Breaking RSA May Be As Difficult As Factoring

Daniel R. L. Brown

October 24, 2005

Abstract

If factoring is hard, then straight line programs cannot efficiently solve the low public exponent RSA problem. More precisely, no efficient algorithm can take an RSA public key as input and then output a straight line program that efficiently solves the low public exponent RSA problem for the given public key — unless factoring is easy. More generally, it suffices that the public exponent has a small factor. Of course, other kinds of algorithms may well be able to solve the RSA problem.

1 Introduction

A long standing open question in cryptology is whether the RSA problem can be solved without factoring. A partial answer is that the low public exponent RSA problem cannot be solved with a straight line program, if factoring is hard. More generally, the same holds if the public exponent has a small factor.

Recall that a straight line program is an algorithm that does a fixed sequence of calculations. No branching or looping is allowed. The sequence of calculations is the same regardless of the results of the calculations. The allowed operations are addition, subtraction and multiplication, thus straight line programs always compute polynomials functions. The secret step of RSA decryption can be implemented with a straight line program.

Naturally, continued research is needed to determine if the RSA problem is equivalent to factoring for all algorithms, not just straight line programs. Although the result for straight line programs is a small step towards answering this open question, it may prove a dead end. Inverses modulo an RSA modulus can be calculated using the Euclidean algorithm, and this is not believed to reveal the factorization of the RSA modulus. Typical straight line programs for calculating for inverses, however, would reveal the factorization. Algorithms similar the Euclidean algorithm for inversion may exist for solving the RSA problem that neither use a straight line program nor reveal the factorization.

1.1 Related Work

Revealing an RSA private exponent[1] reveals the factorization of a RSA modulus. Rivest and Kaliski [RK], in their concise summary of the RSA problem, attribute this well known and fundamental result to de Laurentis [dL84] and Miller [Mil76]. A private exponent can be used to build a straight line program for solving the RSA problem. Conversely, however, it is not clear that a straight line

[1] Many different private exponents correspond to a single public exponent.

program for solving the RSA problem reveals a private exponent, so the result in this paper is not immediate from [dL84, Mil76].

The main result discussed in this paper is surprisingly simple. Indeed, chances are that it has already been published, or at least discovered and somewhat circulated. Some of the best surveys [Bon99, RK] of the RSA problem, however, do not mention such a result. Although arguably not of major importance, in that [dL84, Mil76] covers most of this ground, the result seems important enough to be made more widely known. Anybody who has already published, or even previously discovered and communicated, this result may be duly credited in an update of this paper.

Boneh and Venkatesan [BV98][2] made a breakthrough result that goes towards showing solving the RSA problem may be easier than factoring. They prove that one cannot prove, with a straight line program, that the solving the RSA problem is as hard as factoring. Although both results involve straight line programs, and the two results take steps in opposite directions regarding the question of the equivalence of factoring and the RSA problem, the results do not contradict each other.

Okamoto and Uchiyama [OU98] give a proof that shows the order of elliptic curve group defined over $\mathbb{Z}_n$ generally reveals the factorization of n. They use twists of elliptic curves. The result has [OU98] has a role in this discovery of the result here, as outlined in §1.2.

1.2 Realization of the Result

Although the result may well have already been published or otherwise known, it was not known to the author. The result seemed interesting, if not important, enough to write it up and release it, even if it had already been published, on the chance a few others, ignorant like him, may become aware of this interesting result. Recounts here is how the author independently came by this result, much by accident, and how Steven Galbraith was instrumental to this discovery through kindly pointing out and discussing errors in the author's previous work.

The author, misinterpreting[3] [OU98], submitted ePrint 2005/208, in which an elliptic curve order over $\mathbb{Z}_n$ was used to make possible efficient inversion of certain polynomials over $\mathbb{Z}_n$. The inversion algorithm used straight line programs based on elliptic curve arithmetic. Galbraith then corrected the author about how the proof in [OU98] worked, noting that the author's inversion algorithm also revealed the factorization, whereas the author had thought otherwise. The author attempted to correct the error by working over extension rings of $\mathbb{Z}_n$. Galbraith foiled these attempts by again applying twists as done [OU98]. The author attempted to iterate the process, effectively extending to a degree with 2^m for some large m. This seemed to work, albeit much less effectively, until Galbraith noted that a third degree extension could be used instead of twist, at which point, the whole endeavour seemed doomed.

The author hoped that obfuscation of inversion algorithm could be used to salvage something from the erroneous ePrint, but then realized that Galbraith's strategy worked provided that the inversion algorithm was specified as a straight line program. Therefore, obfuscation was no barrier to fundamental flaw (unless obfuscation hid the actual ring operations of addition, subtraction and multiplication). The author realized this to be an impossibility result: certain polynomials arising from elliptic curve arithmetic apparently could not be inverted with a straight line program, unless factoring was easy. The result seemed, however, too specialized to be of interest, because the polynomials were not directly related to RSA, or any other cryptosystem in use.

[2] Which this paper takes its title from.

[3] The author had thought the proof worked by searching for a smooth order curve.

Upon further thought, the author realized that extension rings could used to factor, given a straight line program for solving the RSA problem, at least if the public exponent were low enough. The degrees needed for the extension rings were actually much lower than for the elliptic curve polynomials, so the reduction was much tighter. This seemed to be such a simple and important idea of a result that the author thought it worth writing it up in initial form and posting it to the ePrint server, even if it were wrong or already known. An early revision includes the observation that it suffices for the public exponent to have a small factor.

2 Straight Line Programs

Straight line programs are a class of algorithms that do not branch or loop, and whose steps are just addition, subtraction or multiplication.

Definition 1. *A* straight line program *of length L is a sequence*

$$P = ((i_1, j_1, \circ_i), \ldots, (i_L, j_L, \circ_L)) \tag{1}$$

of triples, such that $-1 \leqslant i_k, j_k < k$ and $\circ_k \in \{+, -, \cdot\}$. On input x, program P computes an output $P(x)$ as follows.

1. *Let $x_{-1} = 1$.*
2. *Let $x_0 = x$.*
3. *For $1 \leqslant k \leqslant L$, compute $x_k = x_{i_k} \circ_k x_{j_k}$.*
4. *Output $P(x) = x_L$.*

Let R be a ring[4]. If $x \in R$, then the output $P(x) \in R$. An important ring for this paper is the the ring $\mathbb{Z}/\langle n \rangle$ of integers modulo n, where n is the product of two large primes. This is the type of ring over which the RSA problem is defined. Various other rings will be used in this paper.

The ring $\mathbb{Z}[X]$ of integer polynomials over a indeterminate X is useful for classifying straight line programs. The ring $\mathbb{Z}$ has a natural embedding in any ring R (there is a unique homomorphism), and similarly the ring $\mathbb{Z}[X]$ has a unique embedding in $R[X]$ such that X maps to X. Thus, $f(r)$ makes sense for any $f(X) \in \mathbb{Z}[X]$ and any $r \in R$, for any ring R. The polynomial $P(X) \in \mathbb{Z}[X]$ determines the action of P in any ring:

Lemma 1. *If $P(X) = f(X) \in \mathbb{Z}[X]$, then $P(r) = f(r)$ for any $r \in R$ and any ring R.*

Proof. Apply the natural embedding of $\mathbb{Z}[X]$ into $R[X]$. □

The straight line program P is a method to compute $f(r)$ from r. Many different straight line programs may compute $f(r)$ from r, and other kinds of algorithm may also compute $f(r)$ from r.

The length of P is a simple measure of its efficiency. A secondary measure of efficiency, memory usage, will not be considered here. The degree of the polynomial $f(X)$ that P computes is at most 2^L. Thus, there are polynomials of rather high degree can computed efficiently with a straight line program. The following lemma is crucial for the theorems in this paper.

Lemma 2. *Let R be a finite ring, let $p(X), q(X) \in \mathbb{Z}[X]$, and let $\sigma : R \to S$ be a surjective homomorphism. If $p(q(r)) = r$ with probability μ for random $r \in R$, then $p(q(s)) = s$ with probability at least μ for random $s \in S$.*

[4]We assume rings have a multiplicative identity 1.

Proof. Given a random $s \in S$, choose a random $r \in R$, such that $s = \sigma(r)$. Over random s, the resulting r is uniformly random over R. With probability μ, we have $p(q(r)) = r$ and

$$\begin{aligned} p(q(s)) &= p(q(\sigma(r))) \\ &= p(\sigma(q(r))) \\ &= \sigma(p(q(r))) \\ &= \sigma(r) \\ &= s, \end{aligned} \tag{2}$$

using the fact that homomorphisms commute with polynomials. □

If $p(q(r)) \neq r$, which happens with probability $1-\mu$, it may be still be the case that $p(q(s)) = s$, so we can only get a lower bound of μ on the probability that $p(q(s)) = s$.

3 Factoring with a Straight Line Program RSA Problem Solver

When the RSA public exponent e is small enough, one can use an efficient SLP for the RSA private key operation to efficiently factor the RSA modulus. The case of $e = 3$ is especially simple, so is described first. Larger e involves a more detailed case analysis, but follows the same principles.

3.1 Cube Roots

Theorem 3. *Let $f(X) \in \mathbb{Z}[X]$, let p and q be primes with $p \equiv q \equiv 2 \mod 3$, let $n = pq$ and let $R = \mathbb{Z}/\langle n \rangle$. Suppose that $f(X)$ is efficiently computable with a straight line program F of length L, and that for random $r \in R$, the probability that $f(r^3) = r$ is μ. Then n can be factored with a probability of success at least $\frac{2}{3}\mu^2$, using a straight line program of length $7L + K$ running over R together a small amount of other work, for some constant K.*

Proof. Pick a random $u \in R$, until one is found with $\left(\frac{u}{n}\right) = -1$. Without loss of generality, assume that

$$\left(\frac{u}{p}\right) = 1 \quad \text{and} \quad \left(\frac{u}{q}\right) = -1. \tag{3}$$

Let $U = R[X]/\langle X^2 - u \rangle$, which is the desired quadratic extension of R. The ring U has structure:

$$U \cong \mathbb{F}_p \times \mathbb{F}_p \times \mathbb{F}_{q^2}. \tag{4}$$

To see this, suppose that $v^2 = u$ in $\mathbb{F}_p$. Let ψ be the isomorphism that maps $a + bX \in U$, to $(a + bv, a - bv, a + bX) \in \mathbb{F}_p \times \mathbb{F}_p \times \mathbb{F}_{q^2}$, where the integers a and b are reduced modulo the appropriate modulus. Elements of U that map to $(s, 0, 0)$ form a subring $S \cong \mathbb{F}_p$, and elements mapping to the form $(0, \bar{s}, 0)$ form a subring $\bar{S} \cong \mathbb{F}_p$. Elements of S can also be characterized as elements $a + bX$ of U such that $a = b$ and $q \mid a$, while elements of S' can be characterized as those with $a = -b$ and $q \mid a$. Elements of U that map to $(0, 0, t)$ form a subring T, which can also be characterized as those elements $a + bX$, with $p \mid a, b$.

Because $R \cong \mathbb{F}_p \times \mathbb{F}_q$, there are surjective homomorphisms $\sigma : R \to S$ and $\bar{\sigma} : R \to \bar{S}$. Lemma 2 then implies that $f(s^3) = s$ with probability at least μ for any random $s \in S$, and similarly if $s \in \bar{S}$.

Now pick a random $r \in U$, and compute $f(r^3)$, using straight line program F, which can be done by Lemma 1. Suppose that $\psi(r) = (s, \bar{s}, t)$. Because ψ is an isomorphism, we have

$$\psi(f(r^3)) = (f(s^3), f(\bar{s}^3), f(t^3)) \tag{5}$$

With probability μ^2, we have $f(s^3) = s$ and $f(\bar{s}^3) = \bar{s}$. In this event, we have:

$$\psi(f(r^3) - r) = (0, 0, f(t^3) - t). \tag{6}$$

If $f(t^3) \neq t$, then $f(r^3) - r \in T \setminus \{0\}$, and thus has the form $a + bX$ where $p \mid a, b$ and one of a or b is nonzero. Therefore

$$p \in \{\gcd(a, n), \gcd(b, n)\} \tag{7}$$

and n has been factored.

Otherwise $f(t^3) = t$. We claim that this can happen with probability at most $\frac{1}{3}$. To see this, note that $q^2 \equiv 1 \mod 3$, so that $3 \mid q^2 - 1$. Hence $T \cong \mathbb{F}_{q^2}$ has an element ω of multiplicative order 3. It follows that, in T, only one third of elements are perfect cubes, and each perfect cube has three cube roots. These cube roots are called conjugates, the set of them always takes the form $\{z, \omega z, \omega^2 z\}$, which is called a conjugacy class.

For a random $t \in T$ with a given value of t^3, each element of the conjugacy class has probability $\frac{1}{3}$ of occurring. Given only t^3, there is at most $\frac{1}{3}$ chance of determining t, no matter what algorithm is used.

Therefore, with one run of F on the ring U, we have a probability of $\frac{2}{3}\mu^2$ of obtaining a factor of n. Running F on the ring U, can also be implemented as a straight line program G running on the ring R, with length at most $7L + K$, which can be seen by the fact that a multiplication in U can be implemented as seven operations in R, because $(a + bX)(c + dX) = (ac + bdu) + (cb + ad)X$. □

A toy example may help illustrate. Any straight line program F for polynomial X^7 finds cube roots in $R = \mathbb{Z}/\langle 55 \rangle$. The ring $U = \mathbb{Z}[X]/\langle 55, X^2 - 6 \rangle$ is isomorphic to $\mathbb{F}_5 \times \mathbb{F}_5 \times \mathbb{F}_{121}$. For a random element of U, we can pick $r = 4 + 7X$. Then we computed $w = r^3 = 17 - 26X$, and submit w to the straight line program F, which gives $z = F(w) = w^7 = 9 + 17X$. Now $F(r^3) - r = z - r = 5 + 10X$. As predicted, $z - r$ is $0 + 0X \mod 5$, and happens to be nonzero in U (which should happen with probability at least $\frac{2}{3}$). In this case, computing $\gcd(55, c) = 5$, where c is either coefficient of $z - r$, recovers the desired factor of $n = 55$.

The factor μ^2 in the success rate of factoring can be improved to μ. The proof above of Theorem 3 is a simplification that do not exploit the full strength of the reduction.

Theorem 4. *Theorem 3 is also true with a success rate of the factoring algorithm of at least $\frac{2}{3}\mu$.*

Proof. The proof is the same as the proof of Theorem 3, except that we only use the event $f(s^3) = s$, so that it does not matter if $f(\bar{s}^3) \neq \bar{s}$. If $f(s^3) = s$ and $f(t^3) \neq 0$, then

$$\psi(f(r^3) - r) = (0, y, z) \tag{8}$$

for some $y \in \mathbb{F}_p$ and $0 \neq z \in \mathbb{F}_{q^2}$. Write $r = a + bX$, and let $\bar{r} = a - bX$. Then clearly $\psi(\bar{r}) = (y, 0, \bar{z})$ for some $\bar{z} \in \mathbb{F}_{q^2}$. The norm of r is $r\bar{r} = a^2 - b^2 u$, and:

$$\psi(a^2 - b^2 u) = \psi(r\bar{r}) = (0, y, z)(y, 0, \bar{z}) = (0, 0, z\bar{z}). \tag{9}$$

Therefore, $p \mid a^2 - b^2 u$ and $q \nmid a^2 - b^2 u$, because $z \neq 0$ implies $\bar{z} \neq 0$. □

This improvement in the success rate of factoring is important, because otherwise for small but non-negligible μ, the factoring algorithm could have a negligible success rate.

Theorem 5. *Let A be a probabilistic algorithm which on input n of an RSA number of given size with public exponent of three, outputs an efficient straight line program F that finds cube roots modulo n with probability at least μ. Then A can be used to factor RSA numbers of the given size, with probability at least $\frac{2}{3}\mu$, and with similar efficiency to A and the efficiency of the straight line program it outputs.*

Proof. To factor n, run algorithm A, then apply Theorem 4 to its output program. □

Provided that μ is not unreasonably small, then one can factor efficiently, if A is efficient. The success rate of the factor algorithm can be increased, by repeating it, or by using random self-reducibility of the RSA problem to first increase the success rate of A. Increase of the success rate in this manner costs extra computation time in the usual trade-off.

3.2 Higher Degree Roots

We generalize the result to higher public exponents.

Theorem 6. *Let $f(X) \in \mathbb{Z}[X]$, let $e > 3$ be an integer, let p and q be primes with $\gcd(e, (p-1)(q-1)) = 1$, let $n = pq$ and let $R = \mathbb{Z}/\langle n \rangle$. Suppose that $f(X)$ is efficiently computable as a straight line program F of length L, and for random $r \in R$, the probability that $f(r^e) = r$ is μ. Then n can be factored with an approximate probability of success at least $\frac{e-1}{e\phi(e)E}\mu$, where E is the base of the natural logarithm, using a straight line program of length at most about $3\phi(e)^2 L + K$ running over R together a small amount of other work for some constant K depending on e and R.*

Proof. There are two phases to the factoring algorithm. In the first phase, a random polynomial $g(X) \in R[X]$ of degree $\phi(e)$ is selected. The second phase uses the resulting $g(X)$, generalizes the previous proofs, and is successful if $g(X)$ has a root modulo p and is irreducible modulo q (or vice versa). After presenting the second phase, we analyze the probability of this condition on $g(X)$.

Given such a $g(X)$, factoring proceeds almost exactly as in the proof of Theorem 4. Let:

$$U = \mathbb{Z}[X]/\langle n, g(X) \rangle \tag{10}$$

Because of the property of $g(X)$, the ring U has structure:

$$U \cong \mathbb{F}_p \times \mathbb{F}_{p^{d_2}} \times \cdots \times \mathbb{F}_{p^{d_s}} \times \mathbb{F}_{q^{\phi(e)}} \tag{11}$$

Let S be a subring of U isomorphic to $\mathbb{F}_p$ and let T be the subring isomorphic to $\mathbb{F}_{q^{\phi(e)}}$. Note that T^* has $q^{\phi(e)} - 1$ elements, and that $e \mid q^{\phi(e)} - 1$, so that a fraction $\frac{1}{e}$ of elements of T are perfect e^{th} powers, and that every such perfect power has exactly e roots forming a conjugacy class.

Pick a random $r \in U$. Compute $F(r^3)$. Let s and t be the homomorphic projections of r in components S and T. Then $F(s^3) = s$, because $S \cong \mathbb{F}_p$ and $\mathbb{F}_p$ is the homomorphic image of R, where F computes e^{th} roots, so Lemma 2 applies. Hence $F(r^3) - r$ projects to 0 in S. Let $F(r^3) - r = z_0 + z_1 X + \cdots + z_{\phi(e)-1} X^{\phi(e)-1} = z(X)$. In the proof of Theorem 4, a norm was calculated. The generalization needed here is the resultant:

$$\text{Res}(z(X), g(X)) \tag{12}$$

The resultant is the product of all the differences between roots of the first and second polynomials, and can be computed efficiently using a determinant. Let s be the root of $g(X)$ in $\mathbb{F}_p$ that was assumed to exist. A polynomial $u(X) \in \mathbb{Z}[X]$ regarded as an element of U projects to the subring S as $u(s)$. Since $z(X)$ projects to 0 in S, we have $z(s) = 0$ in S. Therefore, $z(X)$ and $g(X)$ have a common root s in S. Thus the resultant projects to zero in S. But the resultant is a polynomial of degree zero, and is thus an element of $\mathbb{Z}/\langle n \rangle$. Being an integer and belonging to S, implies being divisible by p. Therefore:

$$p \mid \gcd(n, \mathrm{Res}(z(X), g(X))) \tag{13}$$

With probability at most $\frac{1}{e}$ this gcd is n, which corresponds to F having guessed correctly which of the e conjugates t was. Therefore, with probability at least $\frac{e-1}{e}$, the gcd is p and n has been factored.

In the first phase, a random monic polynomial $g(X) \in R[X]$ of degree $d = \phi(e)$ was selected. We now calculate the probability of the polynomial having a root or being irreducible in the field $\mathbb{F}_p$, to determine the success rate of the second phase. The total number of monic polynomials of degree d is p^d. The number of irreducible polynomials of degree d is:

$$\frac{1}{d} \sum_{f|d} \mu\left(\frac{f}{d}\right) p^f, \tag{14}$$

where $\mu(\cdot)$ is the Möbius function. This can be seen by applying the inclusion-exclusion principle to the degrees of elements in extension fields of $\mathbb{F}_p$. For large p, the probability of being irreducible is thus approximately $\frac{1}{d}$. The number of $g(X)$ with at least one root is:

$$\sum_{f=1}^{d} (-1)^f \binom{p}{f} p^{d-f}. \tag{15}$$

This can be seen by the inclusion-exclusion principle on the set of roots. Therefore, for large p, the probability of having a root in $\mathbb{F}_p$ is approximately $\frac{1}{E}$, where E is the base of the natural log (not to be confused the RSA public exponent), with a better approximation for larger d.

The straight line program F when run over U can be implemented as a longer straight line program G run over R. Each multiplication step in F involves at most about $2\phi(e)^2$ multiplication steps in G and $\phi(e)^2$ addition steps. □

To increase the success rate of the factoring algorithm, one can repeat the process. A better improvement may be possible, however, with a more judicious selection of $g(X)$. For example, to increasing the chance that $g(X)$ is irreducible may be possible by selecting $g(X)$ to be irreducible over the integers. It is not clear, however, when doing so, what the probability of having a root is. Alternatively, one may select $g(X) = X^d - u$, with u random. The factorization of such binomials is fairly well understood: it depends on the field size and the order of u in the field. Such polynomials are never irreducible over $\mathbb{F}_p$ if $4 \mid t$ and $p \equiv 3 \mod 4$, but otherwise they can be irreducible for certain choices of u. This approach has the potential to increase the probability of finding $g(X)$ by preprocessing u through computation of higher degree equivalents of the Jacobi symbol, resorting to higher degree equivalents of quadratic reciprocity.

One may be able use smaller extension degrees than used in the proof of Theorem 6. For example, if algorithm F fails to find e^{th} roots in $\mathbb{F}_{p^2}$ or $\mathbb{F}_{q^2}$, even though unique e^{th} roots exist in both this fields, it is sufficient to work in a quadratic extension. In the proof, we cannot make such

an assumption, so we use an extension of higher degree. It is possible to devise a factoring strategy that tries a quadratic extensions first, then extensions of degree of successive higher factors $d \mid \phi(e)$, which would probably succeed in factoring more often (or quickly, in iterated form) except in the worst case.

Theorem 7. *Let A be a probabilistic algorithm that, on input n of an RSA number of given size with public exponent of a fixed e, outputs an efficient straight line program F that finds e^{th} roots modulo n with probability at least μ. Then A can be used to factor RSA numbers of the given size, with probability at least $\frac{e-1}{e\phi(e)E}\mu$ where E is the base of the natural logarithm, and with similar cost to the cost of A plus the $3\phi(e)^2$ times cost of the straight line program it outputs.*

Proof. To factor n, run algorithm A, then apply Theorem 6 to its output program. □

In other words, using straight line programs to solve the RSA problem for public exponent e is at most about $3E\phi(e)^3$ times easier than factoring (using an iterated strategy).

With the commonly used public exponent $e = 2^{16} + 1$, key size $n \approx 2^{1024}$, and standard estimate that factoring costs the equivalent of about 2^{80} operations in $\mathbb{Z}/\langle n\rangle$ for this key size, then the estimated lower bound on the difficulty of solving the associated RSA problem is about 2^{30} operations in $\mathbb{Z}/\langle n\rangle$. This very loose estimate may be made more precise by more careful accounting of the proofs, (and perhaps it can be improved as well, with some optimization of the proof algorithms, such as Karatsuba). It must be emphasized the actual difficulty may be much higher than the lower bounds proven here, or much lower if general algorithms are used.

3.3 Exponents with a Small Factor

If the public exponent e has a small factor d, then any algorithm for finding e^{th} roots can be used to find d^{th} roots, simply by calculating the e^{th} root and then exponentiating by $\frac{e}{d}$. Therefore, the theorems above actually generalize to when the public exponent is any multiple of the low value stated.

Note that the smaller the smallest factor of an RSA public exponent, then the tighter the known security reductions are between the RSA problem and factoring. With a smallest factor of two, the well known reductions [dL84, Mil76] are very tight and are not limited to straight line programs. With a smallest factor of three, the reduction described here is quite tight, but limited to straight line programs. As the smallest factor gets larger, however, the tightness diminishes.

Because there are various security concerns about low public exponent RSA, such as the attacks of Coppersmith and Hastad and the theoretical work of Boneh and Venkatesan [BV98], it is natural to doubt the general equivalence between the low public exponent RSA problem and factoring, that is, when algorithms other than straight line programs are allowed. It remains prudent to use a moderately large public exponent. Indeed, it is prudent to use a public exponent that is not product of small exponents; for otherwise, if the RSA problem is solvable for each of the small exponents, then it is solvable for their product.

To best take advantage of the results here about low public exponents, one may use an exponent of a form similar to $e = 3(2^{16} + 1)$. Computing e^{th} is at least as difficult as computing cube roots, and thereby the results describe in this paper provide some assurance, however limited it may be, of the hardness of the RSA problem. Conversely, computing e^{th} roots is as difficult as computing $(2^{16} + 1)^{\text{th}}$ roots, so this choice of e is at least as secure as the exponent $2^{16} + 1$, which is in widespread use today. Public exponent $3(2^{16} + 1)$ is only slightly more expensive to implement than $2^{16} + 1$, so the cost of extra security benefit may be low enough to warrant such a practice.

4 Limitations

The result seems to require the public exponent, or one of its factors, to be quite low, or else the extension degree gets quite large, and factoring with this result becomes much slower than solving the RSA problem. A bit surprisingly, past work has generally shown security concerns with low public exponent RSA. This result in no way undoes past work, so this result should never be used to justify disregard the security concerns with low public exponent RSA.

Unlike [BV98], this result does not seem to be a breakthrough, given that the only known straight line programs for solving the RSA problem, involve a decryption exponent. Chances are that such a straight line program would reveal the decryption exponent, unless it were very well obfuscated. If the decryption exponent were revealed by the straight line program, then known results [dL84, Mil76] reveal the factorization, so the result accomplishes nothing new. Obfuscation seems to be the main new hurdle that this result overcomes. Furthermore, unlike known results [dL84, Mil76], this result is limited to public exponents with small factors.

Some possibility remains, however, that there are straight line programs for solving the RSA problem that other than by applying a private exponent. For example, in a finite field, Cipolla's algorithm can be regarded as a probabilistic straight line program for finding square roots with success rate of about $\frac{1}{2}$, and Cipolla's algorithm can be generalized to cube roots.

Note that finding inverses in $R = \mathbb{Z}/\langle n \rangle$ can be done efficiently using the Euclidean algorithm. This does not require the factorization of n, nor is it generally believed that the existence of the Euclidean algorithm helps in any way to factor n. Straight line programs for computing inverses in R, however, typically compute a polynomial $X^{k\phi(n)-1}$. If one can extract the exponent from the program, then one can then factor n, if k is small enough. Moreover, it may be possible to extend the results here to show that any straight line program for computing inverses in $\mathbb{Z}/\langle n \rangle$ can be used to factor n. Preliminary attempts to do this involve using an extension whose degree grows with the length of the straight line program, and if this works out, it is likely to be a far weaker result than the results about the RSA problem. Jacobi symbols can also be computed efficiently with an algorithm similar in nature to the Euclidean algorithm, and it may be well the case that straight line programs for computing the Jacobi symbol reveal the factorization too.

Efficient straight line programs can only evaluate a very small proportion of all integer polynomials. The number of straight line programs of length L is $3^L(L+1)!^2$. Consider the field $\mathbb{F}_p$ with $p \approx 2^{512}$. The number of polynomials is $p^p \approx 2^{2^{521}}$. In the context of RSA factorization, we may consider a straight line program to be efficient if $L \leqslant 2^{80}$. The number of such straight line programs is quite a bit less than $2^{2^{88}}$. Certain integer polynomial may not be efficiently computable with a straight line program, but may be computable by other algorithms. If so, they may be useful for solving the RSA problem, and this paper shows nothing to the contrary.

Not all functions in an RSA ring $R = \mathbb{Z}/\langle n \rangle$ can be computed with a polynomial. Recall that all functions from a field to itself can be computed with a polynomial. Only a negligible proportion of functions $f : R \to R$ can be expressed as polynomial functions. If $n = pq$, then the number of polynomial functions is $p^p q^q$, which is considerably smaller than the total number of functions, which is $(p+q)^{p+q}$. The probability that a random function is a polynomial is thus approximately $2^{-2\sqrt{n}}$. Non-polynomial functions can be very simple: the function $f(x) = \lfloor \frac{2x}{n} \rfloor$, where $0 \leqslant x < n$ cannot be expressed as a polynomial. It may be the case that efficient non-polynomial functions can be used solve the RSA problem in ways that are not possible in fields.

5 Conclusion

Solving the low public exponent RSA problem with a straight line program (even one that depends on the RSA public key) is as difficult as factoring. If factoring is hard, then no algorithm can output a straight line program that solves the RSA problem efficiently, provided the public exponent has a small enough factor.

Acknowledgements

I am much obliged to Steven Galbraith for his edifying conversations.

References

[Bon99] Dan Boneh, *Twenty years of attacks on the RSA cryptosystem*, Notices of the American Mathematical Society **46** (1999), no. 2, 203–213, `http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html`.

[BV98] Dan Boneh and Ramarathnam Venkatesan, *Breaking RSA may be easier than factoring*, in Nyberg [Nyb98], `http://crypto.stanford.edu/~dabo/abstracts/no_rsa_red.html`, pp. 59–71.

[dL84] J. M . de Laurentis, *A further weakness in the common modulus protocol for the RSA cryptoalgorithm*, Cryptologia **8** (1984), 253–259.

[Mil76] Gary L. Miller, *Riemann's hypothesis and test for primality*, Journal of Computer and Systems Science **13** (1976), no. 3, 300–317.

[Nyb98] Kaisa Nyberg (ed.), *Advances in cryptology — EUROCRYPT '98*, LNCS, no. 1403, IACR, Springer, May 1998.

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama, *Security of an identity-based cryptosystem and the related reductions*, in Nyberg [Nyb98], pp. 546–560.

[RK] Ronald L. Rivest and Burton Kaliski, *Encyclopedia of cryptography and security*, ch. RSA Problem, Kluwer, To appear. `http://theory.lcs.mit.edu/~rivest/RivestKaliski-RSAProblem.pdf`.