However, the algorithms we discuss will also work in general finite fields \mathbb{F}_q .

1.1 Roadmap

We will break the problem of factoring $f \in \mathbb{F}_q[x]$ into various special cases. One important special case is to handle the case when f is *squarefree*, meaning that in the decomposition $f = \prod_i f_i$ all of the f_i are distinct. That is, a polynomial is squarefree when it is not divisible by the square any polynomial (except constant polynomials). For example, $f = (x+1)^2$ is not squarefree and $f = (x+1)(x-1)$ is squarefree over any field \mathbb{Z}_p with $p > 2$.

A second special case of the factoring problem is to f into *distinct-degree factors*, i.e., write f as $f = g_1 \cdots g_n$ where g_i is the product of all irreducible factors of degree i . For example, if $f = (x+1)^2(x-1)(x^2+3)(x^3+x+1) \in \mathbb{F}_5[x]$ then $g_1 = (x+1)^2(x-1)$, $g_2 = x^2+3$, $g_3 = x^3+x+1$, and $g_4 = \cdots = g_8 = 1$.

The third special case is to factor the g_i that appear in the distinct-degree factorization. That is, given a $g_i \in \mathbb{F}_q[x]$ of degree d whose irreducible factors are all guaranteed to be of a known degree i (and hence there must be d/i of them), find all d/i irreducible factors $g_i = h_1 \cdots h_{d/i}$. For example, given $g_1 = x^3 + x^2 - x - 1 \in \mathbb{F}_5[x]$ (which is a product of only linear factors), find its decomposition into linear factors $g_1 = (x+1)^2(x-1)$.

1.2 Distinct-degree Factorization

First, we consider the problem of taking a squarefree monic $f \in \mathbb{F}_q[x]$ of degree n and writing it as the product $g_1 \cdots g_n$ where g_i is the product of the irreducible polynomials of degree i dividing f . Our algorithm for doing this will be based on a generalization of Fermat's little theorem in $\mathbb{F}_q[x]$.

1.2.1 Fermat's Little Theorem

Recall Fermat's little theorem says if p is prime then $a^p \equiv a \pmod{p}$, i.e., all $a \in \mathbb{Z}_p$ are roots of the polynomial $x^p - x \in \mathbb{Z}_p[x]$. In fact, over general finite fields \mathbb{F}_q one still has that all $a \in \mathbb{F}_q$ are roots of $x^q - x$. Thus, Fermat's little theorem can equivalently be written as

$$x^q - x = \prod_{a \in \mathbb{F}_q} (x - a).$$

Thus, given f one can compute g_1 , the product of all linear factors of f , via $g_1 := \gcd(f, x^q - x)$.

Generalizing this, one can also show that

$$x^{q^2} - x = \prod_{\substack{\alpha \in \mathbb{F}_q[x], \text{ irred.} \\ \deg(\alpha) \leq 2}} \alpha(x).$$

It follows that once the linear factors of f have been removed (by dividing by g_1) one can find g_2 , all the quadratic factors of f/g_1 , via $g_2 := \gcd(f/g_1, x^{q^2} - x)$.

This process can be generalized; the proper generalization of Fermat's last theorem that enables finding all irreducible factors of degree $d \geq 1$ is

$$x^{q^d} - x = \prod_{\substack{\alpha \in \mathbb{F}_q[x], \text{ irred.} \\ \deg(\alpha) | d}} \alpha(x).$$

Once all factors of f less than degree d have been removed from f (via $f \cdot \prod_{i < d} g_i^{-1}$) all irreducible factors of degree exactly d is found with $g_d := \gcd(f \cdot \prod_{i < d} g_i^{-1}, x^{q^d} - x)$.

1.2.2 Efficiency

How efficient would this be? If it was computed directly it would be an issue, since it would require $O(nq^d)$ field operations for computing $\gcd(f, x^{q^d} - x)$ where $\deg(f) = n$. Since $d = \Theta(n)$ in the worst case, computing this gcd would take exponential time in n . Thus, this proposed approach is seemingly totally infeasible.

A simple observation reduces the exponential running time to a polynomial one. Note that when computing $\gcd(f, x^{q^d} - x)$ the polynomial does not need to be constructed explicitly; it is enough to construct the polynomial $x^{q^d} - x \bmod f$ which has degree at most n . At first this might not seem useful, since computing $x^{q^d} - x \bmod f$ directly would also require exponential time.

However, we can use repeated squaring in order to compute the modular exponentiation $x^{q^d} \bmod f$ efficiently. Also, since

$$x^{q^d} \bmod f = (x^{q^{d-1}} \bmod f)^q \bmod f$$

and $x^{q^{d-1}} \bmod f$ was used on the previous iteration (in order to extract the factors of $d - 1$ from f). Thus, on the i th iteration one can save $x^{q^i} \bmod f$ and reuse it on iteration $i + 1$ without needed to recompute it.

1.2.3 Pseudocode

Input: Squarefree and monic $f \in \mathbb{F}_q[x]$ of degree n

$h := x, f_{\text{orig}} := f$

for i from 1 to n :

$h := h^q \bmod f_{\text{orig}}$

$g_i := \gcd(h - x, f)$

$f := f / g_i$

return g_1, \dots, g_n

1.2.4 Analysis

Assuming naive multiplication, the first operation in the loop uses $O(n^2 \log q)$ field operations and the last two operations in the loop use $O(n^2)$ field operations. Thus, the total cost is $O(n^3 \log q)$ field operations. With fast multiplication and fast gcd algorithms this cost can be brought down to $O^\sim(n^2 \log q)$.

1.3 Equal-degree factorization

Next, we consider the problem of splitting a polynomial whose irreducible factors are all of the same degree (like g_i from above). That is, given a squarefree monic $f \in \mathbb{F}_q[x]$ of degree n and the knowledge that all of f 's irreducible factors have degree d we want to decompose f as $f = g_1 \cdots g_k$ where $k = n/d$. In fact, it will be sufficient to develop an algorithm that can find any nontrivial factorization $f = g_1 \cdot g_2$ with neither g_1 or g_2 a constant polynomial, because then the problem is split into two smaller subproblems and we can run our splitting algorithm on both g_1 and g_2 separately.

In this section we assume that q is odd, though a similar algorithm can be developed for the even case.

1.3.1 The Chinese Remainder Theorem

The Chinese Remainder Theorem says that solving a modular equation $\bmod n = p_1 \cdots p_k$ (where the p_i are distinct primes) is equivalent to solving the equation \bmod each prime p_i individually:

$$f(x) \equiv 0 \pmod{n} \iff \left\{ \begin{array}{l} f(x) \equiv 0 \pmod{p_1} \\ \vdots \\ f(x) \equiv 0 \pmod{p_k} \end{array} \right\}.$$

It also provides an efficiently-computable way to take a solution of the system of equations on the right-hand side and translate it into a solution on the left-hand side. For example, if $x \equiv a_i$

$(\text{mod } p_i)$ for $1 \leq i \leq k$, it tells you how to find an $x \in \mathbb{Z}_n$ so that $x \equiv a \pmod{n}$. In ring theoretic terms this means that the ring \mathbb{Z}_n is equivalent to the “direct product” of the rings \mathbb{Z}_{p_i} :

$$\begin{aligned} \mathbb{Z}_n &\cong \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_k} \\ &\text{via the mapping} \\ x &\mapsto (x \bmod p_1, x \bmod p_2, \dots, x \bmod p_k) \end{aligned}$$

In fact, this idea applies to more than the integers modulo n ; the exact same relationship holds for polynomials modulo $f = g_1 \cdots g_k$ where the g_i are distinct irreducible polynomials. Just like \mathbb{Z}_n is the set of integers with arithmetic performed modulo n (this is also denoted by $\mathbb{Z}/n\mathbb{Z}$ or $\mathbb{Z}/(n)$), the set of polynomials $\mathbb{F}_q[x]$ with arithmetic performed modulo f is denoted by $\mathbb{F}_q[x]/(f)$. Then the Chinese Remainder Theorem for polynomials says that

$$\begin{aligned} \mathbb{F}_q[x]/(f) &\cong \mathbb{F}_q[x]/(g_1) \times \mathbb{F}_q[x]/(g_2) \times \cdots \times \mathbb{F}_q[x]/(g_k) \\ &\text{via the mapping} \\ f &\mapsto (f \bmod g_1, f \bmod g_2, \dots, f \bmod g_k). \end{aligned}$$

The structure of $\mathbb{F}_q[x]/(g)$ What does the arithmetic of $\mathbb{F}_q[x]/(g)$ look like when g is an irreducible polynomial? In fact, every nonzero polynomial in $h \in \mathbb{F}_q[x]/(g)$ has an inverse h^{-1} which can be computed by solving $h\alpha = 1$ for α in $\mathbb{F}_q[x]/(g)$, i.e., solving $h\alpha + g\beta = 1$ for $\alpha, \beta \in \mathbb{F}_q[x]$. We know how to solve this using the extended Euclidean algorithm on $h, g \in \mathbb{F}_q[x]$, assuming that h and g are coprime. (Which they are, since g is irreducible and $\deg(h) < \deg(g)$.)

Thus, $\mathbb{F}_q[x]/(g)$ is a field! When g has degree d , the field has the q^d elements $\{\sum_{i < d} c_i x^i : c_0, \dots, c_{d-1} \in \mathbb{F}_q\}$.

1.3.2 Fermat’s Little Theorem in a field

Fermat’s Little Theorem also applies to any finite field \mathbb{F} ; if \mathbb{F} has q^d elements and $a \in \mathbb{F}$ then $a^{q^d-1} = 1$. The original version of Fermat’s Little Theorem is recovered when q is prime and $d = 1$ (so $\mathbb{F} = \mathbb{Z}_p$).

Moreover, since the nonzero elements of any finite field form a cyclic group, what we called the “square root” of Fermat’s Little Theorem also holds in any finite field. That is, if \mathbb{F} has q^d elements and $a \in \mathbb{F}$ then $a^{(q^d-1)/2} = \pm 1$. (This is where we assume that q is odd, so that $(q^d - 1)/2$ is an integer.)

1.3.3 Applying Fermat’s Little Theorem

Now let’s go back to the problem of factoring $f \in \mathbb{F}_q[x]$ of degree n where we know that all of its irreducible factors g_i are of the same degree d . Suppose we choose a random $\alpha \in \mathbb{F}_q[x]$ of degree less than n . If we compute $\alpha^{q^d-1} \bmod f$ what will we get? Note that Fermat’s Little Theorem doesn’t apply directly here, since f is not irreducible and hence $\mathbb{F}_q[x]/(f)$ is not a field. However, the Chinese Remainder Theorem allows us to write $\mathbb{F}_q[x]/(f)$ as a direct product of fields where Fermat’s Little Theorem *does* apply.

By the Chinese Remainder Theorem, computing $\alpha^{q^d-1} \bmod f$ is essentially equivalent to computing

$$(\alpha^{q^d-1} \bmod g_1, \dots, \alpha^{q^d-1} \bmod g_k),$$

and because each g_i is specifically known to be irreducible and of degree d , by Fermat's Little Theorem the above vector of residues is $(1, 1, \dots, 1)$, at least assuming that α is coprime to each g_i . Since α was chosen randomly, it is likely α is coprime to each g_i . However, if this is not the case then things are even easier, since a factor of f can be recovered by $\gcd(\alpha, f)$. Thus, we can assume that α and each g_i are coprime.

Thus, assuming α is coprime to f we do in fact have $\alpha^{p^d-1} = 1$ in $\mathbb{F}_q[x]/(f)$, because $\alpha^{p^d-1} = 1$ in each of $\mathbb{F}_q[x]/(g_i)$ for $1 \leq i \leq k$.

1.3.4 Splitting f

We saw above that $\alpha^{p^d-1} = 1$ in $\mathbb{F}_q[x]/(f)$. What about the square root $\alpha^{(p^d-1)/2}$ in $\mathbb{F}_q[x]/(f)$? Note that this is **not** necessarily ± 1 , since recall that $\mathbb{F}_q[x]/(f)$ is **not** a field. In general rings (that are not fields), the identity 1 may have more square roots than just 1 and -1 .

Again, we can use the Chinese Remainder Theorem to evaluate what $\alpha^{(p^d-1)/2} \bmod f$ is. This is essentially equivalent to computing

$$(\alpha^{(q^d-1)/2} \bmod g_1, \dots, \alpha^{(q^d-1)/2} \bmod g_k),$$

which by the square root of Fermat's Little Theorem is a vector whose entries are all ± 1 . If by chance this vector is $(-1, -1, \dots, -1)$ then it **will** be the case that $\alpha^{(q^d-1)/2} = -1$ in $\mathbb{F}_q[x]/(f)$. However, this is quite unlikely, especially if k is large. In fact, since α was chosen randomly, we expect that 50% of the entries of the vector will be 1 and 50% of the entries of the vector will be -1 . If there is at least one 1 and -1 entry in the vector, then $\alpha^{(q^d-1)/2} \neq \pm 1$ in $\mathbb{F}_q[x]/(f)$.

The scenario when $\alpha^{(q^d-1)/2} \bmod f \neq \pm 1$ is the one that is beneficial, because that means that $\alpha^{(q^d-1)/2} \bmod g_i = 1$ for some i and $\alpha^{(q^d-1)/2} \bmod g_j = -1$ for some j . In other words, we have that g_j divides $\alpha^{(q^d-1)/2} - 1$ and g_i does not divide $\alpha^{(q^d-1)/2} - 1$. Thus, $\gcd(\alpha^{(q^d-1)/2} - 1, f)$ reveals a nontrivial factor of f , since it definitely includes g_j but not g_i .

1.3.5 Psuedocode

Input: Squarefree and monic $f \in \mathbb{F}_q[x]$ of degree n and d , the degree of all irreducible factors of f

Choose $\alpha \in \mathbb{F}_q[x]$ randomly of degree less than n .

If $g := \gcd(\alpha, f)$ is nontrivial, then f is split by $f = g \cdot (f/g)$.

Compute $A := \alpha^{(q^d-1)/2} \bmod f$.

If $g := \gcd(A - 1, f)$ is nontrivial, then f is split by $f = g \cdot (f/g)$.

If g is trivial, then repeat the algorithm with another random α .

1.3.6 Analysis

The running time of the algorithm is dominated by the computation of A , which using repeated squaring requires $O(d(\log q)n^2)$ operations in \mathbb{F}_q . How many times do we expect to get unlucky, though? In the worst case f has exactly two irreducible factors, i.e., $d = n/2$. In this case we expect 50% of the time A will be 1 or -1 which gives a trivial gcd. However, 50% of the time A

will not be ± 1 and as we saw above this results in a nontrivial g being found. Thus, even in the worst case we do not expect to have to try too many random α before a factor is found.

In order to completely factor f , the above algorithm must be called recursively at most n/d times, one for each factor of f . In the worst case, every time a factor g is recovered it would be of degree exactly d , meaning that g itself is irreducible and only a single recursive call needs to be made on f/g , a polynomial of degree $n - d$. In such a case the total cost of splitting f completely would be n/d times $O(dn^2 \log q)$, which is $O(n^3 \log q)$.

However, since α was chosen randomly, it is expected that g will contain about half of the irreducible factors of f and f/g will contain the other half of the irreducible factors. In this case, there will be two recursive calls and each will be of size roughly $n/2$. In such a case the depth of the recursion is expected to be *logarithmic* in n/d , not linear in n/d . Thus, the expected running time of the algorithm is $O(dn^2 \log(q) \log(n/d))$ field operations.

1.4 Factoring Squarefull Polynomials

So far, we've assumed that the input polynomial $f \in \mathbb{F}_q[x]$ to factor is squarefree. In fact, everything in the algorithm we described works if f is "squarefull" (meaning that it is divisible by an irreducible polynomial more than once) but we need to be a bit careful.

After the distinct-degree factorization step, we will have computed g_1, \dots, g_n which are each squarefree polynomials because $x^{q^d} - x$ is a squarefree polynomial. However, the product of the g_i will not equal f exactly when f is squarefull. Instead, we will have $f = S \cdot \prod_i g_i$ where S is a product of the duplicated factors of f .

The input to the equal-degree factorization step will be the g_i , so nothing changes in the equal-degree step which will still factor the g_i into their irreducible components.

At the end, we take each irreducible factor of the g_i and see if divides S (and if so, how many times). This can be done with repeated quotient and remainder. Since the polynomials involved all have degree at most n this will be less than the cost of the other parts of the algorithm.

1.4.1 Pseudocode for a Complete Factoring Algorithm

Input: Monic $f \in \mathbb{F}_q[x]$ of degree n

$h := x, f_{\text{orig}} := f$

for i from 1 to n :

$h := h^q \bmod f_{\text{orig}}$

$g := \gcd(h - x, f)$

$f := f/g$

Apply equal-degree factorization on g to write $g = g_1 \cdots g_k$

Add g_1, \dots, g_k to the list of irreducible factors

for j from 1 to k :

while g_j divides f :

$f := f/g_j$

Add another copy of g_j to the list of irreducible factors

Output the list of irreducible factors of f

1.4.2 Analysis

The bottleneck of the outer loop in the complete factoring algorithm is the cost of computing the equal-degree factorization $g = g_1 \cdots g_k$. During step i of the outer loop say that the degree of g is m_i . Then the equal-degree factorization on step i will produce the m_i/i factors $g_1, \dots, g_{m_i/i}$ and cost of finding these will be an expected $O(in^2 \log(q) \log(m_i/i))$ field operations.

Note that we have

$$i \log(m_i/i) = m_i \frac{\log(m_i/i)}{m_i/i} \leq m_i, \quad \text{since } \frac{\log x}{x} \leq 1.$$

Thus iteration i of the loop takes an expected $O(m_i n^2 \log(q))$ field operations. Because $\sum_{i=1}^n m_i \leq n$ the total expected running time of the entire algorithm is $\sum_{i=1}^n O(m_i n^2 \log(q)) = O(n^3 \log(q))$ field operations.

1.5 Factoring Polynomials in $\mathbb{Z}[x]$

Lastly, we will see how factoring polynomials in $\mathbb{F}_q[x]$ can also be used as a basis for factoring polynomials in $\mathbb{Z}[x]$. The algorithm we present will have exponential running time, but with some additional cleverness can be made to run in polynomial time.

First, note that to factor polynomials in $\mathbb{Z}[x]$ actually requires the ability to factor integers. For example, suppose all coefficients of your input polynomial $f \in \mathbb{Z}[x]$ are divisible by the same number N . Then in order to write f as a product of factors where each factor cannot be factored any further requires N to also be factored. One workaround to this is to consider the factorization problem over $\mathbb{Q}[x]$ instead of $\mathbb{Z}[x]$, since over \mathbb{Q} every nonzero constant is invertible and cannot be factored further. If we ignore the issue of factoring integer constants, then the factoring problem in $\mathbb{Q}[x]$ is equivalent to the factoring problem in $\mathbb{Z}[x]$. We will sidestep the issue by just assuming that $f \in \mathbb{Z}[x]$ is monic.

1.5.1 Reducing $f \bmod p$

The coefficients of the polynomial $f \in \mathbb{Z}[x]$ can be reduced modulo p to form a polynomial $\bar{f} \in \mathbb{Z}_p[x]$. Our idea will be to compute \bar{f} for large enough p , then factor \bar{f} over $\mathbb{Z}_p[x]$. This will provide a factorization

$$\bar{f} = g_1 \cdots g_k \tag{**}$$

for irreducible polynomials $\bar{g}_i \in \mathbb{Z}_p[x]$. If a polynomial is irreducible in $\mathbb{Z}_p[x]$ this definitely implies it is irreducible in $\mathbb{Z}[x]$ (because equality in \mathbb{Z} implies equality in \mathbb{Z}_p). However, the converse does not hold: a polynomial might factor *farther* over \mathbb{Z}_p than it does over \mathbb{Z} .

Note that if p is chosen large enough, one can recover a polynomial α from its reduction $\bar{\alpha}$ modulo p . For example, suppose that you know the coefficients of α are all at most 5 in absolute value and the bar denotes reduction modulo $p = 11$. If $\bar{\alpha} = x^3 - 5x^2 + 5x - 2$ then the coefficients of α and $\bar{\alpha}$ must be the same, because any other way of “lifting” the coefficients of \mathbb{Z}_p to \mathbb{Z} would introduce

a coefficient c with $|c| \geq 6$. If the polynomial α we want to recover has a maximum coefficient of absolute value N , then we choose $p > 2N$. Using the “symmetric range” $\{-(p-1)/2, \dots, (p-1)/2\}$ of residues mod p , we can capture all of α ’s coefficients exactly mod p , and therefore will be able to recover the α from $\bar{\alpha}$.

So by taking p large enough we will be able to recover the coefficients of the factors of f from their modular reductions—if we can compute their modular reductions. Say f_1 is an irreducible factor of f . Since $f \bmod p$ can only factor *farther* than f , it must be the case that some product of the g_i s in $(**)$ must combine in order to give f_1 , i.e., there is a set $S \subseteq \{1, \dots, k\}$ such that

$$f_1 = \prod_{i \in S} g_i.$$

If we can find the set S then we would be able to compute the product f_1 and we can easily test that f_1 is indeed a true factor of f by checking that $f \bmod f_1 = 0$. The problem with this approach is that there seems no easy way to find the set S . Of course, we can try all possible subsets $S \subseteq \{1, \dots, k\}$ and figure out which ones yield true factors in $\mathbb{Z}[x]$, not $\mathbb{Z}_p[x]$. Of course, this requires exponential time in the number of factors.

1.5.2 Squarefree Factorization

Incidentally, it is easy to find the squarefree part of a polynomial in $\mathbb{Z}[x]$ or $\mathbb{Q}[x]$ (or more generally any field \mathbb{F} where $1 + 1 + \dots + 1 \neq 0$ for arbitrary many additions). This is because in $\mathbb{F}[x]$ a factor divides $f = \sum_{i \geq 0} a_i x^i \in \mathbb{F}[x]$ more than once if and only if it divides the derivative of f , defined by $f' := \sum_{i \geq 1} i a_i x^{i-1}$.

Thus, the squarefree part of f can be computed by $f / \gcd(f, f')$. You have to be careful over a finite field, as the precondition on the field isn’t met (in that case $1 + 1 + \dots + 1 = 0$ when there are p ones) and it is possible that $f' = 0$ even when $f \neq 0$. Though even in a finite field it still is the case that $\gcd(f, f') = 1$ does imply that f is squarefree.

1.5.3 Pseudocode

Input: A squarefree and monic $f \in \mathbb{Z}[x]$ of degree n and maximum coefficient in absolute value of A

Let $p \in [2B, 4B]$ be a random prime where $B := 2^n A \sqrt{n+1}$

Factor $\bar{f} \in \mathbb{Z}_p[x]$ as $g_1 \cdots g_k$ for irreducible $g_i \pmod{p}$ and write the g_i as polynomials with coefficients absolutely bounded by $p/2$

$T := \{1, \dots, k\}$

for all $S \subseteq T$, starting with the smallest S :

$g := \prod_{i \in S} g_i$

if $f \bmod g = 0$ then

$f := f/g$

$T := T \setminus S$

add g to the list of irreducible factors

Output the list of irreducible factors of f

1.5.4 Analysis

Unfortunately, the loop may run exponentially many times, since there are 2^k subsets of T . There is a better method for determining which g_i combine together to form actual irreducible factors of f , but it involves more mathematical machinery. In particular, an algorithm of Lenstra, Lenstra, and Lovász from 1982 is able to solve the factoring problem in $\mathbb{Q}[x]$ in polynomial time in $\deg(f) = n$ and in A , the size of the coefficients of f . At the time this was somewhat surprising, even to the discoverers. Their method is even totally deterministic, which at first seems paradoxical since it relies on a $\mathbb{Z}_p[x]$ factoring method and there is no known deterministic polynomial time $\mathbb{Z}_p[x]$ factoring method. This is possible because there is a deterministic variant of the $\mathbb{Z}_p[x]$ factoring method that runs in time $O^\sim(n^3 + pn^2)$. Even though this is exponential in $\log(p)$, they are able to find an acceptable prime p (without relying on randomness) for which p is small enough that running the exponential $\mathbb{Z}_p[x]$ factoring algorithm will still be polynomial time in n and A .