Breaking RSA May Be As Difficult As Factoring

Daniel R. L. Brown*

January 7, 2008

Abstract

If factoring is hard, this paper shows that 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.

Key Words: RSA, Factoring, Straight Line Programs.

1 Introduction

A long standing open question in cryptology is whether the RSA problem is as difficult as factoring. This paper provides a partial answer to this question: solving the RSA problem with a straight line program is almost as difficult as factoring, provided that the public exponent has a small factor.

A straight line program is an algorithm limited to a fixed sequence of addition, subtraction or multiplication steps. No branching or looping is allowed, so such a program computes a fixed integer polynomial function of its input. This paper shows that any efficient algorithm that takes an RSA modulus as input and outputs an efficient straight line program that solves the corresponding low exponent RSA problem can be used to factor the RSA modulus. Therefore if factoring is hard, then the RSA problem cannot be solved by a straight line program.

Note, however, that straight line programs also appear unable to solve certain problems that are known to be tractable, such as computing multiplicative inverses modulo an RSA number of unknown factorization. The difficulty of solving the RSA problem with algorithms that are not straight line programs is not addressed in this paper. Therefore, the existence of the efficient unlimited algorithms for solving the RSA problem, analogous to the Euclidean algorithm for finding inverses, has not been excluded.

Related Work: An RSA private exponent is known to reveal the factorization of the RSA modulus. This classical result about the difficulty of the RSA problem has been attributed in [8] to de Laurentis [4] and Miller [6], while in [5], it is attributed to [9]. The result in this paper extends the class of information that reveals the factorization. Let an RSA private exponent d correspond to the straight line program that takes input x and computes x^d. This straight line program solves the RSA problem. The extension here is that any other straight line program for solving the RSA problem also reveals the factorization. A secondary aspect of the extension is that, if a straight line program for x^d does not directly reveal d, the factorization is revealed nonetheless.

*Certicom Research

Rabin [7], in another classic result, showed that finding e^{th} roots where the RSA (Rabin) public exponent e has a very small factor, namely two, is equivalent to factoring. In some sense, this paper generalizes Rabin's result to larger factors of e, albeit adding the severe limitation to straight line programs. Table 1 illustrates the relationship between the results of this paper and the classical reductions [4, 6, 7, 9], showing its intermediacy between [4, 6, 9] and [7].

	Exponentiation	SLP	Any root-finding algorithm
Any e	[4, 6, 9]		
Small $e' \mid e$		[0]	
$2 \mid e$			[7]

Table 1: Relation of this reduction [0] to classical reductions between finding roots and factoring

The results of this paper combined with Rabin's result suggest an interesting pattern: the difficulty of finding e^{th} roots decreases as the smallest prime factor of e increases. Curiously, other results suggest just the opposite: that small e is less secure. Boneh and Venkatesan [2] show that solving the RSA problem may be easier than factoring, provided that the public exponent e is small or a product of small factors. They do so by proving that a straight line reduction cannot prove that solving the RSA problem is as difficult as factoring (unless factoring is easy). Their result, although in the opposite direction of the work here, did not contradict either the classical results [4, 6, 7, 9], and does not contradict the extension discussed here either, because the classical reduction and the extension do not fit exactly into the type of reduction considered in [2].

Also going towards the trend that smaller e is less secure is the long series work culminating in results such as Coppersmith's [3] that e^{th} roots can be found efficiently once partial information is known about the e^{th} root, provided that e is very small. These results apply to $e = 2$ as well. The fact that the partial information reveals full information about the root means that if it is hard to find the root, which is widely believed, then it is also hard to find merely some partial information about the root. For a general discussion of the paradoxical connection between attacks and security proofs, see §G. Shoup [10] has already argued, somewhat similarly, that $e = 3$ may offer more security than larger e in the context of OAEP. In the context of this paper, namely reductions between finding roots and factoring, we merely comment that this observation about the difficulty of finding even partial information about small degree roots ties in to the difficulty of factoring.

Organization of the Remainder of the Paper: Section 2 provides some definitions and lemmas for straight line programs and inverse pairs of polynomials. Section 3 gives reductions between factoring and solving the RSA problem with a straight line program when the public exponent has a small factor. Section 4 discusses how the implications of the paper are limited. Section 5 discusses why this paper does not contradict Boneh and Venkatesan's paper. Appendix A discusses a moderately complicated straight line program for computing cube roots. Appendix B discusses the difficulty of computing inverses using a straight line program. Appendix C discusses some generalizations of the RSA problem. Appendix D discusses applicability of this result to variants of the RSA problem such as the strong RSA problem. Appendix E discusses allowing division in the definition of straight line programs. Appendix F sketches how to extend the results of this paper to wider a class of algorithms that allow branching based on testing of equality.

Appendix G discusses a general form duality between positive and negative security results and its impact on this paper.

2 Straight Line Programs and Inverse Integer Polynomials

Straight line programs are a class of algorithms that do not branch, 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 $x_{-1} = 1$ and $x_0 = x$.*
2. *For $1 \leqslant k \leqslant L$, compute $x_k = x_{i_k} \circ_k x_{j_k}$.*
3. *Output $P(x) = x_L$.*

Let R be a ring. If $x \in R$, then $P(x) \in R$. An important ring for this paper is 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. The ring $\mathbb{Z}[X]$ of integer polynomials over the 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 natural 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 $r \in R$. Apply the straight line program P to the polynomial $X \in \mathbb{Z}[X]$, and let $\hat{P}(X) \in \mathbb{Z}[X]$ be the resulting output. The polynomial $\hat{P}(X)$ characterizes the action of P in any ring:

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

Proof. In the natural embedding of $\mathbb{Z}[X]$ into $R[X]$, we have $X \mapsto X$ and $\hat{P}(X) \mapsto \hat{P}(X)$. Therefore, in the ring $R[X]$, we have $P : X \mapsto \hat{P}(X)$. Now apply the natural homomorphism $R[X] \to R$, such that $X \mapsto r$, to get in the ring R that $P : r \mapsto \hat{P}(r)$. □

A straight line program P is essentially a particular algorithm to compute the polynomial $\hat{P}$ in any ring. The length of P is a simple measure of its efficiency, and an upper bound on the complexity of computing the polynomial $\hat{P}$. A secondary measure of efficiency, memory usage, will not be considered in this paper. Note that the degree of the polynomial $f(X)$ that P computes is at most 2^L, and similarly the largest coefficient is at most 2^L.

The main results of this paper use an observation about the actions of *inverse* pairs of integer polynomials in rings. If integer polynomial functions invert each other in finite ring R, then they invert each other in any image of R:

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 $s \in S$.*

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 integer 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 $p(q(s)) = s$.

For the sake of greater generality, one can also consider straight line programs that include division steps, not just addition, subtraction and multiplication steps. Such straight line programs have already been considered in [2], but are also reviewed briefly in §E of this paper for completeness. For even greater generality, one can also consider steps that branch based on whether two previous values are equal, which is consider in §F.

3 Factoring, the RSA Problem, and Straight Line Programs

When the RSA public exponent e is sufficiently small, one can use an efficient straight line program 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. More generally, it suffices for e to have a small factor.

3.1 Cube Roots: Public Exponent $e = 3$

A straight line program that finds cube roots modulo n can be used to construct another straight line program that finds a factor of n:

Theorem 3. *Let $f(X) \in \mathbb{Z}[X]$, let p and q be primes with $p \equiv q \equiv 2 \bmod 3$, 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 running over R of length $7L + K$, for some constant K, together a small amount of additional work.*

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 a 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 and $\mathbb{F}_{q^2}$ is represented as $\mathbb{F}_q[X]/\langle X^2 - u \rangle$. 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 $\bar{S}$ 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; this 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 at least μ^2, we have $f(s^3) = s$ and $f(\bar{s}^3) = \bar{s}$, since s and $\bar{s}$ are independent. 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 will now see that this event can happen with probability at most $\frac{1}{3}$. Note that $q^2 \equiv 1 \bmod 3$, 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 only one third of elements in T are perfect cubes, and each perfect cube has three cube roots. These cube roots are called conjugates. A set of such conjugates 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, the event $f(t^3) \neq t$ has probability at least $\frac{2}{3}$.

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 be implemented as a straight line program G running on the ring R. The resulting program has length at most $7L + K$, because multiplication in U can be implemented as seven ring operations in R, since $(a + bX)(c + dX) = (ac + bdu) + (cb + ad)X$. □

A small 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 compute $y = r^3 = 17 - 26X$, and submit y to the straight line program F, which gives $z = F(y) = y^7 = 9 + 17X$. Now $F(r^3) - r = z - r = 5 + 10X$. As predicted, $z - r$ is $0 + 0X \bmod 5$, and also 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 $n = 55$.

Given that the classical result [4, 6, 9] of a private exponent revealing the factorization, and that 7 is a private exponent for 3 modulo 55, one could have also used the classical results instead of Theorem 3. The example above does not illustrate the greater generality of Theorem 3. A class of polynomials outside the range of the classical result is those of the form X^d where $d = \frac{1}{3} \bmod m$ and m is some proper factor of $\operatorname{lcm}(p-1, q-1)$. These have success rate $\mu < 1$. Technically, such a d is not a private exponent, even though it can be used to find cube roots for a fraction of elements in $\mathbb{Z}/\langle pq \rangle$. We would expect, however, that the proofs in [4, 6, 9], or some minor extensions thereof, apply to such d. The polynomial $11X^3 + 45X^{17}$ will also compute cube roots modulo in $\mathbb{Z}/\langle 55 \rangle$,

being derived via the Chinese remainder theorem. This polynomial is not of the form X^d, but on the other hand the factorization of 55 is readily ascertained from its coefficients. Another class of polynomials can be derived from Cipolla's algorithm (see §A). While any given small example may obviously reveal the factorization by inspection, the power of Theorem 3 is that all examples will reveal the factorization.

The factor μ^2 in the success rate of factoring can be improved to μ, because the proof used in Theorem 3 is actually a simplification that does 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 $0 \neq z \in \mathbb{F}_{q^2}$. Let $c = f(r^3) - r$. Write $c = a + bX$, and let $\bar{c} = a - bX$. Then $\psi(\bar{c}) = (y, 0, \bar{z})$ for some $\bar{z} \in \mathbb{F}_{q^2}$. The norm of c is $c\bar{c} = a^2 - b^2u$, and

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

Therefore, $p \mid a^2 - b^2u$ and $q \nmid a^2 - b^2u$, 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 that takes as input an RSA modulus n of given size with public exponent of three and 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 $\frac{2}{3}\mu$. The cost of the factoring algorithm is roughly the cost of A plus seven times the cost of evaluating F.*

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

Provided that μ is not too small, that F is efficient, and that A is efficient, then one can factor efficiently. The success rate of the factoring 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.

The results above extend to straight line programs with division, although the efficiencies may change slightly due to the cost of implementing in division in the extension ring.

3.2 Higher Degree Roots: Public Exponent $e > 3$

The results for $e = 3$ generalize to higher public exponents. The following result requires e to be sufficiently small to make certain approximations in the proof, but this upper bound seems well above the threshold of values for which the result has cryptological significance.

Theorem 6. *Let $f(X) \in \mathbb{Z}[X]$, let $e > 3$ be an integer, let p and q be primes with $\gcd(e, (p-1)(q-1)) = 1$ and $p, q \gg e$, 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 $\frac{(e-1)(E-1)}{\phi(e)eE}\mu$, where E is the base of the natural logarithm, using a straight line program of length at most about*

$3\phi(e)^2L + K$ running over R, together with 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 that the first phase obtains this necessary condition on $g(X)$ for the second phase to succeed.

If $g(X)$ meets the condition, then 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 $g(X)$, and a generalization of the Chinese Remainder theorem, if $g(X)$ is square-free, the ring U has structure:

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

(If $g(X)$ is not square-free, then we can factor n by computing the discriminant of $g(X)$. So, if the following procedure fails, then we may attempt to do this.)

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 $F(r^e)$. Let s and t be the homomorphic projections of r in components S and T. Then $F(s^e) = s$ with probability at least μ, 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. In this event, $F(r^e) - r$ projects to 0 in S. Let $F(r^e) - r = z_0 + z_1X + \cdots + z_{\phi(e)-1}X^{\phi(e)-1} = z(X)$. 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 defined here as the determinant of the Sylvester matrix of the two polynomials. For polynomials defined over a field, the resultant is the product of all the differences between roots of the first and second polynomials, times the product of the leading coefficients each raised to the degree of the other polynomial. The resultant can be computed efficiently using a determinant or using an algorithm similar to the Euclidean algorithm. This takes approximately $O(\phi(e)^3)$, or $O(\phi(e)^2)$ respectively, $\mathbb{Z}/\langle n\rangle$ operations, including divisions. Henceforth, we absorb this as a relatively small cost, but note that for large e this cost may actually be significant compared to factoring n.

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:

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

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. Note that $g(X)$ is irreducible modulo q, so the only chance of having

common factors with $z(X)$ is if $g(X) \mid z(X)$, and thus F found a root in $\mathbb{F}_{q^{\phi(e)}}$. Therefore, with probability at least $\frac{e-1}{e}$, the gcd is p, which gives the desired factor of n.

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:

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

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 $\frac{1}{d}$. For large p, this approximation is very tight. The number of $g(X)$ with at least one root is:

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

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 $\frac{E-1}{E}$, where E is the base of the natural log (not to be confused the RSA public exponent), with a better approximation for larger d. A more accurate estimate for the probability, especially for smaller e is

$$\frac{1}{1!} - \frac{1}{2!} + \cdots \pm \frac{1}{\phi(e)!}, \tag{16}$$

which approaches $\frac{E-1}{E}$ quite quickly. Estimate (16) uses the approximation $\binom{p}{f} \approx \frac{p^f}{f!}$, which is only accurate if $p \gg f$. Once e gets large enough, other estimates may take over with the alternating sum in (15) being quite different from (16).

The straight line program F, as run over U, can be translated into a longer straight line program G running 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, 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 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 $p \equiv 3 \bmod 4$, 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.

Instead of the resultant in the proof, a greatest common divisor of polynomials could have been used. Modulo p, the polynomials $z(X)$ and $g(X)$ have a common root, namely s, so $(X - s) \mid \gcd(z(X), g(X))$, so the gcd has degree at least 1. Modulo q they do not have a root, so $\gcd(z(X), g(X))$ has degree 0. Modulo n, we should therefore have $\gcd(z(X), g(X))$ as a polynomial of degree at least 1, all of whose non-constant coefficients are zero modulo q. Therefore one of the non-zero non-constant coefficients c is such that $\gcd(c, n) = p$. The only problem with this approach

is defining the greatest common divisor over the ring $\mathbb{Z}[X]/\langle n\rangle$. The resultant has the advantage of being easily definable as the determinant of the Sylvester matrix, so it is not necessary to deal with a generalized definition of greatest common denominators in the proof.

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 these 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 may succeed in factoring more often (or quickly, in iterated form), except in the worst case.

The analogue of Theorem 5 about algorithms that take an RSA modulus and output a straight line program for finding roots is:

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 $\frac{(e-1)(E-1)}{\phi(e)eE}\mu$ 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. □

If A is as slow as factoring, then the straight line program F can be very efficient, such as exponentiating by the private exponent. The opposite extreme is with A very efficient, almost negligible compared to the cost of factoring, which entails a method to solve the RSA problem almost purely with a straight line program. We can consider how low the cost of F can be in this case. Essentially, the cost of solving the RSA problem almost purely with a straight line program is at least $\frac{(E-1)(e-1)}{3E\phi(e)^3}$ times the cost of factoring. This estimate uses Theorem 7 and incorporates a strategy of repeating F as often as necessary until the factorization is obtained.

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 purely with a straight line program is about 2^{30} operations in $\mathbb{Z}/\langle n\rangle$. This very loose estimate may be made more precise by more careful accounting in the proofs, (and perhaps it can be improved as well, with some optimization of the proof algorithms, such as Karatsuba).

The results above extend to straight line programs with division, although the efficiencies may change slightly due to the cost of implementing in division in the extension ring.

It must be emphasized that the actual difficulty of the RSA problem may be higher than the bounds proven here, or lower when not limited to straight line programs.

3.3 Security of the Hybrid Public Exponent $e = 3(2^{16} + 1)$

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 extend to when the public exponent is any multiple of stated public exponent.

The smaller the smallest factor of an RSA public exponent is, the tighter the bounds between the RSA problem and factoring given in the theorems above are. Furthermore, with a smallest factor of two, the classical reduction [7] between finding square roots and factoring can applied. This is very a tight reduction, and moreover is 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.

Because there are various security concerns about low public exponent RSA, such as the attacks of Coppersmith [3] (see also [1] for a survey of such attacks), and the theoretical work of Boneh and Venkatesan [2], it has been natural to doubt the general security of the low public exponent RSA problem, and especially the equivalence of its security to factoring. This paper may set aside some doubts, but only in a limited way because of the restriction to straight line programs. Therefore, it still remains prudent to use a moderately large public exponent, rather than, say, $e = 3$. By the same token, it may also be prudent to use a public exponent that is not product of small exponents. Otherwise, if the RSA problem is solvable for each of the small exponents, then it is solvable for their product. In this light, the commonly used prime public exponent $e = 2^{16} + 1$ enjoys some security properties: it resists the known attacks and yet is small enough to offer very competitive performance of public key operations. The exponent $e = 2^{16} + 1$, though, does not enjoy significantly the benefits of this paper, especially when compared to $e = 3$. The results of this paper are strongest when $e = 3$, or a multiple thereof.

Fortunately, there are public exponents that enjoy some of the benefits of both $e = 3$ and $e = 3(2^{16} + 1)$. Consider the exponent $e = 3(2^{16} + 1)$. Computing e^{th} roots is at least as difficult as computing cube roots, and thereby the results described in this paper provide some assurance, however limited it may be, of the hardness of the RSA problem for public exponent $e = 3(2^{16} + 1)$. 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. Also, the exponent $e = 3(2^{16} + 1)$, like $e = 2^{16} + 1$, seems to resist some of the attacks, such as Hastad's and Coppersmith's, against the public low exponent RSA scheme.

4 Limited Implications

The implications of the results in this paper have several limitations.

Small factors required in the public exponent. The results in this paper require the RSA public exponent to have a small factor. Otherwise, the extension degree gets quite large, and factoring with this result becomes much slower than solving the RSA problem. The results in this paper do not apply, for example, to large prime public exponents e, and thus do not provide any lower bounds on the RSA problem in these instances. Somewhat surprisingly, past work — such as the attacks of Coppersmith [3] and others (see [1] or [8] for a survey of other such attacks) and more theoretical results such as Boneh and Venkatesan's [2], — has generally shown security concerns with low public exponent RSA. The results of this paper in no way undo such past work. Despite the results here in favour of low exponent RSA, low exponent RSA should be avoided. At least, countermeasures to the known attacks are necessary, if low exponent RSA must be for some arcane reason.

Algorithms exist that solve SLP-hard problems. Inverses in $R = \mathbb{Z}/\langle n \rangle$ can be found efficiently using the Euclidean algorithm. The Euclidean algorithm neither requires the factorization of n nor is known to help significantly in factoring 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 factor n as long as 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 (see Appendix B) 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 looser reduction than between the RSA problem and factoring

Most functions $\mathbb{Z}/\langle n\rangle$ are not polynomials. Not all functions in an RSA ring $R = \mathbb{Z}/\langle n\rangle$ can be computed with a polynomial. All functions from a field to itself can be computed with a polynomial, but 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 $(pq)^{pq}$. The probability that a random function on R is a polynomial is thus approximately $n^{-\sqrt{n}}$. 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. Also the Jacobi symbol cannot be expressed as an integer polynomial, since it is not true that $x \equiv y \bmod p$ implies $\left(\frac{x}{n}\right) \equiv \left(\frac{y}{n}\right) \bmod p$. If it is the case that efficient non-polynomial functions can be used solve the RSA problem, then this paper's results would not apply to give a factoring algorithm.

Most polynomials are not SLP-efficient. 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 polynomial functions 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.

5 Why This Paper Does Not Contradict Boneh and Venkatesan's

The results of this paper, not to mention the classical results [4, 6, 7, 9], do not immediately contradict the results of Boneh and Venkatesan [2], despite being results in opposite directions. Neither this paper nor [2] claim to resolve the open question of whether the RSA problem is as difficult as factoring — both papers only provide evidence towards one possible answer — so there is no contradiction between the opposite sounding claims, at least without inspecting the details. Nevertheless, even though each piece of evidence is inconclusive in its own right, one naturally wonders how such conflicting pieces of evidence could co-exist, so a few words of explanation are worthwhile to explain the lack of contradiction.

Recall that Boneh and Venkatesan show that any factoring algorithm that is a straight line program that also uses an oracle for solving the RSA problem can be made into another factoring algorithm that is a straight line program that does not use an oracle for solving the RSA problem. In other words, if the initial factoring algorithm is a *straight line reduction*, then factoring is easy. Our reductions appear not to be straight line reductions, as defined in [2], for the reasons given below, and therefore we have no technical contradictions between the oppositely directed results.

The straight line reductions defined in [2] are very powerful in that they do not look inside the RSA problem solving oracle. Any proof about such powerful reductions does not apply to reductions that violate this condition, such as ours. In other words, results such as [2] about

straight line reductions, or more generally reductions with oracle-only access to the RSA problem solving algorithm, are weak in the sense that they are limited to a very special kind of reduction.

Normally, in direct reductions, having oracle-only access is the strongest possible condition. In *metareductions*, reductions about reductions, however, oracle-only access becomes a weaker condition on the results. At first, this appears counterintuitive, but once one gets use to the idea of metareductions such as [2], it should become clearer. Since our reductions are direct reductions, not metareductions, the fact that we use more than oracle-only access means that our results are weaker than the strongest possible. Oracle-only access strengthens direct reductions but weakens metareductions. Therefore both the result of this paper and [2] are weaker than they could theoretically be.

Perhaps a bottom line metaphor would be the reductions given a lower negative bound and [2] gives a positive upper bound, which are non-contradictory bounds on the opposite side of quantity. Neither result completely settles the dilemma of whether the quantity is positive or negative, that is, whether the RSA problem is as difficult as factoring.

6 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 efficient algorithm can output a straight line program that solves the RSA problem efficiently, provided the public exponent has a small enough factor. The reduction is loose for the common public exponent $e = 2^{16} + 1$, but is quite tight for public exponents divisible by three. It must be emphasized that this work in no way rules out algorithms that solve the RSA problem other than by a straight line program.

Acknowledgments

Steven Galbraith pointed out a major mistake in a previous paper of the author. The author's efforts to correct this mistake ultimately led to this paper. Alfred Menezes provided extensive comments on the presentation of this paper. Adrian Antipa, Rob Lambert, Scott Vanstone, Rene Struik, and John Goyo also provided comments. Andy Rupp provided several comments.[1]

References

[1] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society*, 46(2):203–213, 1999. `http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html`.

[2] D. Boneh and R. Venkatesan. Breaking RSA may be easier than factoring. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, number 1403 in LNCS, pages 59–71. IACR, Springer, May 1998. `http://crypto.stanford.edu/~dabo/abstracts/no_rsa_red.html`.

[1] One of which led to the correction: the condition that $g(X)$ should be square-free, which can be used to factor if it fails.

[3] D. Coppersmith. Finding a small root of a univariate modular equation. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, number 1070 in LNCS, pages 155–165. IACR, Springer, May 1996.

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

[5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[6] G. L. Miller. Riemann's hypothesis and test for primality. *Journal of Computer and Systems Science*, 13(3):300–317, 1976.

[7] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. LCS/TR 212, MIT, 1979.

[8] R. L. Rivest and B. Kaliski. *Encyclopedia of Cryptography and Security*, chapter RSA Problem. Kluwer, 2002. To appear. `http://theory.lcs.mit.edu/~rivest/RivestKaliski-RSAProblem.pdf`.

[9] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[10] V. Shoup. OAEP reconsidered. ePrint, IACR, 2000. `http://eprint.iacr.org/2000/060`.

A Cipolla's Algorithm for Cube Roots over Composite Rings

Choose two large primes $p < q$. Suppose that:

$$\gcd(p^2 - 1, q^2 - 1) \mid q - p \tag{17}$$

Let D be such that:

$$\begin{aligned} D &\equiv p + 1 \bmod p^2 - 1 \\ D &\equiv q + 1 \bmod q^2 - 1 \end{aligned} \tag{18}$$

The Chinese remainder theorem, together with (17), ensures that D exists. If $p \equiv q \equiv 2 \bmod 3$, then $D \equiv 0 \bmod 3$. Let $d = D/3$. Let $c = \mathrm{lcm}(\frac{p^2-1}{3}, \frac{q^2-1}{3})$. Let $n = pq$ and let $t \in \mathbb{Z}/\langle n \rangle$. Define the ring:

$$R_t = \mathbb{Z}[X, Y]/\langle n, Y^2 - tY + X \rangle \tag{19}$$

Every element in this ring can be represented in the form $a(X) + b(X)Y$, where $a(X), b(X) \in \mathbb{Z}[X]/\langle n \rangle$, by repeatedly substituting $Y^2 = tY - X$ until no higher powers of Y remain. Use this observation to define a polynomial $f(X)$ with the property:

$$(r + sY)^c Y^d = f(X) + g(X)Y, \tag{20}$$

where r, s are some random elements of $\mathbb{Z}/\langle n \rangle$. For a fixed t and random $x \in \mathbb{Z}/\langle n \rangle$, then $f(x)^3 = x$ with probability about $\frac{1}{36}$. To see this, we work in the ring $R_{t,x} = R_t/\langle X - x \rangle = \mathbb{Z}[Y]/\langle n, Y^2 - tY +$

$x\rangle$. Suppose that $Y^2 - tY + x$ is irreducible in $\mathbb{F}_p[Y]$ and $\mathbb{F}_q[Y]$, which happens with probability about $\frac{1}{4}$. In this case:

$$R_{t,x} \cong \mathbb{F}_{p^2} \times \mathbb{F}_{q^2}. \tag{21}$$

Consider the image of Y^D in $\mathbb{F}_{p^2}$:

$$Y^D = Y^{p+1} = YY^p. \tag{22}$$

The roots of the polynomial $Y^2 - tY + x$ in $\mathbb{F}_{p^2}$ are Y and Y^p (with a slight abuse of notation). The product of the roots is the constant coefficient, so $Y^D = x$. The same holds in $\mathbb{F}_{q^2}$ and thus in $R_{t,x}$. Therefore Y^d is a cube root of x, which is what we seek, but is not yet in the correct form.

Again, working in $\mathbb{F}_{p^2}$, note that Y^d is a cube root of $x \in \mathbb{F}_p$, but it may not be the case that $Y^d \in \mathbb{F}_p$. But since, $p \equiv 2 \bmod 3$, there exists exactly one cube of x in $\mathbb{F}_p$, say y. Therefore, Y^d is y or a conjugate of y. Let u be a primitive cube root of unity in $\mathbb{F}_{p^2}$, which exists since $3 \mid p^2 - 1$. Then we have $y \in \{Y^d, uY^d, u^2Y^d\}$. One approach is to compute u, then try each conjugate of Y^d in order to find y. A slightly different approach is to take random $v = r + sY \in \mathbb{F}_{p^2}$ and compute $v^{(p^2-1)/3}Y^d$, because $v^{(p^2-1)/3} \in \{1, u, u^2\}$. The latter approach gives a $\frac{1}{3}$ probability of obtaining y. This holds in $\mathbb{F}_{q^2}$ and thus in $R_{t,x}$. The left hand side of (20) equals y, the unique cube root of x in the ring $\mathbb{Z}/\langle n\rangle$, with probability $\frac{1}{4} \times \frac{1}{3} \times \frac{1}{3}$. When this happens, the right hand side is in $\mathbb{Z}/\langle n\rangle$, because y is, therefore the second coefficient $g(x) = 0$. It thus suffices to consider only the polynomial $f(X)$.

To be a relevant example for this paper, the polynomial $f(X)$ should be efficiently computable via a straight line program. This can be done using a square and multiply algorithm to compute the powers in (20), with reduction modulo $Y^2 - tY + X$ done at every step. Each intermediate value in the straight line program for $f(X)$, will be either Y^0 or the Y^1 coefficient of some intermediate value of the square-and-multiply algorithm for $(r + sY)^c Y^d$.

This example used a quadratic extension to find cube roots. Other degree extensions can be used to find other degree roots. A large variety of straight line programs exist that solve the RSA problem. This paper shows that finding any of these programs without knowing the factorization is almost as difficult as factoring.

B An Easy but SLP-Hard Problem: Finding Inverses

Finding inverses in $\mathbb{Z}/\langle n\rangle$ is easy with the Euclidean algorithm. Straight line programs can also find inverses, just as they can be used to find cube roots. Typical straight line programs for finding inverses reveal the factorization of n. In this section, we investigate whether any such straight line program reveals the factorization. If so, then finding inverses is an example of a problem that is (a) similar to the RSA problem, in that it can be solved with a straight line program but only if the factorization is known, but (b) dissimilar to the RSA problem in that we know how to solve it easily with another kind of algorithm. If finding inverses is an easy but SLP-hard problem, then the RSA problem might be too.

Suppose $f(X) \in \mathbb{Z}[X]$ is such that $xf(x) \equiv 1 \bmod n$ for any $x \in \mathbb{Z}/\langle n\rangle$, and that $f(X)$ is efficiently computable as a straight line program (SLP) of length L. Note that the degree of $f(X)$ is at most 2^L, and that $Xf(X) - 1$ has at most $2^L + 1$ roots in any field.

Let $g(X) \in \mathbb{Z}[X]$ be a polynomial that is irreducible over $\mathbb{F}_q$ with degree d such that $q^d \gg 2^L + 1$. Suppose that $g(X)$ has a root in $\mathbb{F}_p$. The ring $R = \mathbb{Z}/\langle n, g(X)\rangle$ has subrings isomorphic to the fields $\mathbb{F}_p$ and $\mathbb{F}_{q^d}$. In the field $\mathbb{F}_{q^d}$, the polynomial $Xf(X) - 1$, has a negligible proportion of zeros. For random $r \in R$, the probability that $rf(r) = 1$ in R is thus negligible. However, the image of

$rf(r)$ in the subring isomorphic to $\mathbb{F}_p$ will be 1. Write $rf(r)-1$ in R as $z(X)$ for some $z(X) \in \mathbb{Z}[X]$. Then, as before, $\gcd(n, \mathrm{Res}(z(X), g(X))) = p$, gives the desired factorization.

The efficiency and success rate of the procedure above depends on the degree d of $g(X)$. Larger d reduces the efficiency, and larger L increases d. We can bound d above by 2^L. The highest possible degree of $f(X)$ is 2^L, which is attained only by $f(X) = X^{2^L}$. For this $f(X)$ of this maximum degree, there is only one root. But the polynomial $X^{2^L} - 1$ has length $L+1$ and 2^L distinct roots of over the algebraic closure $\overline{\mathbb{F}}_q$.

Therefore, finding inverses modulo an RSA modulus of unknown factorization using a straight line program (without division) may be somewhat difficult, at least to do with a very short straight line program.

C Generalized RSA

For some rings $\mathbb{Z}/\langle n \rangle$, the function $E(x) = x^e$ is a bijection. The bijectivity of this function is the basis for the RSA public key cryptosystem. One may generalize this by taking $E(x)$ to be some other rational function, rather than a monomial. (More generally, one need not confine oneself to rings $\mathbb{Z}/\langle n \rangle$.) This leads to a couple questions:

1. When is a rational function $E(x)$ is a bijection over $\mathbb{Z}/\langle n \rangle$, and how does one compute its inverse?

2. When can such a bijective rational function $E(x)$ over $\mathbb{Z}/\langle n \rangle$ serve securely as a public key operation, thus generalizing the RSA public key cryptosystem? In particular, when do the results of this paper generalize to such an $E(x)$?

To address these questions, write $E(x) = \frac{e_1(x)}{e_0(x)}$ where e_0 and e_1 are polynomials. For E to be a bijection the equation $E(x) = E(y)$ must imply $x = y$ for $x, y \in R$. Therefore consider the expression $E(x) - E(y) = 0$. Multiply this by $e_0(x)e_0(y)$ to a get a polynomial in x and y. This polynomial is zero whenever $x = y$, so we may divide out by the factor $x - y$, to get another polynomial $e_2(x, y)$ in two variables, that is

$$e_2(x, y) = \frac{(E(x) - E(y))(e_0(x)e_0(y))}{x - y} = \frac{e_1(x)e_0(y) - e_0(x)e_1(y)}{x - y} \tag{23}$$

In fact, the factor $x - y$ may actually divide $e_2(x, y)$ as defined above, once again. So instead, in that case, we define $e_2(x, y)$ by dividing the highest power of $(x - y)$ possible. Bijectivity of $E(x)$ over the ring R, is now essentially characterized by the condition that the curve $e_2(x, y) = 0$ has no points in $R \times R$.

The results of this paper may apply if we can find extension rings S of ring R, such that we can ensure that with a sufficiently larger probability for a given x there exists a y with $e_2(x, y) = 0$ in one field component of the ring S, but not for the other field component. The problem is then to start with fields in which the curve defined by e_2 has no points, and extending the field until the curve has some points.

Some examples may illustrate the general applicability of the framework above.

- Let $E(x) = x^e$. Then $e_2(x, y) = \prod_{j=1}^{e-1}(u^j - y)$, where u is a primitive e^{th} root of unity. The curve $e_2 = 0$ has an R-rational point if and only if $u^j \in R$ for some $1 \leqslant j \leqslant e - 1$. This example is just the classic RSA case. For $R = \mathbb{Z}ideal n$, there are no such u^j, but in the reductions of this paper, we found extensions for which such u^j existed.

- Let $E(x) = 1/x$. Then $e_2(x, y) = -1$. The curve $e_2 = 0$ has no R-rational points for any ring R. The inverse function is bijective, but since the curve $e_2 = 0$ never has any points for any ring, it is much harder to apply the results of this paper (but see §B). Much more importantly, the function $E(x)$ is completely insecure as a public key operation, since it is its own inverse.

- Let
$$E(x) = \frac{(3x^2 + a)^2}{4(x^3 + ax + b)} - 2x, \tag{24}$$
which is the formula for computing the x-coordinate of the double of point (x, y) on an elliptic curve defined by $y^2 = x^3 + ax + b$. If p is an odd prime, and this curve has an odd number of points modulo p, then $E(x)$ is essentially invertible in $\mathbb{F}_p$. In this case, $e_2(x, y)$ has a rather complicated expression:
$$\begin{aligned} e_2(x, y) = x^3y^3 + axy(x^2 + xy + y^2) + 2ax^2y^2 + b(x^3 + x^2y + xy^2 + y^3) \\ + 8bxy(x + y) - 2a^2xy - a^2(x^2 + xy + y^2) - 2ab(x + y) - 8b - a^3, \end{aligned} \tag{25}$$
from which not a lot is immediately obvious. However, knowing that $E(x)$ represents a point doubling formula, we realize that if we find an extension of $\mathbb{F}_p$ over which $y^2 = x^3 + ax + b$ has an even number of points, we can expect $E(x)$ to be at least two-to-one over this extension. In other words, we need a point of order two, which must have form $(x, 0)$, so that $x^3 + ax + b = 0$. Note that because the curve order is odd over $\mathbb{F}_p$, the polynomial $x^3 + ax + b$ has no roots in $\mathbb{F}_p$, and thus is irreducible. In a third degree extension of $\mathbb{F}_p$, the polynomial $x^3 + ax + b$ always has at least one root, so the curve order over $\mathbb{F}_{p^3}$ is always even. If the curve order is even, then ensure $E(x) = x'$ has at least two solutions x for every $x' \in \mathbb{F}_{p^3}$. Suppose algorithm A takes an RSA modulus n for which $E(x)$ is invertible over $\mathbb{Z}/\langle n\rangle$ and outputs a straight line program F that inverts $E(x)$. Take a random third degree polynomial $g(X) \in \mathbb{Z}[X]$ and form the ring $R = \mathbb{Z}[X]/\langle n, g(X)\rangle$. As usual, with reasonable probability we will have that $g(X)$ is irreducible modulo q, but has a root modulo p. As in the reductions between root-finding algorithms and factoring, compute $F(E(r))$ for random $r \in R$. Note that $F(E(r)) = r$ modulo p, or more precisely that this holds in a projection to a subring isomorphic to $\mathbb{F}_p$. Meanwhile, modulo q, we will have $F(E(r)) \neq r$, with probability at least $\frac{1}{2}$. As before, in this event a resultant and a gcd can be used to find p. This gives some evidence that $E(x)$ could be used securely as a public key encryption function.

D Variant RSA Problems

It is not uncommon in cryptology to consider easier variants of the RSA problem, because the security of certain RSA-based cryptographic schemes can be proven more easily and more tightly related to the easier variant RSA problems than to the classic RSA problem. For example:

- In the *strong RSA problem*, the exponent is part of the solution. The input is (n, y), where n is the RSA modulus, and the output is (e, x) such that $x^e \equiv y \bmod n$ for some $e > 1$. The strong RSA problem is easier than the classic RSA problem.

- The *oracle RSA problem* is $m+1$ copies of the classic RSA problem except that the solver gets m accesses to an oracle for solving the classic RSA problem. The input is $(n, e, y_1, \ldots, y_{m+1})$,

and the output is $(x_1, \ldots, x_{m+1})$, such that $x_i^e \equiv y_i \bmod n$. Before generating its output, the solver may select any $(w_1, \ldots, w_m)$ and receive $(z_1, \ldots, z_m)$ such that $z_i^e \equiv w_i \bmod n$.

It is natural to ask whether the results of this paper say anything about the difficulty of such variants of the RSA problem. For the strong RSA problem, it appears that nothing can be said because the results in this paper say nothing for large public exponent e.

For the oracle RSA problem, the public exponent would have to be small for our results to apply, but a complication arises from answering the oracle queries. It appears to be possible to simulate correct oracle responses by using extension rings, as follows. Apply the reduction in this paper until the problem solver makes its first oracle query. Note what this element is, and then start over with a larger extension ring in which the oracle input has a root. This process must be repeated m times, with the extension ring expanding m times. The field component of the final extension ring look like $\mathbb{F}_{p^{e^m}}$. In order for the ring operations to be efficient, m has to be quite small.

E Straight Line Programs with Division

In the rings of interest in this paper, $\mathbb{Z}/\langle n \rangle$, division is almost always defined, and furthermore can be computed via an efficient algorithm, namely the Euclidean algorithm for inversion. More precisely, failure of division of two random elements occurs with negligible probability in $\mathbb{Z}/\langle n \rangle$ if n is an RSA modulus. More importantly, if a division does fail, then the factorization of n will generally be revealed, because the denominator will have a nontrivial gcd with n.

Rings like $\mathbb{Z}/\langle n \rangle$ with the property that division is almost always defined (and can be computed effectively) will be called *near-fields*. Straight line programs with division allowed make sense for near-fields. To extend the main results of this paper to straight line programs with division, the following helps:

Lemma 8. *Let R be a near-field and let $g(X) \in \mathbb{Z}[X]$. Then $S = R[X]/\langle g(X) \rangle$ is a near-field. Any straight line program (with division) over S can implemented as a (multi-input) straight line program (with division) over R.*

Proof. By definition, S is a ring, so it suffices to define inverses on S for almost all elements of S. Let d be the degree of $g(X)$. Elements of S may be represented as polynomials in $R[X]$ of degree at most $d-1$. For almost any element $s(X) \in S$ with this representation, we can compute $s(X)^{-1}$ using the extended Euclidean algorithm applied to $g(X)$ and $s(X)$. The extended Euclidean algorithm will generally involve d applications of the polynomial division algorithm. Each polynomial division will require a certain number of divisions in the near-field R. The total number of near-field division in R is generally about $\binom{d}{2}$, when computed as above. However, upon simplification all these divisions in R can be consolidated into a single division, if desired.

Addition, subtraction, multiplication and division in S can each be implemented as multi-input straight line programs acting on the coefficients of elements of S when represented as polynomials in $R[X]$. □

To illustrate, suppose that $g(X) = X^3 + aX^2 + bX + c$, and that we want to compute the inverse of $f(X) = rX^2 + sX + t$. We will apply the polynomial division algorithm twice to get:

$$g(X) = q(X)f(X) + h(X) \tag{26}$$

$$f(X) = u(X)h(X) + k(X) \tag{27}$$

where $q(X)$, $h(X)$ and $u(X)$ have degree one, while $k(X) = k$ has degree zero, so is a constant scalar. Combining these equations, we get $k = f - uh = f - u(g - qf) = (qu+1)f - ug$. Therefore $f^{\frac{(qu+1)}{k}} \equiv 1 \bmod g$. The results of the polynomial divisions give:

$$q(X) = \frac{1}{r}X + \frac{a}{r} - \frac{s}{r^2}, \tag{28}$$

$$h(X) = \left(b - \frac{t}{r} - \frac{as}{r} + \frac{s^2}{r^2}\right)X + c - \frac{at}{r} + \frac{st}{r^2}, \tag{29}$$

$$u(X) = \frac{r^3}{br^2 - rt - rsa + s^2}X + \frac{r^3(s(br^2 - rt - rsa + s^2) - (cr^2 - art + st))}{(br^2 - rt - rsa + s^2)^2}, \tag{30}$$

$$k(X) = t - \frac{r(s(br^2 - rt - rsa + s^2) - (cr^2 - art + st))(cr^2 - art + st)}{(br^2 - rt - rsa + s^2)^2}. \tag{31}$$

Upon simplification to a single division we get:

$$\frac{1}{rX^2 + sX + t} = \frac{\begin{array}{c}\big((br^2 - rt - rsa + s^2)X^2 \\ + (s^2a - rsa^2 - r^2c - st + r^2bc)X \\ + t^2 + r^2b^2 - ast - acr^2 - 2brt - rsab + rsc + a^2rt + bs^2\big)\end{array}}{r^3c^2 - bcr^2s - 2acr^2t + b^2r^2t - abrst + acrs^2 + 3crst + a^2rt^2 - 2brt^2 + t^3 - ast^2 + bs^2t - cs^3}. \tag{32}$$

A straight line program with division computes a rational function $\mathbb{Q}[X]$. Lemmas 1 and 2 can be extended accordingly, with polynomials in $\mathbb{Z}[X]$ replaced by polynomials in $\mathbb{Q}[X]$, rings replaced with near-fields, straight line programs without division replaced by those allowing division. Unless specifically stated otherwise, however, straight line programs in this paper will not include division. Most of the results in this paper extend to straight line programs with division. The proofs of these extensions and the impact on tightness of the reductions depend on the lemma above, and are not discussed in detail.

F Straight-Line Equality-Excepted Programs

Straight line programs are called so partly because they do not involve branching steps, that is, conditional statements. As such, they represent quite a narrow class of algorithms. The results of this paper would be strengthened if the affected class of algorithms were broadened. In this section, we consider a limited form of branching where equality-testing is allowed, which we call a straight-line, equality-branching-excepted program (SLEEP). The significance of this extension will remain debatable, however, until an convincing example is provided that a SLEEP can do more powerful things than an SLP. To formally model a SLEEP, we allow another kind of step in the form (i_k, j_k, l_k, m_k), where $i_j, j_k, l_k, m_k < k$, which is taken to mean that $x_k = x_{i_k}$ if $x_{l_k} = x_{m_k}$ and $x_k = x_{j_k}$ otherwise.

Neither Lemma 1 nor Lemma 2 apply when an SLP or integer polynomial is replaced by a SLEEP. Indeed, a SLEEP is capable of computing non-polynomial functions, unlike an SLP. We therefore consider some modified lemmas, and argue that these lemmas can be used in to make the

proofs of the theorems apply to a SLEEP. The first lemma corresponds to something that was used as an implicit consequence of Lemma 1: that the action of a program on the product ring was the product of the actions on each ring.

Lemma 9. *Let R and S be rings. Let F be a SLEEP. Let $(r, s) \in R \times S$. Then $F(r, s) = (F(r), F(s))$, or in the course of running F on (r, s), one can find $(u, v) \in R \times S$ with $u = 0$ or $v = 0$.*

Proof. Run F on (r, s) and r and s. Let F_k indicate the SLEEP up to and including the k^{th} step in the SLEEP. Compare $F_k(r, s)$ and $(F_k(r), F_k(s))$. At the first k where these two values diverge, the divergence must be due to an equality testing step (i_k, j_k, l_k, m_k), because arithmetic steps will not cause divergence. Letting r_k and s_k indicating $F_k(r)$ and $F_k(s)$, one can see this divergence arises if and only if $r_{l_k} = r_{m_k}$ and $s_{l_k} \neq s_{m_k}$, or vice versa. Let $(u, v) = (r_{l_k} - r_{m_k}, s_{l_k} - s_{m_k})$. □

Similarly, we have a modified version of Lemma 2.

Lemma 10. *Let R and S be rings. Let $\sigma : R \to S$ be a surjective homomorphism. Let P and Q be SLEEPs. Let $r \in R$ and $s \in S$ be selected at uniformly random. If, $P(Q(r)) = r$ with probability at least π, then, with probability at least π, $P(Q(s)) = s$ or during the course of computing $P(Q(s))$ one can find $u \in R$ such that $u \neq 0$ and $\sigma(u) = 0$.*

Proof. For the given s, we may select r as a random preimage of s under σ. This r is uniformly randomly distributed in R, so therefore $r = P(Q(r))$ with probability at least π. Apply σ to both sides to get $s = \sigma(r) = \sigma(P(Q(r)))$. Unlike in Lemma 2, homomorphism σ may not commute with P and Q, because they are SLEEPs, not integer polynomials. However, it is true that $\sigma P^\sigma = P\sigma$, where P^σ is a modified SLEEP in which equality testing is done modulo the kernel of σ. Therefore, $P(Q(s)) = \sigma(P^\sigma(Q^\sigma(r)))$. If $P(Q(r)) = P^\sigma(Q^\sigma(r))$, then we have established that $s = P(Q(s))$.

Otherwise $P(Q(r)) \neq P^\sigma(Q^\sigma(r))$ which can only happen if the divergence is due to the difference in equality testing. In the first step where divergences happens we will be able to find nonzero u in the kernel of σ, by subtracting the two quantities being compared for equality, which is similar in principal to what was done in the proof of Lemma 9. □

When applying these modified lemmas in the proofs of the theorems, if they fail to work just as the original lemmas, then they reveal a factor of $n = pq$.

G Duality Between Positive and Negative Security Results

A frequent phenomenon in cryptology is that what in some contexts is negative security result, namely an attack, in other contexts can be interpreted as a positive security result. Although this may sound counterintuitive, it derives from the logically sound contrapositive. The archetypal example is the Rabin public-key cryptosystem, which was both (a) proven to be secure as factoring and (b) broken by a chosen-ciphertext attack. The underlying basis for these two paradoxically contradictory results is the contrapositive. Result (a) flows from the Rabin problem (RSA problem with even e) being hard as factoring, and result (b) flows from the factoring being as easy as the Rabin problem. The fully dramatic and practical form of the contradiction then applies when different contexts to this simple contrapositive. In result (a), the context is that factoring is assumed hard and that security is defined is as hardness of the Rabin problem, while in result (b), the context is that of chosen ciphertext attacks exploiting a victim as an oracle to solve the Rabin problem. Of course, not every negative or positive security result has such a dual, although

it may be worthwhile examining in each case the possibility and significance of an opposite dual result. In practice, the paradoxical situation of such dual results can often be resolved by carefully mitigating the negative side by adjusting the context. For the Rabin cryptosystems, measures can be taken whereby the victim, the private key holder, does not provide an adversary oracle access to the Rabin problem.

In this paper, we have presented a positive security result, namely that the RSA problem seems hard because the factoring seems hard. The immediate contrapositive is that factoring is easy if the RSA problem is easy, specifically easy to enough to solve with an SLP. Truly, one could valiantly attempt to solve the RSA problem with an SLP in an effort to factor, but hardly anybody expects that the RSA problem could be that easy. Next, one may more subtly consider a context wherein the RSA problem becomes easy, say via the victim acting as an oracle to solve it, as was done in the chosen ciphertext on the Rabin cryptosystem. In this setting, however, we would need the victim to be an SLP whose description is available to the adversary, because the factoring algorithm looks inside the SLP in order to run it on extension rings. To make this a realistic attack, the adversary would essentially have to look inside the victim's implementation. In practice, of course, the victim's implementation would already contain the private key, so it would not be necessary to use the full power of the theorems in this paper. Thus there does not seem any significance negative security results deriving from the dual of the positive security results. A possible explanation of the lesser signficance of the dual of the results in this paper is that the positive security result in this paper is weaker (with respect to the type of root-finding algorithm allowed) than the positive security result for the Rabin problem, so therefore the dual attack is correspondingly weaker (with respect to the type of adversary allowed).