Integer factoring using small algebraic dependencies*

Manindra Agrawal[1], Nitin Saxena[2], and Shubham Sahai Srivastava[3]

1 **Indian Institute of Technology, Kanpur, INDIA**
`manindra@cse.iitk.ac.in`
2 **Indian Institute of Technology, Kanpur, INDIA**
`nitin@cse.iitk.ac.in`
3 **Indian Institute of Technology, Kanpur, INDIA**
`ssahai@cse.iitk.ac.in`

Abstract

Integer factoring is a curious number theory problem with wide applications in complexity and cryptography. The best known algorithm to factor a number n takes time, roughly, $\exp(2\log^{1/3} n \cdot \log^{2/3}\log n)$ (*number field sieve*, 1989). One basic idea used is to find two squares, possibly in a number field, that are congruent modulo n. Several variants of this idea have been utilized to get other factoring algorithms in the last century. In this work we intend to explore new ideas towards integer factoring. In particular, we adapt the AKS primality test (2004) ideas for integer factoring.

In the motivating case of semiprimes $n = pq$, i.e. $p < q$ are primes, we exploit the difference in the two Frobenius morphisms (one over $\mathbb{F}_p$ and the other over $\mathbb{F}_q$) to factor n in special cases. Specifically, our algorithm is polynomial time (on number theoretic conjectures) if we know a *small algebraic dependence* between p, q. We discuss families of n where our algorithm is significantly faster than the algorithms based on known techniques.

1998 ACM Subject Classification **Primary:** F.2.1 ; **Secondary:** I.1.2

Keywords and phrases integer, factorization, factoring integers, algebraic dependence, dependencies

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.7

1 Introduction

Factoring a positive integer n is the process of finding a positive integer m $(1 < m < n)$ that divides n. Integer factorization has been fascinating mathematicians for centuries [9]. There has been continuous attempts to expand our abilities to factor larger and larger integers (see [14], [3]).

In general, factoring a composite number is widely believed to be a "hard" problem, with no efficient general purpose algorithms known. There are several *special* purpose factoring algorithms which can factor composites efficiently, provided some specific property is satisfied. Some of the algorithms being: Trial division (or Eratosthenes sieve, see [12]), Fermat's factorization [15], Euler's factorization [20][22], Pollard's rho algorithm [24], Pollard's $p-1$ algorithm [23], Williams' $p+1$ algorithm [30], Lenstra's elliptic curve factorization [18], quadratic sieve [10], and the number field sieve [6]. Sieve ideas have been the most successful ones in factoring, see an excellent survey in [26].

* N.S. is supported by DST/SJF/MSA-01/2013-14.

The "hardness" of integer factorization has no known proof, but, the belief hinges only on our inability to factor a general composite efficiently. However this belief is so strong, that the most widely used public key cryptosystems (eg. RSA [5]) are based on this "inherent" difficulty to factorize integers. Such applications in cryptography make integer factorization problem even more interesting. Giving a polynomial time algorithm to factorize any given integer, might result in breaking most widely used cryptosystems. On the other hand, proving (or giving evidence) that no efficient algorithm exists for factoring a general composite would further strengthen the trust on these cryptosystems.

This does not mean that no progress was made in the direction, to come up with a general purpose algorithm. Although there is no algorithm that can factor (even heuristically) all integers in "polynomial time" (i.e. polynomial in the bit-size of the input number), yet there are several algorithms that run in *subexponential* time (i.e. $\exp(O(\log^{\epsilon} n))$ time for $\epsilon < 1$). These are faster than the simple "high school" method (i.e. trial division algorithm, having exponential running time). The fastest general purpose algorithm for factoring a number n, is the general number field sieve (see [16]), with heuristic running time $\exp\left(\left(\sqrt[3]{64/9} + o(1)\right)(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right)$. The other widely used algorithm in practice is the quadratic sieve algorithm [25], having running time $\exp\left((1+o(1))\sqrt{\log n \log\log n}\right)$, which is a modification of Dixon's algorithm [8], that had a (rigorously provable) running time of $\exp\left((2\sqrt{2}+o(1))\sqrt{\log n \log\log n}\right)$.

In 1997, Peter Shor discovered the first polynomial time algorithm for factoring integers on a *quantum computer* [28]. To factor an integer n, it takes $O((\log n)^2 \log\log n \log\log\log n)$ time[1] on quantum computer and $O(\log n)$ post-processing time on classical computer for converting the output of quantum computer to factors of n. If one day quantum computation becomes feasible for large inputs, then this will have serious implications in cryptography [4].

One common thread in these, increasingly complex, algorithms is the trick of finding two squares in some number field, such that the *difference of the squares*, say $a^2 - b^2$, is a multiple of n. Then we can hope that the factors $(a-b), (a+b)$ would also lead us to the factors of n. The origins of this trick dates back atleast to Fermat, and was also exploited by Gauss, Seelhoff and Kraitchik (see the early history of factoring in [31]). One wonders whether other natural tricks or ideas could be discovered for factoring integers.

In this work we propose a method for factoring semiprimes $n = pq$ (i.e. $p < q$ are primes) using the difference in the Frobenius morphisms over the finite fields $\mathbb{F}_p$ and $\mathbb{F}_q$. We do this by working in a *cyclotomic* ring extension $(\mathbb{Z}/n\mathbb{Z})[\zeta] := \mathbb{Z}[X]/\left(n, \frac{X^r-1}{X-1}\right)$. We pick a random element $u(\zeta) \in (\mathbb{Z}/n\mathbb{Z})[\zeta]$ and compute the exponentiation u^e, for a carefully chosen positive integer e. For example, when $e = n$ we can invoke the Frobenius morphisms to deduce $u(\zeta)^n = u(\zeta^p)^q \pmod p$ and $u(\zeta)^n = u(\zeta^q)^p \pmod q$. A similar line of thought has been explored in [7], where they viewed the problem from the perspective of AKS [1] polynomial. Although no family of n was identified in that work to be particularly good. The asymptotic complexity of the algorithm was also not analyzed, but some supporting experimental data was included.

We identify certain families of n where this idea gives a fast factoring algorithm. Especially, in our main result we pick e corresponding to a *known* algebraic dependency of p and q. In this case, we show that the ring computations in $(\mathbb{Z}/n\mathbb{Z})[\zeta]$ are expected to factorize n. We believe that such computations in the cyclotomic ring have a good chance in further improving the state of the art in factoring. Similar techniques were utilized in [1] to give the

[1] We can shorten this using the soft-Oh notation as $\tilde{O}(\log^2 n)$.

first deterministic poly-time primality test. Moreover, for integer factoring even "heuristic" algorithms that are expected to run in poly-time (in the worst-case) would be of great interest.

Our notion of "small" algebraic dependence and the proof of its existence is captured in the following proposition. We say that a bivariate polynomial $f(X,Y)$ is *nondegenerate* if there appears, with a nonzero coefficient, a monomial X^iY^j in f such that $i \neq j$.

▶ Proposition 1.1 (Small dependency exists). For numbers $d, a < b \in \mathbb{N}$, there exists a degree $\leq d$ nondegenerate integral polynomial $f(X,Y)$ of sparsity 2γ and coefficients c_i's of magnitude at most $b^{d/(\gamma-1)}$ such that: $f(a,b) = \sum_{i=1}^{2\gamma} c_i\, a^{\alpha_i}\, b^{\beta_i} = 0$. (Note that $2\gamma \leq \binom{d+2}{2}$ as $0 \leq \alpha_i + \beta_i \leq d$.)

It is proven in Section 3. Recall that Fermat's factoring algorithm works fast when the primes p, q are really close[2]; formally, when there is an $f(x,y) = y - x - \alpha$, for a small α, such that $f(p,q) = 0$. We generalize the condition of Fermat's factoring algorithm to higher degree dependencies (and with more general coefficients). The above proposition gives the parameters for such f to exist. Our algorithms will require the *knowledge* of such an f (unfortunately, in general, it may be hard to find f given only n).

One such interesting dependence is addressed in Section 4.2. For $n = pq$, $p < q$, we represent q in base p as $q = a_0 + a_1.p + a_2.p^2 + a_3.p^3 + \cdots$. We define the p^{th} *norm of* q as $|q|_p := \prod_i (a_i + 1)$. Given a small bound B on $|q|_p$, our algorithm factors n in time $O(B^2 \log^2 n)$. This immediately gives us a family of n which can be factored efficiently (under certain number theory conjectures) using our algorithm. This family is a natural generalization of the family of numbers ($n = ab$, where $b - a$ is small) that can be factored efficiently using Fermat's factoring algorithm.

Our general approach works in polynomial time, assuming that a suitable dependency is provided (and that certain number theory conjectures hold). The algorithm presented in this paper runs in $\tilde{O}(\gamma^3 d \log^2 n)$ time, where d is the degree bound of the dependency, γ is its sparsity, and n is the number to be factored. Observe that once such a bivariate nondegenerate dependency $f(X,Y)$, of degree d is given, we can easily transform it to get a univariate polynomial $X^d f(X, n/X)$ which has p as a root. Notice, that it is important here that the dependency is nondegenerate. For degenerate dependency of the form $f(X,Y) = \sum_{i \leq d} a_i X^i Y^i$ the substitution $f(X, n/X)$ will give us a number instead of a univariate polynomial, and we could not proceed further.

Now, once we get a univariate polynomial $f' := X^d f(X, n/X)$ which has p as a root, we could simply try to find its integral roots by factoring it using Schönhage's algorithm [27] having time complexity $\tilde{O}(d^4 \cdot (d^2 + \log^2 |f'|))$, where $|f'|$ upper bounds the coefficients in f'. On the other hand, our new approach is sensitive to sparsity γ and tolerates bigger coefficients. So, for dependencies, having 'small' γ and 'large' d, our algorithm will outperform Schönhage's algorithm by several orders. For example, given dependence $f(x,y) = y + c_1 x^d + c_0$, where $|c_1| = |c_0| = n^{O(d)}$, our algorithm will factor it in time $\tilde{O}(d \log^2 n)$, whereas Schönhage's algorithm will take time $\tilde{O}(d^4 \cdot (d^2 + d^2 \log^2 n)) = \tilde{O}(d^6 \log^2 n)$.

The main result established is presented in Section 5. The section presents the algorithm to factor n when a small dependency is provided. The result is summarized by the following theorem.

▶ Theorem 1.2 (Main Result). For an integer $n = p \cdot q$ ($p < q$ are primes), given a nondegenerate integral (p,q) dependency of the form $f(X,Y) = \sum_{i=1}^{\gamma} c_i\, X^{\alpha_i}\, Y^{\beta_i}$, where $\forall i$, $0 \leq \alpha_i + \beta_i \leq d$,

[2] Essentially, one tries to find $q - p$ by brute-force.

$|c_i| = n^{O(d)} := A$ we can factor n in $\tilde{O}(\gamma^3 d \log^2 n)$ time. (Assuming Artin's conjecture & 3.)

We also present an alternate analysis of this algorithm in Section 6. This section also generalizes the result for integers of the form $n = p \cdot n'$, where p is a prime smaller than the largest prime factor of n. The following theorem presents the result of that section.

▶ Theorem 1.3. For an integer $n = p \cdot n'$ (where p is a prime smaller than the largest prime factor of n), given a nondegenerate integral (p, n') dependency of the form $f(X, Y) = \sum_{i=1}^{\gamma} c_i X^{\alpha_i} Y^{\beta_i}$, where $\forall i$, $0 \leq \alpha_i + \beta_i \leq d$, $|c_i| < n^d$, we can factor n in $\tilde{O}(\mu^3 \cdot \gamma d^4 \log^2 n)$ time. Here $\mu := \sum_i e_i$ for the prime factorization $n = \prod_i p_i^{e_i}$. (Assuming Artin's conjecture & 4.)

Here as well, for $\mu, \gamma = O(1)$ the time complexity is better than that of simply factoring $X^d f(X, n/X)$ by Schönhage's algorithm. Also, the algorithm seems simpler than the sophisticated lattice computations that underlie Schönhage's polynomial factoring algorithm (see [13]).

The paper is organized into following sections: Section 2 talks about the notations and results used in the paper. Section 3 proves the existence of small dependence. In Section 4, we discuss two simple dependencies as motivating examples, and explore the idea of exponentiation in the cyclotomic ring to factor n. Section 5 presents the main result of the paper. An alternate analysis of the algorithm is presented in Section 6.

2 Notation and Preliminaries

This section states the notations and number theory results that we will use later.

Polynomial notation. The form of polynomials that we compute in this work is exponentiation; motivated by the *AKS polynomial* (see [1]) used for primality testing:

$$\mathcal{P} = a(x)^e \pmod{n, x^r - 1}, \quad \text{where } a(x) \text{ is a polynomial.} \tag{1}$$

For technical reasons we will actually work modulo the r-th cyclotomic polynomial $\varphi_r(x)$. Then, we represent exponentiation by the following shorthand notation

$$\mathcal{P} = a(\zeta_r)^e \pmod{n}, \text{ and might drop } r \text{ when clear from the context.} \tag{2}$$

Formally, this arithmetic happens in the ring $(\mathbb{Z}/n\mathbb{Z})[\zeta_r] := \mathbb{Z}[X]/(n, \varphi_r(X))$, where every element can be written as a $(\mathbb{Z}/n\mathbb{Z})$-linear-combination of the monomials $\{X^i \mid 0 \leq i \leq \varphi(r) - 1\}$, where $\varphi(r)$ is the Euler totient function (also, the degree of the cyclotomic polynomial). This will be our standard representation.

In this paper we assume r to be a prime, mainly, to simplify the analysis since $\varphi_r(x) = (x^r - 1)/(x - 1)$. Also for composite r's the cyclotomic extension is quite well structured. For the basic properties of the cyclotomics see [29, Chap.2].

Artin's conjecture. Emil Artin (1927, see [21]) conjectured: For any non-square $a \in \mathbb{Q} \setminus \{-1\}$ there exist infinitely many primes p such that a is a *primitive root* modulo p, i.e. the multiplicative order $\text{ord}_p(a) = p - 1$.

There has been impressive positive progress towards this conjecture [11]. Moreover, a quantitative version of this conjecture is also believed to be true.

▶ Conjecture (Artin's conjecture, see [21]). For any non-square $a \in \mathbb{Q} \setminus \{-1\}$, the number of primes $p \leq x$ with $\text{ord}_p(a) = p - 1$ is asymptotically at least $\mathcal{C}_{Artin} \cdot \pi(x)$. ($\pi(x)$ is the number of primes in the interval $[1, x]$ and $\mathcal{C}_{Artin} = 0.3739558136192\cdots$.)

Frobenius morphism. For a prime p consider the polynomial ring $R := \mathbb{F}_p[X]$ over the finite field $\mathbb{F}_p$. Consider the map $\phi : R \to R$ given by the exponentiation $a(X) \mapsto a(X)^p$. It is easy to see that ϕ is actually a (ring) endomorphism of R, and the trivial[3] automorphism of $\mathbb{F}_p$. In other words, we have the useful identity: $\forall a(X) \in R$, $a(X)^p = a(X^p)$.

Other notations. We use $[n]$ to denote the set $\{1, 2 \cdots, n\}$. The notation $\log_{q,r} p$ is used to denote, the *discrete log*, $\log_q p$ in the field $\mathbb{F}_r$, i.e. it is the exponent $i \in \{0, \ldots, r-2\}$ such that $p = q^i \pmod r$. Here, we assumed that r is a prime, and q is a primitive root modulo r. (We hope to get such an r corresponding to a q as the density of r's is high as per Artin's conjecture.) Bold faced symbols (e.g. $\boldsymbol{\alpha}$) represent vectors. $\mathbb{F}_q[\zeta]$ represents some ring $\mathbb{F}_q[X]/(\varphi_r(X))$.

We recall a useful standard property of cyclotomic polynomials. This is the main reason why Artin's conjecture appears in this work.

▶ Lemma 2.1. Let $q \neq r$ be primes. The integral polynomial $\varphi_r(x) = (x^r - 1)/(x - 1)$ is irreducible over $\mathbb{F}_q$ iff q is a primitive root modulo r.

Proof. Let q generate $\mathbb{F}_r^*$. Wlog assume $r > 2$, as $\varphi_r(x)$ is linear for $r = 2$. Suppose $\varphi_r(x)$ is reducible and has a degree d factor $g(x)$, where $d \in [r-2]$. Let α be a root of $g(x)$ in the (splitting) field $\mathbb{F}_q[x]/(g(x))$. As this is the field $\mathbb{F}_{q^d}$, the multiplicative order $\mathrm{ord}(\alpha)$ will divide $q^d - 1$. Since α is a root of $x^r - 1$, we also have $\mathrm{ord}(\alpha) | r$. Thus, $\mathrm{ord}(\alpha)$ is 1 or r. It cannot be 1 as $q \neq r$. So,

$$\mathrm{ord}(\alpha) = r.$$

Consequently, $r \,|\, q^d - 1$

$$q^d = 1 \pmod r$$

$$(r-1) \,|\, d \quad [\because q \text{ generates } \mathbb{F}_r^*] .$$

This contradicts $d \in [r-2]$. Hence, $\varphi_r(x)$ is irreducible modulo q.

For the converse note that $\varphi_r(x)$ being irreducible modulo q, means that it divides $x^{q^{r-1}} - x$, and no other $x^{q^i} - x$ for a smaller i. Equivalently, $r \,|\, q^{r-1} - 1$ and no other $q^i - 1$ for a smaller i. Thus, q generates $\mathbb{F}_r^*$. ◀

3 Existence of small dependencies

Our basic idea is based on the following elementary property of numbers.

Proof for Proposition 1.1. Clearly, $2\gamma \leq \binom{d+2}{2} =: \gamma_0$ which is the upper bound for the number of exponents (α_i, β_i) in f.

Let $A := 2b^{d/(\gamma-1)}$. Consider a set $\mathcal{S}$ of nondegenerate combinations (i.e. $i_1 \neq i_2$ for at least one monomial in each sum),

$$\mathcal{S} := \left\{ \sum_{0 \leq i_1 + i_2 \leq d} \alpha_{i_1,i_2} \cdot a^{i_1} b^{i_2} \;\middle|\; \alpha_{i_1,i_2} \in \mathbb{Z}, |\alpha_{i_1,i_2}| \leq \frac{A}{2}, \text{ at most } \gamma \; \alpha_{i_1,i_2}\text{'s are nonzero} \right\}.$$

[3] Fermat's little theorem (1640).

Then, we have

$$\forall \beta \in \mathcal{S},\ |\beta| \leq \gamma \cdot \frac{A}{2} \cdot b^d\,.$$

Consider the set $\mathcal{V}$ comprising the coefficient-vectors $\boldsymbol{\alpha}$ corresponding to every element of $\mathcal{S}$. Then the cardinality of $\mathcal{V}$ can be lower bounded (by doing a sum over the possible supports of $\boldsymbol{\alpha}$) as,

$$\begin{aligned}|\mathcal{V}| &\geq \binom{\gamma_0}{\gamma} \cdot A^{\gamma} + \binom{\gamma_0}{\gamma-1} \cdot A^{\gamma-1} + \cdots + \binom{\gamma_0}{1} \cdot A + 1 \\ &> \left(\frac{\gamma_0}{\gamma}\right)^{\gamma} \cdot A^{\gamma} \qquad \text{[Simple binomial estimate]}\end{aligned}$$

Clearly, if $|\mathcal{V}| = |\mathcal{S}|$ is greater than $\max\{|\beta| \mid \beta \in \mathcal{S}\}$, then by the pigeon-hole principle there will be atleast two distinct vectors $\boldsymbol{\alpha}, \boldsymbol{\alpha'} \in \mathcal{V}$ that correspond to the same number $\beta \in \mathcal{S}$. This gives us the desired dependency,

$$0 = \sum_{0 \leq i_1 + i_2 \leq d} (\alpha_{i_1,i_2} - \alpha'_{i_1,i_2}) \cdot a^{i_1} b^{i_2}.$$

Hence, for the desired small dependency it suffices to ensure that,

$$\begin{aligned}&|\mathcal{V}| > \max\{|\beta|\} \\ \text{or } &\left(\frac{\gamma_0 A}{\gamma}\right)^{\gamma} \geq \gamma \cdot \frac{A}{2} \cdot b^d \\ \text{or } &A^{\gamma-1} \geq \left(\frac{\gamma}{\gamma_0}\right)^{\gamma} \cdot \frac{\gamma b^d}{2} \\ \text{or } &A \geq \left(\frac{\gamma}{\gamma_0}\right)^{\gamma/(\gamma-1)} \cdot \left(\frac{\gamma}{2}\right)^{1/(\gamma-1)} \cdot b^{d/(\gamma-1)} \\ \text{or } &A \geq 2 \cdot b^{d/(\gamma-1)} \qquad (3)\end{aligned}$$

Clearly, for our A, Equation 3 is satisfied. Hence, the required dependency exists. ◀

Hence, there is a trade-off between the sparsity (γ) and the magnitude (c_i) of the dependency polynomial.

▶ Remark. This bound is not optimal. Eg. if we allow f to have sparsity γ_0 then a slightly better bound of $A = 2b^{d/(\gamma_0-1)}$ can be shown; which for $d = 1$ seems optimal.

For a nonconstant γ, or a superpolynomial coefficient-bound A, it would be quite expensive to search for such a dependency f in general. So, our algorithms would be interesting only for those $n = pq$ where it is relatively easy to find an f such that $f(p, q) = 0$.

4 Motivating Dependencies

In the previous section we have shown that a "small" dependency will always exist (Proposition 1.1). Although, in general this dependency could be hard to find, but in special cases there are several natural dependencies. Some of them have already been witnessed and worked upon. An example of one such naturally occurring dependency is, when the two factors are very close to each other. In other words, for $n = pq$, $q - p = \alpha$, where α is some *small*[4]

[4] The term "small" is used vaguely here. The running time of the algorithm is proportional to α. Hence, we could work with α according to the running time we aim for. For polynomial time algorithm, we want $\alpha = \text{poly} \log n$.

constant. Consequently, in such cases both p and q will be close to $\sqrt{n}$. Hence, to factor n, we can simply use the trial division algorithm, starting from $\sqrt{n}$, which would work efficiently as α is small. A more sophisticated and faster way to factor a number having such a dependency ($q - p = \alpha$) would be to use Fermat's factorization method. We propose a new method to factor numbers having such a dependency.

4.1 Factoring numbers having dependency of the form $q - p = \alpha$

Assuming that we have n and a bound B such that $q - p = \alpha \leq B$ the idea is to pick an element $(x + a) \in (\mathbb{Z}/n\mathbb{Z})[x]/(n, x^r - 1)$ and compute $\mathcal{P} := (x + a)^n$, for an r slightly bigger than B. The hope is that the two (underlying) polynomials, $\mathcal{P}_q = (x^q + a)^p \pmod{q, x^r - 1}$ and $\mathcal{P}_p = (x^p + a)^q \pmod{p, x^r - 1}$ would have different *supports* (i.e. there is a monomial x^i, $i \in [0, 1, \cdots, r - 1]$, that appears with zero coefficient in exactly one of the polynomials[5]). We can clearly see, that $r \leq q$ is the trivial upper bound. But we can likely improve this upper bound further.

For $r < p < q$ it seems likely that for most a's, $(x + a)^p \pmod{q, x^r - 1}$ will have each of the r monomials (i.e. x^i, $i \in [0, 1, \cdots, r - 1]$) appearing with nonzero coefficient[6]. We pose this formally.

▶ Conjecture 1. For primes $p < q$, $1 \leq r < p$ and a random $a \in \mathbb{Z}/q\mathbb{Z}$, $(x+a)^p \pmod{q, x^r - 1}$ is full support with high (i.e. constant) probability.

The rationale for this conjecture is that we expect $(x + a)^p$ to be a "random" element in the cyclotomic ring. So, it will be rare that there is a zero coefficient in its standard representation.

On the other hand $(x + a)^q \pmod{p, x^r - 1}$, for $r \geq 2B + 3$, has proper support as we now show.

▶ Theorem 4.1. For primes $p < q$ and $r \geq 2(q - p) + 3$, $(x + a)^q \pmod{x^r - 1, p}$ is proper support.

Proof. Consider the polynomial,

$$\begin{aligned}
\mathcal{P}_p &= (x + a)^q \pmod{p} \\
&= (x + a)^p (x + a)^{q-p} \pmod{p} \\
&= (x^p + a)(x + a)^{q-p} \pmod{p} \\
&= \underbrace{(x^p)(x + a)^{q-p}}_{\text{Sparsity} \leq q-p+1} + \underbrace{a \cdot (x + a)^{q-p}}_{\text{Sparsity} \leq q-p+1} \pmod{p}.
\end{aligned}$$

Hence, $\text{Sparsity}(\mathcal{P}_p) \leq 2(q - p + 1)$. So, taking $r \geq 2(q - p + 1) + 1$ will ensure that atleast one monomial in $(x + a)^q \pmod{x^r - 1, p}$ has the zero coefficient. ◀

These observations motivate the following algorithm.

[5] It is easy to see that this implies that one of the coefficients in $\mathcal{P}$ will share a nontrivial gcd with n.
[6] Such a polynomial we call *full* support, and its opposite is *proper* support.

Algorithm 1 Factoring Integer : $FAC_1(n, B)$

Require: Odd $n = pq$ ($p < q$ are primes) and a parameter $B \geq (q - p)$.

1: $r \leftarrow 2$
2: **while** $r \leq 2B + 3$ and n is not factored **do**
3: Choose a random number $a < n$
4: Compute $\mathcal{P} = (x + a)^n \pmod{x^r - 1, n}$
5: Take gcd of n with ra, and with the coefficients of $\mathcal{P}$.
6: **if** n is factored **then**
7: **return** $factor$
8: **return** 0

Time complexity. The polynomial computation in step 4, takes time $\tilde{O}(r \log^2 n)$ using fast arithmetic. Taking GCD in step 5, takes similar time. Hence, the overall time complexity of the algorithm is $\tilde{O}(B^2 \log^2 n)$. Note that it is a probabilistic algorithm based on Conjecture 1. It can be seen as an alternative (albeit slower) to Fermat's factoring algorithm.

4.2 Bound based on p^{th} norm of q

This subsection discusses a more general, yet natural, dependency and presents the algorithm to factor n in such cases.

Let us represent q in base p (so that a_i's are in $[0, \cdots, p-1]$),

$$q = a_0 + a_1.p + a_2.p^2 + a_3.p^3 + \cdots .$$

We define the p^{th} *norm of* q as $|q|_p := \prod_i (a_i + 1)$. It is defined as a 'measure' for the size of the coefficients in base p representation of q.

Can we factor $n = pq$ (primes $p < q$) if we have an upper bound B on $|q|_p$? We can generalize the methods of the last section.

By Conjecture 1 we expect $(x + a)^p \pmod{x^r - 1, q}$ to be full support, for random a. The other modulus is covered by the following simple observation.

▶ Theorem 4.2. For primes $p < q$ and $r > |q|_p$, $(x + a)^q \pmod{x^r - 1, p}$ is proper support.

Proof. By using the base-p representation of q we have,

$$\begin{aligned}
(x + a)^q &= (x + a)^{a_0 + a_1.p + a_2.p^2 + a_3.p^3 + \cdots} \\
&= \prod_i (x + a)^{a_i.p^i} \\
&= \prod_i (x^{p^i} + a)^{a_i} \pmod{p} \\
\therefore \text{Sparsity}((x + a)^q \mod p) &\leq \prod_i (a_i + 1) \\
&= |q|_p .
\end{aligned}$$

Hence, for $r > |q|_p$, $(x + a)^q \pmod{x^r - 1, p}$ is proper support. ◀

Remark. For dependency of the form $q - p = \alpha$, where $0 < \alpha < p$, the p^{th} norm of q is $2(\alpha + 1)$. Hence, we get a natural generalization of numbers n that are good for Fermat factorization.

These observations again motivate the following algorithm.

Algorithm 2 Factoring Integer : $FAC_2(n, B)$

Require: Odd $n = pq$ ($p < q$ are primes) and a parameter $B > |q|_p$.

```
r ← 2
while r ≤ B and n is not factored do
    Choose a random number a < n
    Compute P = (x + a)^n (mod x^r − 1, n)
    Take gcd of n with ra, and with the coefficients of P.
    if n is factored then
        return factor
return 0
```

Time complexity. The overall time complexity of the algorithm is $\tilde{O}(B^2 \log^2 n)$, as in the previous subsection. Note that it is a probabilistic algorithm based on Conjecture 1. It can be seen as a natural generalization (albeit slower) of Fermat's factoring algorithm.

4.3 Relaxing conjecture 1

In the previous subsections the proofs of factoring depend on Conjecture 1. In this section we relax the conjecture; which might make it easier to prove.

The point is that we just need to prove, that for a random $a(x)$, with high probability there is a difference in the supports of the two polynomials:

$$\begin{aligned} &a(x^p)^q \pmod{x^r - 1, p} \text{ and,} \\ &a(x^q)^p \pmod{x^r - 1, q} \end{aligned} \tag{4}$$

in the case when $r > |q|_p$. The rationale is again that the first polynomial is proper support, while the second polynomial is likely to have a support *different* from the first.

▶ Conjecture 2. For primes $p < q$, $r > |q|_p$ and a random $a(x) \in (\mathbb{Z}/n\mathbb{Z})[x]/(x^r - 1, n)$, the two polynomials in Equation 4 have, with high probability, different support.

It can be seen that based on this conjecture, an algorithm similar to Algorithm 2 can be designed to factor n (in probabilistic time $\tilde{O}(B^2 \log^2 n)$).

5 General dependencies

The previous section addressed dependencies of specific forms. In this section, we move to the case of more general dependencies between the two factors. For $n = pq$, primes $p < q$, we consider a nondegenerate dependency $f(x, y)$ of degree bound d with at most γ nonzero coefficients. So, $0 = f(p, q) = \sum_{i=1}^{\gamma} c_i\, p^{\alpha_i}\, q^{\beta_i} = 0$, where $\forall i, 0 \le \alpha_i + \beta_i \le d$, $|c_i| = n^{O(d)}$. Proposition 1.1 gives the almost optimal parameters for its existence in general.

When we are given n and f, our idea is to compute AKS exponentiation (Eqn.2) in a cyclotomic ring extension over $\mathbb{Z}/n\mathbb{Z}$ and try distinguishing the two Frobenius morphisms. We give the details in the form of an algorithm and then the proof. The key step will be the computation of an expression $\prod_{i=1}^{\gamma} a(\zeta^{p^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}$, for a random element $a(\zeta)$. Note that modulo p it is the same as exponentiation by $\sum_{i=1}^{\gamma} c_i p^{\alpha_i} q^{\beta_i} = 0$. Also, note that $p^{\alpha_i - \beta_i}$ exists modulo r, when r and p are coprime.

Algorithm 3 Factoring Integer : $FAC_3(n, f)$

Require: Odd $n = pq$ ($p < q$ are primes), and a nondegenerate dependency $f = \sum_{i=1}^{\gamma} c_i x^{\alpha_i} y^{\beta_i}$, where $\forall i, 0 \leq \alpha_i + \beta_i \leq d$, $|c_i| = n^{O(d)}$.

Choose a random prime $r \leq 10\gamma \log \gamma$ and verify that $\gcd(r, n) = 1$.
for $t \in [r-1]$ **do**
  $count \leftarrow 0$
  while $count < 5 \log\log n^r$ **do**
    Choose a random element $a(\zeta) := a(x) \in \mathbb{Z}[x]/(\varphi_r(x), n)$.
    Compute $\mathcal{P} := \prod_{i=1}^{\gamma} a(\zeta^{t^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}$.
    Take gcd of n with the coefficients of $\mathcal{P}$.
    if n is factored **then**
      return *factor*
return 0

To study this algorithm we would need a qualitative conjecture about the distribution of discrete logarithms.

▶ Conjecture 3. For a fixed p, q, f as before and $R > 10\gamma$, the function $\log_{q,r} p$ takes almost random values e as we vary $r \in [R]$ such that, with a constant probability,

$$\sum_{i=1}^{\gamma} c_i \, p^{\beta_i} \, q^{e\alpha_i + (1-e)\beta_i} \neq 0 \left(\mathrm{mod} \, \frac{q^{r-1} - 1}{q - 1} \right).$$

The rationale for this conjecture is that as we vary r in a range bigger than $[\gamma]$ we expect e to be "random" enough so that the two γ-dimensional vectors $\left(c_i p^{\beta_i} \mid i \in [\gamma]\right)$ and $\left(q^{e\alpha_i + (1-e)\beta_i} \mid i \in [\gamma]\right)$ are not orthogonal $\pmod{(q^{r-1} - 1)/(q-1)}$. One necessary condition for this is: $\{e\alpha_i + (1-e)\beta_i \mid i \in [\gamma]\}$ be a set of distinct functions in e, with at least one of them being nontrivially dependent on e. The *distinctness* holds because $e\alpha_i + (1-e)\beta_i = e\alpha_j + (1-e)\beta_j$ iff $(\alpha_i, \beta_i) = (\alpha_j, \beta_j)$ iff $i = j$. (*Note :* We use that, for some i, $\alpha_i \neq \beta_i$ as f is nondegenerate.)

Now we are ready to prove the correctness of the algorithm.

Proof for Theorem 1.2. By Artin's conjecture we can deduce that we will get a prime r, with constant probability, such that: q generates the unit group of $\mathbb{F}_r$. In this case Lemma 2.1 asserts that $\varphi_r(x)$ is irreducible over $\mathbb{F}_q$. Hence, $\mathbb{F}_q[\zeta] := \mathbb{F}_q[x]/(\varphi_r(x))$ is a field.

We are interested in the iteration when the variable t equals $p \pmod{r}$. Then we can write,

$$\mathcal{P} = \prod_{i=1}^{\gamma} a(\zeta^{p^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}. \tag{5}$$

Going modulo p, and using the "first" Frobenius morphism, we get:

$$\begin{aligned} \mathcal{P} &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i p^{\alpha_i} q^{\beta_i}} \pmod{p} \\ &= a(\zeta)^{\sum_{i=1}^{\gamma} c_i p^{\alpha_i} q^{\beta_i}} \pmod{p} \quad = a(\zeta)^0 \pmod{p} \\ &= 1 \pmod{p}. \end{aligned} \tag{6}$$

Now let $e := \log_{q,r} p$. So, we can replace p with q^e in Equation 5, to get

$$\mathcal{P} = \prod_{i=1}^{\gamma} a(\zeta^{q^{e(\alpha_i - \beta_i)}})^{c_i n^{\beta_i}}.$$

Going modulo q, and using the "second" Frobenius morphism, we get:

$$\begin{aligned}\mathcal{P} &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i n^{\beta_i} q^{e(\alpha_i-\beta_i)}} \pmod q \\ &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i p^{\beta_i} q^{e\alpha_i+(1-e)\beta_i}} \pmod q \\ &= a(\zeta)^{\sum_{i=1}^{\gamma} c_i p^{\beta_i} q^{e\alpha_i+(1-e)\beta_i}} \pmod q \end{aligned} \tag{7}$$

Let us call the exponent $m := \sum_{i=1}^{\gamma} c_i\, p^{\beta_i}\, q^{e\alpha_i+(1-e)\beta_i}$.

If we can show that $a(\zeta)^m \notin \mathbb{F}_q$ then, by Equation 6, we get different supports in the polynomials $\mathcal{P} \pmod p$ and $\mathcal{P} \pmod q$. This means that step 7 would factor n. So, it suffices to ensure that $a(\zeta)^{m(q-1)} \neq 1 \pmod q$, in other words, the multiplicative order of $a(\zeta)$ in the field $\mathbb{F}_q[\zeta]$, denoted $\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta])$ satisfies:

$$\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta]) \not| \, m(q-1)\,. \tag{8}$$

From step 5 (of the algorithm) we can treat $a(\zeta)$ as a random element in $\mathbb{F}_q[\zeta]$. From the initial discussion we have that $\mathbb{F}_q[\zeta]$ is the field $\mathbb{F}_{q^{r-1}}$. From this we can estimate the probability of $a(\zeta)$ having the largest multiplicative order.

▶ Claim 5.1. $\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta]) = (q^{r-1}-1)$, with probability at least $\frac{1}{3\log\log(q^{r-1}-1)}$, when $r > 7$.

Proof. See full version of the paper. ◀

Thus, the repetitions in step 4 ensure that with a high probability we will pick a generator $a(\zeta)$ of $\mathbb{F}_q[\zeta]$. Now Equation 8 can be rewritten as:

$$\sum_{i=1}^{\gamma} c_i\, p^{\beta_i}\, q^{e\alpha_i+(1-e)\beta_i} \neq 0 \left(\mathrm{mod}\ \frac{q^{r-1}-1}{q-1}\right).$$

Conjecture 3 ensures this with high probability (for a random r). Hence, step 7 will factor with high (i.e. constant) probability.

Time Complexity. The 'for' loop of Step 2-9, runs for $r-1 = \tilde{O}(\gamma)$ times. The 'while' loop in Step 4-9, runs $O(\log\log(n^r)) = \tilde{O}(1)$ times.

The polynomial computation in step 6 is the expensive part. We would use repeated squaring and fast ring arithmetic. It multiplies γ many factors. The exponent of each factor can be bounded by An^d, so, by repeated squaring it takes $O(d\log n + \log A) = O(d\log n)$ time ($\because A := n^{O(d)}$). Also, in each step of repeated squaring there will be two polynomials multiplied in the cyclotomic ring; we can compute the product in $\tilde{O}(r\log n)$ time. Hence, step 6 takes $\tilde{O}(\gamma \cdot d\log n \cdot r\log n) = \tilde{O}(\gamma^2 d\log^2 n)$ time.

So, the overall time complexity of the Algorithm 3 is $\tilde{O}(\gamma^3 d\log^2 n)$.

◀

Clearly, the running time of the algorithm depends on the sparsity. For sparse dependency f, i.e. $\gamma = \tilde{O}(1)$, the running time becomes $\tilde{O}(d\log^2 n)$ which is only linear in d. If the given dependency has sparsity $\gamma = O(d^{1.6})$ then the running time is a much slower $\tilde{O}(d^{5.8}\log^2 n)$, but it is a simple algorithm and still faster than the known methods.

6 Alternate Analysis

In this section we present an alternate analysis and a corresponding algorithm to factor n. The algorithm presented is just a slightly modified version of Algorithm 3, and it will not need n/p to be a prime. The conjecture that our analysis relies on will be different from Conjecture 3.

Algorithm 4 Factoring Integer : $FAC_4(n, f, \mu)$

Require: Odd $n = pn'$ (prime p is not the largest prime factor q of n), and a nondegenerate (p, n') dependency $f = \sum_{i=1}^{\gamma} c_i x^{\alpha_i} y^{\beta_i}$ where $\forall i, 0 \leq \alpha_i + \beta_i \leq d$, $|c_i| \leq n^d$. Let $\mu := \sum_i e_i$ for the prime factorization $n = \prod_i p_i^{e_i}$.

$r \leftarrow 7\mu d$.
while $r \leq 10\mu d \log(d+1)$ **do**
  Choose the next prime r, and verify that $\gcd(r, n) = 1$.
  for t in range $[r-1]$ **do**
    Choose a random element $a(\zeta) := a(x) \in \mathbb{Z}[x]/(\varphi_r(x), n)$.
    Compute $\mathcal{P} := \prod_{i=1}^{\gamma} a(\zeta^{t^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}$.
    Take gcd of n with the coefficients of $\mathcal{P}$.
    if n is factored **then**
      return *factor*
return 0

To study this algorithm we will need a conjecture about discrete logarithm.

▶ Conjecture 4. For f, p, q, d, μ as before, there exists a prime $r \in [7\mu d, 10\mu d \log(d+1)]$ such that: $\mathrm{ord}_r(q) = r - 1$, $e := \log_{q,r} p < \frac{r}{2d}$ and $f(q^e, n/q^e) \neq 0$.

The rationale behind this conjecture is Artin's conjecture together with the feeling that the function $\log_{q,r} p$ should take "random" values in $\{0, \ldots, r-1\}$, in particular, values as small as $\frac{r}{2d}$. Also, since the interval is large enough we expect to get several such (r, e); one of these q^e is expected to not be a root of $f(X, n/X)$.

We now state our theorem.

Proof for Theorem 1.3. See full version of the paper. ◀

Given a sparse dependence of degree d, (small or constant γ and μ) our algorithm's performance is better than Schönhage's univariate polynomial factoring algorithm.

7 Conclusion

We initiate a new factoring idea using the AKS-type cyclotomic computation [1]. It uses the two Frobenius morphisms and we have been able to analyze it for specific families of n (based on some "reasonable" conjectures). It is a simple algorithm and, in special cases, it performs better than applying the previously known techniques. The outstanding question is what do we do when there is no dependency $f(x, y)$ readily available for n?

In this (general) case we could compute several (say, poly$(\log n)$-many) AKS-type polynomials

$$S := \{a(\zeta_r)^e \mid r, a, e \text{ carefully chosen given } n\}$$

and try to apply easy algebraic operations on S. For example, view S as a lattice generator and apply the famous LLL basis reduction algorithm on it [17]. Or, compute other linear algebra operations on S. Do these operations lead us to a factor of n ?

References

1 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Math*, 160(2):781–793, 2004.

2 Eric Bach and Jeffrey Outlaw Shallit. *Algorithmic Number Theory: Efficient Algorithms*, volume 1. MIT press, 1996.

3 Shi Bai, Pierrick Gaudry, Alexander Kruppa, Emmanuel Thome, and Paul Zimmermann. Factorization of RSA-220 with CADO-NFS. 2016.

4 Daniel Julius Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

5 Dan Boneh et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

6 Joe Peter Buhler, Hendrik Willem Lenstra Jr, and Carl Pomerance. Factoring integers with the number field sieve. In *The development of the number field sieve*, pages 50–94. Springer, 1993.

7 Yingpu Deng and Yanbin Pan. An algorithm for factoring integers. Cryptology ePrint Archive, Report 2012/097, 2012.

8 John D Dixon. Asymptotically fast factorization of integers. *Mathematics of computation*, 36(153):255–260, 1981.

9 Carl Friedrich Gauss. *Disquisitiones Arithmeticae*. 1801. Article 329.

10 Joseph Gerver. Factoring large numbers with a quadratic sieve. *Mathematics of Computation*, 41(163):287–294, 1983.

11 Rajiv Gupta and Maruti Ram Murty. A remark on artin's conjecture. *Inventiones mathematicae*, 78(1):127–130, 1984.

12 F.R.S. Horsley, Rev. Samuel. The sieve of eratosthenes. being an account of his method of finding all the prime numbers. *Philosophical Transactions (1683-1775)*, 62:327–347, 1772.

13 Ravi Kannan. Algorithmic geometry of numbers. *Annual review of computer science*, 2(1):231–267, 1987.

14 Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Klaas Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter Lawrence Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology–CRYPTO'10*, pages 333–350. 2010.

15 R Sherman Lehman. Factoring large integers. *Mathematics of Computation*, 28(126):637–646, 1974.

16 Arjen Klaas Lenstra, Hendrik Willem Lenstra Jr., Mark Steven Manasse, and John M. Pollard. The number field sieve. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, pages 564–572, 1990.

17 Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

18 Hendrik Willem Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

19 Calvin T Long. *Elementary introduction to number theory*. Prentice Hall, 1987.

20 James McKee. Turning euler's factoring method into a factoring algorithm. *Bulletin of the London Mathematical Society*, 28(133):351–355, 1996.

21 Pieter Moree. Artin's primitive root conjecture—a survey. *INTEGERS*, 10(6):1305–1416, 2012.

22 Oystein Ore. *Number theory and its history*. Courier Corporation, 2012.

23 John M Pollard. Theorems on factorization and primality testing. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 76 of *Cambridge Univ Press*, pages 521–528, 1974.

24 John M Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

25 Carl Pomerance. The quadratic sieve factoring algorithm. In *Advances in cryptology*, pages 169–182, 1985.

26 Carl Pomerance. A tale of two sieves. *Biscuits of Number Theory*, 85, 2008.

27 Arnold Schönhage. Factorization of univariate integer polynomials by diophantine approximation and improved basis reduction algorithm. *ICALP*, 172:436–447, 1984.

28 Peter Williston Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

29 Lawrence Clinton Washington. *Introduction to cyclotomic fields*, volume 83. Springer, 2012.

30 Hugh Cowie Williams. A $p+1$ method of factoring. *Mathematics of Computation*, 39(159):225–234, 1982.

31 Hugh Cowie Williams and Jeffrey Outlaw Shallit. Factoring integers before computers. *Mathematics of computation*, 48:481–531, 1994. (1943-1993, Fifty Years of Computational Mathematics (W. Gautschi, ed.), Proc. Sympos. Appl. Math.).

A Proofs

A.1 Claim 5.1

▶ Claim. $\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta]) = (q^{r-1}-1)$, with probability at least $\frac{1}{3\log\log(q^{r-1}-1)}$, when $r > 7$.

Proof. Clearly, the number of generators in $\mathbb{F}_q[\zeta]^*$ is $\varphi(q^{r-1}-1)$, using the Euler's totient function [19, p.85]. Hence, if we choose a random element $a(\zeta)$ in $\mathbb{F}_q[\zeta]$ then,

$$\begin{aligned}\Pr\left[\, a(\zeta) \text{ generates } \mathbb{F}_q[\zeta]^* \,\right] &= \frac{\varphi(q^{r-1}-1)}{(q^{r-1}-1)} \\ &> \frac{1}{3\log\log(q^{r-1}-1)}, \quad \text{for } r > 7.\end{aligned}$$

The last line follows from the result: $\frac{\varphi(n)}{n} > (1.79\log\log n + 3/\log\log n)^{-1}$ for $n > 2$, see [2, thm.8.8.7]. ◀

A.2 Theorem 1.3

▶ Theorem. For an integer $n = p \cdot n'$ (where p is a prime smaller than the largest prime factor q of n), given a nondegenerate (p, n') dependency of the form $f(X,Y) = \sum_{i=1}^{\gamma} c_i X^{\alpha_i} Y^{\beta_i}$, where $\forall i,\ 0 \le \alpha_i + \beta_i \le d$, $|c_i| \le n^d$, Algorithm 4 factors n in $\tilde{O}(\mu^3 \cdot \gamma d^4 \log^2 n)$ time. Here $\mu := \sum_i e_i$ for the prime factorization $n = \prod_i p_i^{e_i}$. (Assuming Artin's conjecture & 4.)

Proof. From the proof of Theorem 1.2 we know that, when $t = p \pmod r$ in step 6, then $\mathcal{P} = 1 \pmod p$.

Consider the prime factor q. Let $e := \log_{q,r} p$. From the "second" Frobenius morphism we get:

$$\begin{aligned}\mathcal{P} &= \prod_{i=1}^{\gamma} a(\zeta^{q^{e(\alpha_i-\beta_i)}})^{c_i n^{\beta_i}} \\ &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i q^{e(\alpha_i-\beta_i)} n^{\beta_i}} \pmod q \\ &= a(\zeta)^{\sum_{i=1}^{\gamma} c_i q^{e(\alpha_i-\beta_i)} n^{\beta_i}} \pmod q\,. \end{aligned} \tag{9}$$

Let us call the exponent $m := \sum_{i=1}^{\gamma} c_i q^{e(\alpha_i-\beta_i)} n^{\beta_i}$.

Using the r promised by Conjecture 4 we can upper bound m as:

$$\begin{aligned}
0 < |m| &= \Big|\sum_{i=1}^{\gamma} c_i q^{e(\alpha_i-\beta_i)} n^{\beta_i}\Big| \\
&< \sum_{i=1}^{\gamma} n'^{2d}\ q^{e(\alpha_i-\beta_i)}\ n^{\beta_i} \qquad [\text{Bound } c_i] \\
&< \sum_{i=1}^{\gamma} q^{2(\mu-1)d}\ q^{r/2}\ q^{\mu\beta_i} \\
&< q^{3\mu d + r/2} \\
&< \frac{q^{r-1}-1}{q-1}\,.
\end{aligned}$$

Thus, $0 < |m| \cdot (q-1) < q^{r-1} - 1 = |\mathbb{F}_q[\zeta]^*|$. Consequently, the set $\{a(\zeta) | a(\zeta)^{m(q-1)} = 1\}$ is a proper subgroup of $\mathbb{F}_q[\zeta]^*$, and so on randomly picking $a(\zeta)$ we will get, with high probability, $\mathcal{P} \notin \mathbb{F}_q$.

Thus, step 7 is likely to factor n.

Time Complexity. In Algorithm 4, the while loop in step 2-9, runs $\tilde{O}(\mu d)$ times. The for loop in Step 4-9, runs $O(r) = \tilde{O}(\mu d)$ times.

The expensive step is the AKS polynomial computation in step 6. We would use repeated squaring and fast ring arithmetic. It multiplies γ many factor polynomials, where each polynomial has exponent bounded by $n^{O(d)}$, so, by repeated squaring it takes $O(d \log n)$ time. Also, in each step of repeated squaring, there will be two polynomials multiplied, which requires $\tilde{O}(r \log n)$ ring operations. Hence, step 6 would take $\tilde{O}(\gamma \cdot d \log n \cdot \mu d \log n)$ $= \tilde{O}(\gamma \mu d^2 \log^2 n)$ time.

So, the overall time complexity of Algorithm 4 is $\tilde{O}(\gamma \mu^3 d^4 \log^2 n)$. ◀