MULTI-POWER POST-QUANTUM RSA

JOHN M. SCHANCK

Department of Combinatorics and Optimization
Institute for Quantum Computing
University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

Abstract. Special purpose factoring algorithms have discouraged the adoption of multi-power RSA, even in a post-quantum setting. We revisit the known attacks and find that a general recommendation against repeated factors is unwarranted. We find that one-terabyte RSA keys of the form $n = p_1^2 p_2^3 p_3^5 p_4^7 \cdots p_i^{\pi_i} \cdots p_{20044}^{225287}$ are competitive with one-terabyte RSA keys of the form $n = p_1 p_2 p_3 p_4 \cdots p_i \cdots p_{2^{31}}$. Prime generation can be made to be a factor of 100 000 times faster at a loss of at least 1 but not more than 17 bits of security against known attacks. The range depends on the relative cost of bit and qubit operations under the assumption that qubit operations cost 2^c bit operations for some constant c.

1. Introduction

An RSA modulus is a "publicly specified product, n, of two large secret prime numbers p and q" [17]. This – a direct quote from the paper that introduced RSA – is an uncontroversial and historically accurate definition. But, while $n = pq$ is the most common form for RSA moduli in deployment, it is not clear that things *should* be this way. A lot has changed since [17] was written. The bit-length of n that provide "adequate security" has increased decade by decade. Why hasn't the number of factors? Or their multiplicity? Assuming modest, or even dramatic, reductions in the cost of factoring, what shape should RSA moduli take in the long term?

If you adhere to some dictum like *the right form is that which maximizes security as a function of the bit length of* n, then $n = pq$ is quite plausibly the right form. However, security is not the only measure of a cryptosystem. Users consider computational efficiency, compactness, resistance to side-channel attacks, and intellectual property claims, among other attributes. So there may be room for alternative key forms.

Indeed, a number of alternatives to $n = pq$ have been proposed. The fact that it is possible to use $n = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$ was mentioned in the RSA patent [18]. Other proposals in the literature tend to focus on improving efficiency.

- Takagi [22] has suggested $n = p^r q$. This form admits a fast decryption algorithm based on Takagi's earlier work [21] on "multi-block" RSA with modulus $n^k = (pq)^k$.

E-mail address: jschanck@uwaterloo.ca.
Date: 2018-04-06.

- Lim, Kim, Yie, and Lee [11] have suggested $n = p^r q^s$. They claim that $n = p^2 q^3$ is 15 times faster than $n = pq$ for 8192-bit n. They consider, but dismiss, the use of more than two primes.
- Shamir [19] has suggested unbalanced moduli, $n = pq$, with q much larger than p. He has also proposed efficiency enhancements so that large n can be used – large enough that the system is "likely to provide long term security even to professional paranoids."
- Collins, Hopkins, Langford, and Sabin [4] have patented efficient methods for using moduli of the form $n = p_1 p_2 \cdots p_\ell$ where each prime is $\approx (\lg n)/\ell$ bits and $\ell > 2$ is a constant.
- Bernstein, Heninger, Lou, and Valenta [1] have suggested $n = p_1 p_2 \cdots p_\ell$ where each prime is of $(\lg \lg n)^{2+o(1)}$ bits and ℓ grows proportionally with n. They propose key generation, encryption, and decryption routines that each cost $(\lg n)(\lg \lg n)^{O(1)}$ bit operations.

All of these proposals sacrifice the security of the standard RSA key form for greater efficiency – even Shamir's "paranoids" appear to have limited patience for slow cryptography. The efficiency gains are largely in decryption and are due to the decryption algorithms of Quisquater and Couvreur [16] and Takagi [21]. The cost of these decryption algorithms, at least for moduli with a constant number of prime factors, depends on the size of the private primes rather than the size of n.

Special purpose factoring algorithms limit the extent to which one can use small primes in RSA. Lenstra's elliptic curve method (ECM) heuristically finds a prime p that divides n in $\exp\left((\log p)^{1/2+o(1)}\right)$ bit operations. Hence, a system designer who tries to maximize performance for a given bit length must weigh the cost of ECM against the cost of the best general purpose factoring algorithm in his attack model.

The number field sieve, a pre-quantum general purpose factoring algorithm, factors n in $\exp\left((\log n)^{1/3+o(1)}\right)$ bit operations. As such, small private primes weaken security when $\min_{p|n} \log p < (\log n)^{2/3}$, at least asymptotically. The non-asymptotic analysis is somewhat more delicate: for $\lg n \approx 2048$ no more than 3 equally sized private primes should be used, and for $\lg n \approx 4096$ no more than 4 equally sized private primes should be used [8].

Shor's quantum factoring algorithm dramatically reconfigures the relationship between special and general purpose factoring algorithms. Shor's algorithm factors arbitrary n with high probability using only $(\lg n)^{2+o(1)}$ qubit operations. For some n the cost can even be reduced to $(\log n)^{1+o(1)}$ qubit operations. Faced with Shor's algorithm, one has to consider whether honest parties are left with any advantage over attackers. This advantage can be formalized as a "cost/performance ratio" – the cost of the best attack on the system divided by the cost of using the system.

The proposal of Bernstein, Heninger, Lou, and Valenta [1], which we will call "multi-prime post-quantum RSA" or "multi-prime pqRSA", is an attempt to maximize the cost/performance ratio of RSA against Shor's algorithm. The use of small primes is still a liability in a post-quantum setting, but their use is also a necessity. The cost/performance ratio of RSA with $n = pq$ is constant (assuming qubit operations and bit operations have equal cost), while the cost/performance ratio of multi-prime pqRSA is $(\lg n)^{1+o(1)}$.

Could a different key form provide an even larger post-quantum cost/performance ratio? There is some evidence that other forms are more efficient in practice. Boneh

and Shacham [3] have compared multi-prime RSA, $n = p_1 p_2 p_3$, with Takagi's variant, $n = p_1^2 p_2$, and have found that decryption in Takagi's system is faster. This suggests that a multi-power post-quantum RSA, with $n = p_1^{e_1} p_2^{e_2} \dots p_\ell^{e_\ell}$, might have better performance than multi-prime post-quantum RSA. However, attackers might also have better cost.

Bernstein, Heninger, Lou, and Valenta dismiss post-quantum parameterizations of Takagi's system due to worrisome structure. In [1] they write,

> One can try to further accelerate key generation using Takagi's idea [22] of choosing n as $p^{k-1}q$. We point out two reasons that this is worrisome. The first reason is lattice attacks [2]. The second reason is that any nth power modulo n has small order, namely some divisor of $(p-1)(q-1)$; Shor's algorithm finds the order at relatively high speed once the nth power is computed.

We will elaborate on these two concerns in the remainder of this article.

1.1. **Structure of this document.** In Sections 2 and 3 we review lattice based factoring and its cost. In Sections 4 and 5 we review Shor's algorithm and its cost. In Section 6 we propose a particular exponent sequence for multi-power post-quantum RSA and examine the implications of this choice. Readers familiar with Coppersmith's technique and Shor's algorithm are encouraged to skip to Section 6 and refer to Sections 3 and 5 as needed.

1.2. **Notation.** We use standard notation, $o(\cdot)$, $O(\cdot)$, $\omega(\cdot)$, $\Omega(\cdot)$, for asymptotic growth rates. The natural logarithm is denoted log and the logarithm to base 2 is denoted lg. Euler's totient function is denoted $\phi(n)$. The bit operation cost of $(\lg n)$-bit (modular) multiplication is denoted $\mathcal{M}(\lg n)$.

2. Factoring with Coppersmith's Technique

The first concern, "lattice attacks," refers to Boneh, Durfee, and Howgrave-Graham's work [2] on factoring numbers of the form $n = p^k q$. Their algorithm is based on Coppersmith's technique, and outperforms Lenstra's elliptic curve method for large k. Coppersmith's technique is a general method for finding small modular roots of univariate polynomials. More formally, it solves some instances of the following problem.

Problem 1. *Given a rational polynomial*

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0,$$

and positive integers R and P, enumerate all integers x_0 such that $f(x_0) \equiv 0 \pmod P$ and $|x_0| < R$.

The difficulty of Problem 1 varies dramatically with R and P. The Berlekamp-Zassenhaus-van Hoeij algorithm solves Problem 1 in randomized polynomial time when the factorization of P is known. Coppersmith's technique solves some instances even when the factorization of P is not known.

We will describe a formulation of Coppersmith's technique due to Howgrave-Graham [9]. Our emphasis will be on factoring integers of the form $n = p^k q$ with p and q of known bit length and k large. Howgrave-Graham's version of Coppersmith's technique solves a generalization of Problem 1 in which only a multiple of P, rather than P itself, is given.

Problem 2. *Given a rational polynomial*

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0,$$

positive integers R and N, a real number β, and a promise that N has a divisor P of size N^β, enumerate all integers x_0 such that $f(x_0) \equiv 0 \pmod{P}$ and $|x_0| < R$.

Both variants of the problem have applications to factoring and the cryptanalysis of RSA; Alexander May has compiled an extensive survey of these applications [12].

Suppose that $n = p^k q$ and that we know some bits of p. For example, suppose that we know the high bits of p; or equivalently, that we know an integer u for which $p - u$ is a small (unknown) integer r. Then the rational polynomial

$$f(x) = (u + x)^k / n \tag{1}$$

evaluates to $1/q$ at $x = r$. Likewise $f^2(r) = 1/q^2$, $f^3(r) = 1/q^3$, and every polynomial of the form

$$h(\vec{a}; x) = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} a_{ik+j} f(x)^i x^j, \tag{2}$$

with $\vec{a} \in \mathbb{Z}^{km}$, satisfies $h(\vec{a}; r) \in q^{-(m-1)}\mathbb{Z}$. In other words, $q^{m-1} h(\vec{a}; r)$ is an integer for every $\vec{a} \in \mathbb{Z}^{km}$. Coppersmith's technique involves constructing $\vec{a}$ in such a way as to guarantee that $|h(\vec{a}; r)| < q^{-(m-1)}$ and, consequently, that $h(\vec{a}; r) = 0$.

Remarkably, such an $\vec{a}$ can be constructed in polynomial time whenever sufficiently many bits of p are known. This is made precise by Theorem 1.

Theorem 1. *Let $N = P^k Q$ for positive integers P, Q, and k with $k > 2$. Let $f(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$ with $\gcd\{N, a_0, a_1, \ldots, a_k\} = 1$. Let R be such that*

$$R < \frac{1}{2} \left(P^k / Q\right)^{1/(k+1)}.$$

Then all integers x_0 satisfying $f(x_0) \equiv 0 \pmod{P^k}$ and $|x_0| < R$ can be enumerated at a cost of

$$k^{11+o(1)} \lg N + k^{10+o(1)} \lg^2 N$$

bit operations.

2.1. **Constructing a short polynomial for fixed R.** The set of rational polynomials of degree at most d is denoted $\mathbb{Q}[x]_{\le d}$. Observe that $\mathbb{Q}[x]_{\le d}$ is a $\mathbb{Q}$-vector space of dimension $d + 1$. We equip this space with the usual coefficient norm,

$$\left\| c_0 + c_1 x + \cdots + c_d x^d \right\| = \sqrt{c_0^2 + c_1^2 + \cdots + c_d^2},$$

so that we can view the integer span of a finite collection of elements of $\mathbb{Q}[x]_{\le d}$ as a lattice.

A basis for a full rank lattice $L \subset \mathbb{Q}[x]_{\le d}$ is a set of $d + 1$ linearly independent polynomials in L. The determinant, or co-volume, of L is $\det L := |\det B|$ where B is the matrix of coefficient vectors of a basis of L.

For a rational number t the evaluation map $x \mapsto t$ is a linear functional on $\mathbb{Q}[x]_{\le d}$, which one can think of as an inner product of a coefficient vector with $(1, t, t^2, \ldots, t^d)$. Thus, for $g(x) \in \mathbb{Q}[x]_{\le d}$ we have

$$|g(t)| \le \sqrt{C_t}\, \|g(x)\| \tag{3}$$

with $C_t = \sum_{i=0}^{d} t^{2i}$. More conveniently, when $|t| \leq 1$ we have

$$|g(t)| < \sqrt{d+1}\,\|g(x)\|\,. \tag{4}$$

This simple fact, the Cauchy-Schwarz inequality, proves to be quite useful. Preserving the motivation and definition of $f(x)$ in Eq. 1, recall that we are looking for a small integer r. The definition of small is made precise by introducing a number R such that $|r| < R$. Suppose $s \in \mathbb{Z}$ and $|s| < R$, then Equation 4 can be applied, with $t = s/R$, to polynomials of the form

$$\hat{h}(\vec{a}; x) = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} a_{ik+j} f(xR)^i (xR)^j, \quad \text{with } \vec{a} \in \mathbb{Z}^{km}.$$

These polynomials form a lattice in $\mathbb{Q}[x]_{\leq km-1}$. The parameter $\vec{a}$ expresses $\hat{h}(\vec{a}; x)$ in the basis

$$H = \left\{ \begin{array}{rrrcr} 1, & (xR), & (xR)^2, & \ldots, & (xR)^{k-1}, \\ f(xR), & f(xR)(xR), & f(xR)(xR)^2, & \ldots, & f(xR)(xR)^{k-1}, \\ f(xR)^2, & f(xR)^2(xR), & f(xR)^2(xR)^2, & \ldots, & f(xR)^2(xR)^{k-1}, \\ & & \cdots & & \\ f(xR)^{m-1}, & f(xR)^{m-1}(xR), & f(xR)^{m-1}(xR)^2, & \ldots, & f(xR)^{m-1}(xR)^{k-1} \end{array} \right\}.$$

Note that H contains exactly one element of each degree from 0 to $km-1$. So H is in fact a basis for a lattice of rank km. Moreover, the corresponding matrix of coefficient vectors is triangular, so we can easily compute $|\det H|$. The leading coefficient of $f(xR)^i(xR)^j$ is R^{ik+j}/n^i, hence

$$\begin{aligned} |\det H| &= \prod_{i=0}^{m-1} \prod_{j=0}^{k-1} R^{ik+j}/n^i \\ &= R^{km(km-1)/2} n^{-km(m-1)/2}. \end{aligned} \tag{5}$$

The lattice reduction algorithm of Lenstra, Lenstra, and Lovász (LLL) can be used to find "short" elements of a lattice in polynomial time. The exact meaning of "short" is provided by the following fact.

Fact 1 (LLL [10])**.** *Let* $(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$ *be an* LLL-reduced *basis of a lattice* L*. Then*

$$\|\vec{b}_1\| \leq 2^{(d-1)/4} (\det L)^{1/d}.$$

We omit the precise definition of LLL-reduced; for now it suffices to know that an LLL-reduced basis can be produced, in polynomial time, by applying the LLL algorithm to a basis of L.

If $\hat{h}(x)$ is the first vector obtained by applying LLL to H, then

$$\|\hat{h}(x)\| \leq 2^{(km-1)/4} |\det H|^{1/km} \tag{6}$$

The question now is whether this is small enough to ensure that $\hat{h}(r/R) = 0$.

2.2. **Optimizing** R**.** We will determine the largest R for which

$$|\hat{h}(s/R)| < q^{-(m-1)} \tag{7}$$

for all $s \in \mathbb{Z}$ with $|s| < R$. When Eq. 7 is satisfied, we are able to recover r, and factor n, by computing the rational roots of $\hat{h}(x)$. Equations 4, 5, and 6 imply

$$\begin{aligned} |\hat{h}(s/R)| &< \sqrt{km}\ \|\hat{h}(x)\| \\ &= \sqrt{km}\ 2^{(km-1)/4} R^{(km-1)/2} n^{-(m-1)/2}. \end{aligned}$$

Observe that $\sqrt{km}\ 2^{(km-1)/4} < 2^{(km-1)/2}$ for integers k and m with $km > 7$. So Equation 7 is satisfied, for $km > 7$, when

$$(2R)^{(km-1)/2} < n^{(m-1)/2} q^{-(m-1)}. \tag{8}$$

Taking logarithms we have

$$\lg 2R < \frac{m-1}{km-1} \left(\lg n - 2 \lg q\right). \tag{9}$$

The bound on R in the statement of Theorem 1 is recovered by taking $m = k$ and writing $\lg n = k \lg p + \lg_p q$.

3. The cost of lattice attacks.

We will now consider the cost of using Theorem 1 to factor n of the form $n = p_1^{e_1} p_2^{e_2} \dots p_L^{e_L}$. For simplicity we will assume that the prime factors of n are of equal bit length and that the e_i are in decreasing order.

Let $E = \sum_{i=1}^{L} e_i$. Suppose $E \geq 5$, so that there exist integers P, Q, and k with $k > 2$ and $n = P^k Q$. Let R be as in Theorem 1. An attacker has to guess $\lg P - \lg R$ bits of P to construct an $f(x)$, as in Equation 1, to which Theorem 1 can be applied. With the upper bound on R from Theorem 1 we have

$$\begin{aligned} \lg P - \lg R &> \lg P - \frac{1}{k+1} \left(k \lg P - \lg Q\right) \\ &= \frac{1}{k+1} (\lg P + \lg Q). \end{aligned} \tag{10}$$

The cost of checking a guess is dominated by LLL. Thus, factoring $n = P^k Q$ with Coppersmith's technique costs

$$2^{\lg P - \lg R} \cdot \mathrm{poly}(k, N) > (PQ)^{1/(k+1)} \cdot \mathrm{poly}(k, N) \tag{11}$$

bit operations, where poly(k, N) is the cost of lattice reduction. The expected cost of guessing $\lg P - \lg R$ bits of P will be comparable to the cost of ECM when $\lg P - \lg R \approx \sqrt{\lg p_1}$. Note that $\lg P + \lg Q$ is at least $(\lg p_1)L$, since each prime dividing n must occur to some power in P or Q. If L is constant with respect to n, as it is for $n = p^k q$, the condition to beat ECM is $k = \Omega(\sqrt{\lg p_1})$, as was reported in [2]. However, if L grows with n, then one needs $k = \Omega(L\sqrt{\lg p_1})$.

The LLL algorithm costs $d^{5+o(1)}(\lg B)^{2+o(1)}$ bit operations where d is the lattice rank and $\lg B$ is the number of bits needed to represent the longest vector of the input basis. The $o(1)$ terms assume fast multiplication techniques.

Much of the algorithmic research on lattice reduction focuses on improving the constant $2^{(d-1)/4}$ in Fact 1. In Equation 8 we discarded any impact these algorithms could have, with little consequence.

On the other hand, Stehlé and Nguyen's L^2 variant of LLL [13] does seem to help. They use (carefully analyzed) floating point arithmetic to produces an LLL-reduced basis in $d^{5+o(1)} \lg B + d^{4+o(1)} \lg^2 B$ bit operations. The complexity claim in the statement of Theorem 1 is recovered with $m = k$, $d = k^2$, and $\lg B = O(k \lg N)$.

The cost can also be improved by using Novocin, Stehlé, and Villard's $\tilde{L}^1$ variant of LLL [14]. That algorithm costs $d^{5+o(1)} \lg B + d^{\omega+1+o(1)} (\lg B)^{1+o(1)}$ bit operations, where $\omega < 2.373$ is the matrix multiplication exponent.

4. Shor's Algorithm.

We turn now to quantum attacks. We will describe Shor's algorithm without the usual machinery of unitary transformations on complex euclidean space. Instead we break the algorithm into four steps: initialization, randomization, modular exponentiation, and the quantum Fourier transform. We give a high level description of each step, the cost of each step, and the probability of events that we label "$R_j^{(i)} = k$." The event "$R_j^{(i)} = k$" should be read as "terminating Shor's algorithm after step i and reading qubit register j yields outcome k."

The computation involves two qubit registers. The first register represents an element of $\mathbb{Z}/S\mathbb{Z}$, for an S that our analysis will determine later. The second register represents an element of $\mathbb{Z}/n\mathbb{Z}$, where n is the number to be factored. Each step of the algorithm may use "scratch" registers as well, but we assume that these scratch registers are statistically independent from the data registers in between steps[1]. The cost of each step is given as the depth of a quantum circuit that performs that step. Circuit depth is measured in elementary single- and two-qubit gates, hereafter "qubit operations."

Shor's algorithm computes the multiplicative order of 3 modulo n, i.e the smallest r such that $3^r \equiv 1 \pmod{n}$. If r is even, and $3^{r/2} \not\equiv -1 \pmod{n}$, then $\gcd(3^{r/2} - 1,\ n)$ is a non-trivial factor of n. If r is odd, or $3^{r/2} \equiv -1 \pmod{n}$, then one tries again with a different constant in place of 3. The algorithm proceeds as follows.

Step 1: The registers are initialized to 0, i.e.

$$\Pr[R_1^{(1)} = 0 \wedge R_2^{(1)} = 0] = 1.$$

The cost of this step is omitted.

Step 2: The first register is transformed so that it represents the uniform distribution on $\mathbb{Z}/S\mathbb{Z}$:

$$\Pr[R_1^{(2)} = x \wedge R_2^{(2)} = 0] = 1/S$$

for all $x \in \mathbb{Z}/S\mathbb{Z}$. When S is a power of 2 the circuit for this operation – parallel Hadamard gates on each qubit of the first register – has depth 1.

Step 3: The value $\mathsf{modexp}(3, x, n) := 3^x \bmod n$ is calculated and stored in the second register. The transformation $(x, 0) \mapsto (x, \mathsf{modexp}(3, x, n))$ is a bijection, so this does not affect the probability of finding x in the first register, i.e.

$$\Pr[R_1^{(3)} = x \ \wedge \ R_2^{(3)} = \mathsf{modexp}(3, x, n)] = 1/S$$

[1]Standard reversible computing techniques make this a benign assumption, though it is slightly stronger than what is needed.

for all $x \in \mathbb{Z}/S\mathbb{Z}$. This operation costs $(\lg S) \cdot \mathcal{M}(\lg n)$ qubit operations, where $\mathcal{M}(\lg n)$ is the cost of $(\lg n)$-bit modular multiplication. It is reasonable to expect that quantum modular multiplication will cost $(\lg n)^{1+o(1)}$ qubit operations, matching the classical bit operation cost[2].

Step 4: The (approximate) $\mathbb{Z}/S\mathbb{Z}$ quantum Fourier transform is applied to the first register. This costs $(\lg S)^{1+o(1)}$ qubit operations [5]. The effect of this transformation is best described by the distribution of $R_1^{(4)}$ conditioned on $R_2^{(4)} = 3^k \bmod n$. Since 3 is of multiplicative order r we may assume $0 \leq k < r$. Then for $y \in \mathbb{Z}/S\mathbb{Z}$ Shor shows that

$$\Pr[R_1^{(4)} = y \mid R_2^{(4)} = 3^k \bmod n] = \left| \frac{\sqrt{r}}{S} \sum_{b=0}^{\lfloor \frac{S-k-1}{r} \rfloor} \exp\left(2\pi i b \tfrac{ry}{S}\right) \right|^2 . \tag{12}$$

We will say that y is *good* if there exists an integer d such that

$$\left| \frac{y}{S} - \frac{d}{r} \right| \leq \frac{1}{2S}. \tag{13}$$

Shor shows that for good y the right hand side of Equation 12 is at least $1/3r$. If $S > 2r$ then there are at least $\phi(r)$ pairs (d, y) for which d/r is in lowest terms and y is good. Hence, for $S > 2r$, the total probability assigned to the set of good y is at least $\phi(r)/3r = O(1/\log\log r)$.

The algorithm terminates after step 4. Shor's strategy for recovering r from the known quantities y and S is to compute the convergents of the continued fraction expansion of y/S. If $S > r^2$ then Equation 13 is a sufficient condition (well known in the theory of Diophantine approximation) for d/r to appear among the convergents. For S of this size there are also techniques that amplify the success probability from $O(1/\log\log r)$ to $O(1)$ through classical post-processing alone [20]. We ignore the bit operation cost of post-processing, including the cost of computing $\gcd(3^{r/2} - 1, n)$.

5. The cost of period finding.

The $(\lg S) \cdot \mathcal{M}(\lg n)$ qubit operations in step 3 are the dominant cost of Shor's algorithm. For general integers, Shor recommends taking S to be the power of two between n^2 and $2n^2$. However, for certain integers one can take S slightly smaller.

This brings us to the second reason that Bernstein, Heninger, Lou, and Valenta find $n = p^k q$ to be worrisome. Since $\phi(n) = p^{k-1}(p-1)(q-1)$ the multiplicative order of any nth power in $(\mathbb{Z}/n\mathbb{Z})^\times$ divides $(p-1)(q-1)$. Hence, we may take $S = ((p-1)(q-1))^2 \approx n^{4/(k+1)}$ so long as we by replace the constant 3 in Step 3 with $3^n \bmod n$. The cost of computing $3^n \bmod n$ is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations. Following this, the cost of Step 3 of Shor's algorithm is $O(\frac{\lg n}{k+1}) \cdot \mathcal{M}(\lg n)$ qubit operations. In other words, "Shor's algorithm finds the order at relatively high speed once the nth power is computed" [1].

The same argument applies to more general multi-power moduli $n = p_1^{e_1} p_2^{e_2} \dots p_L^{e_L}$. The adversary can take $S = n^{2L/E}$ where $E = \sum_i e_i$. The asymptotic cost of factoring such n by Shor's algorithm is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations plus

[2]State of the art quantum circuits for mod-n multiplication cost $32(\lg n)^2 + O(\lg n)$ qubit operations [7], but there are no fundamental barriers to implementing fast multiplication.

$O(\frac{L}{E}\lg n)\cdot\mathcal{M}(\lg n)$ qubit operations. If $E-L$ is large, and qubit operations are much more expensive than bit operations, this could be a dramatic improvement.

6. Multi-power pqRSA

The multi-prime post-quantum RSA system proposed in [1] uses fast multiplication techniques to reduce the cost of users' operations when $n=p_1p_2\cdots p_\ell$ and ℓ grows as $\lg n/(\lg\lg n)^{2+o(1)}$. The multi-power post-quantum RSA system we propose here applies similar techniques to reduce the cost of users' operations when $n=p_1^{e_1}p_2^{e_2}\cdots p_L^{e_L}$ and L grows as $(\lg n)^{1/(2+o(1))}$. Our decryption procedure follows [22] and [11] in using a Newton iteration to compute a cube root modulo $p_i^{e_i}$ from a cube root modulo p_i.

The constraints on parameter selection for multi-prime pqRSA come from ECM and Shor's algorithm. When repeated factors are allowed we also need to ensure that lattice attacks cost more than ECM and that special purpose variants of Shor's algorithm do not eliminate our cost/performance ratio.

6.1. **Parameters.** We take

$$n=p_1^2p_2^3p_3^5p_4^7\cdots p_i^{\pi_i}\cdots p_L^{\pi_L}$$

where π_i is the ith prime. As in [1], we take each prime factor of n to be of bit length $(\lg\lg n)^{2+o(1)}$. We also take $p_i\equiv 2 \pmod 3$ for $i=1,\ldots,L$ so that encryption exponent 3 can be used.

We define $E(L)=\sum_{i\le L}\pi_i$. The prime number theorem suggests that $E(L)\sim\frac{1}{2}L^2\log L$. To obtain keys of a specified bit-length we choose L such that $E(L)=\lg n/(\lg\lg n)^{2+o(1)}$.

For concreteness we will focus on 1-terabyte keys, $\lg n=2^{43}$. We follow [1] and fix the bit length of the p_i at 4096 bits. We then take $L=20044$, which gives $E(20044)=2^{31.0001\ldots}$.

6.2. **Security.**

6.2.1. *Small factor attacks.* The asymptotic security analysis of multi-power pqRSA with respect to trial division, Pollard's $p-1$ method, Williams' $p+1$ method, and Lenstra's elliptic curve method is identical to multi-prime pqRSA. This is also true for variants of these algorithms that use Grover's algorithm as a subroutine. We refer to [1] for a detailed discussion of why these attacks are not likely to succeed at a cost of less than 2^{140} bit operations when 4096-bit primes are used.

Note that Peralta and Okamoto [15] describe a method to speed up ECM for moduli of the form $n=p^2q$. This method does not improve on the asymptotic $\exp((\log p)^{1/2+o(1)})$ cost estimate for ECM that is used here.

6.2.2. *Lattice attacks.* The lattice attack in Section 2 is competitive with ECM when the attacker needs to guess fewer than $\sqrt{\lg p_1}=(\lg\lg n)^{1+o(1)}$ bits. We will argue that there do not exist integers P, Q, and k, with $n=P^kQ$, for which the lattice attack is competitive with ECM. From Equation 10, we see that P, Q, and k must satisfy

$$\frac{1}{k+1}(\lg P+\lg Q)\le(\lg\lg n)^{1+o(1)}. \tag{14}$$

Observe that $p_1^2p_2^3\cdots p_j^{\pi_j}\mid Q$ for j such that $\pi_j<k$. The sum of the primes less than k grows at least as fast as the sum of the first $k/\log k$ integers. In particular

the sum dominates k. Hence $\lg Q = \omega((\lg\lg n)^2 k)$ and we see that Eq. 14 cannot be satisfied. It is a simple exercise to show that replacing Equation 14 with a condition based on Equation 9 does not improve the situation.

Even if a satisfying k did exist, the cost of LLL would be prohibitive. Theorem 1 suggests a cost of well over $k^{10} \lg n$ bit operations. Taking k smaller than L is unrealistic, as $\lg P + \lg Q$ grows super-linearly with L. With $\lg n = 2^{43}$ and $L = 20044$ we have $L^{10} \lg n > 2^{185}$.

The cost of lattice reduction becomes an enormous obstacle if one attempts to guess bits of P using Grover search.

6.2.3. *Period finding.* As described in Section 5, one can reduce the cost of period finding by replacing the constant 3 in Shor's algorithm with $3^n \bmod n$. The resulting cost is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations plus $(\lg S) \cdot \mathcal{M}(\lg n)$ qubit operations where $S = n^{2L/E}$. For our multi-power pqRSA moduli $\lg S = (\lg n)^{1/2+o(1)}$, so the cost of Shor's algorithm is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations plus $(\lg n)^{1.5+o(1)} \cdot \mathcal{M}(\lg n)$ qubit operations.

Bernstein, Heninger, Lou, and Valenta [1] estimate $\mathcal{M}(\lg n)$ using contemporary speed records for multiplication in $\mathbb{F}_{2^{60}}$. Assuming that integer multiplication can be made as fast as binary polynomial multiplication, that reduction modulo n is free, and that bit operations have the same cost as qubit operations, they suggest that $\mathcal{M}(2^{43}) \approx 2^{56}$ bit (or qubit) operations.

For our 1-terabyte multi-power pqRSA parameters $S = (2^{4096L})^2 \approx 2^{2^{27}}$. Consequently, Shor's algorithm costs $(\lg n) \cdot \mathcal{M}(2^{43}) \approx 2^{99}$ bit operations plus $(\lg S) \cdot \mathcal{M}(2^{43}) \approx 2^{83}$ qubit operations. Compare this with the cost of attacking 1-terabyte multi-prime pqRSA: with $S = n^2$ Shor's algorithm costs $(\lg n) \cdot \mathcal{M}(2^{43}) \approx 2^{100}$ qubit operations.

6.3. **Efficiency.** Apart from decryption, the user's computations in multi-power pqRSA are identical to the user's computations in multi-prime pqRSA. We refer the reader to [1] for the justification that key generation and encryption take $(\lg n)^{1+o(1)}$ bit operations.

Multi-power pqRSA decryption may be somewhat faster than multi-prime pqRSA decryption, but we do not expect a significant asymptotic improvement. To decrypt a ciphertext c, one first uses a remainder tree to compute $c \bmod p_i^{\pi_i}$ for $i = 1, \ldots, L$. One then computes the cube root of c modulo each p_i. The decryption algorithm differs from multi-prime pqRSA in that there are two distinct interpolation steps. The first, as in [22, 11], lifts the cube root of c modulo p_i to a cube root of c modulo $p_i^{\pi_i}$ for $i = 1, \ldots, L$. The second uses a CRT tree to construct the cube root of c modulo n. If there is any speedup for multi-power pqRSA decryption, it is likely due to the fact that fewer cube roots need to be computed. The two interpolation steps together are likely as expensive as the CRT tree in multi-prime pqRSA. Regardless, decryption will cost $(\lg n)^{1+o(1)}$ bit operations.

We have not implemented multi-power pqRSA, and can only estimate its non-asymptotic efficiency. Bernstein, Heninger, Lou, and Valenta report a prime generation rate of between 750 and 1585 primes per core-hour. They also report that a set of 2^{31} primes was generated in 1 975 000 core-hours [1]. At the same rate, generating the 20 044 primes needed for a 1-terabyte multi-power pqRSA key would take only 18.5 core-hours. Keep in mind that the full key generation procedure also involves a large product tree.

6.3.1. *Cost/performance ratio.* Multi-prime pqRSA key generation, encryption, and decryption all cost $(\lg n)^{1+o(1)}$ bit operations. The best attack costs $(\lg n)^{2+o(1)}$ qubit operations. As there is no reason to believe that qubit operations are *less* expensive than bit operations, we can safely say that there is a quadratic gap between the attacker's cost and the user's cost. The gap could be larger if qubit operations are fundamentally more expensive than bit operations.

Multi-power pqRSA key generation, encryption, and decryption, likewise, cost $(\lg n)^{1+o(1)}$ bit operations. The best attack costs $(\lg n)^{2+o(1)}$ bit operations plus $(\lg n)^{1.5+o(1)}$ qubit operations. If qubit operations cost the same as bit operations, then multi-power pqRSA users still enjoy a quadratic advantage against attackers. This remains true when qubit operations cost C bit operations, for any constant C. The quadratic advantage is only threatened when the cost of qubit operations grows with n.

7. Conclusion

One-terabyte multi-prime pqRSA key generation is quite costly. In one test, prime generation took "four months running on spare compute capacity of a 1,400-core cluster" and evaluating the product tree took "about four days" [1]. A user with a comparable machine should be able to generate a multi-power pqRSA key in just 5 days. The speedup comes with some loss in security, but quantifying this loss presents challenges.

In Section 6.2.3 we saw that if bit operations and qubit operations have equal cost, then 1-terabyte multi-power pqRSA users suffer a 1 bit security loss. However, if qubit operations are much more expensive than bit operations, these users suffer a 17 bit security loss – albeit from a higher initial security level.

The security gap could be amplified in other cost models. For instance, if qubit operations cost $O(\lg n)$ bit operations, then the cost of attacking multi-prime pqRSA is $(\lg n)^{3+o(1)}$ bit operations, while the cost of attacking multi-power pqRSA is only $(\lg n)^{2.5+o(1)}$ bit operations. This may seem unrealistic, but there are models where each qubit operation in Shor's algorithm costs $O(\lg n)$ bit operations.

In [6] the cost of a quantum circuit is expressed in "surface code cycles." Each surface code cycle involves error correction that can be assigned a cost in bit operations. The amount of error correction that needs to be done grows with the number of qubits in the quantum circuit. Hence, the bit operation cost of quantum computation, in this model, scales with both the number of qubits and the depth of the circuit.

Shor's algorithm needs at least $\lg S + \lg n$ qubits for the registers described in Section 4. This is $3 \lg n$ for multi-prime pqRSA, and only slightly more than $\lg n$ for multi-power pqRSA. Suppose the cost of error correction is 2^c bit operations per qubit per qubit operation (possibly with c negative). Then the error correction needed for an attack on multi-prime pqRSA costs $2^{143+\lg 3+c}$ bit operations, and the error correction needed for an attack on multi-power pqRSA modulus costs 2^{126+c} bit operations.

Questions about the relative cost of bit and qubit operations will need to be answered whether users choose multi-prime pqRSA, multi-power pqRSA, or some other system. Even if users decide that no variant of pqRSA is fast enough for their needs, a robust quantum cryptanalysis of RSA will greatly improve our understanding of the post-quantum security of other systems.

References

[1] Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. In *International Workshop on Post-Quantum Cryptography*, pages 311–329. Springer, 2017.

[2] Dan Boneh, Glenn Durfee, and Nick Howgrave-Graham. Factoring $N = p^r q$ for large r. In *Annual International Cryptology Conference*, pages 326–337. Springer, 1999.

[3] Dan Boneh and Hovav Shacham. Fast variants of RSA. RSA Laboratories' CryptoBytes, vol 5, no 1, pp 1–9, 2002.

[4] T. Collins, D. Hopkins, S. Langford, and M. Sabin. Public key cryptographic apparatus and method. US Patent #5,848,159, Jan. 1997.

[5] Don Coppersmith. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002.

[6] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.

[7] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n+ 2 qubits with Toffoli based modular multiplication. *arXiv preprint arXiv:1611.07995*, 2016.

[8] M. Jason Hinek. On the security of multi-prime RSA. *Journal of Mathematical Cryptology*, 2(2):117–147, 2008.

[9] Nicholas Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *IMA International Conference on Cryptography and Coding*, pages 131–142. Springer, 1997.

[10] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[11] Seongan Lim, Seungjoo Kim, Ikkwon Yie, and Hongsub Lee. A generalized Takagi-cryptosystem with a modulus of the form $p^r q^s$. In Bimal Roy and Eiji Okamoto, editors, *Progress in Cryptology —INDOCRYPT 2000*, pages 283–294. Springer, 2000.

[12] Alexander May. Using LLL-reduction for solving RSA and factorization problems. In *The LLL algorithm*, pages 315–348. Springer, 2009.

[13] Phong Q Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[14] Andrew Novocin, Damien Stehlé, and Gilles Villard. An LLL-reduction algorithm with quasi-linear time complexity: Extended abstract. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 403–412, New York, NY, USA, 2011. ACM.

[15] René Peralta and Eiji Okamoto. Faster factoring of integers of a special form. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 79(4):489–493, 1996.

[16] J-J Quisquater and Chantal Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics letters*, 18(21):905–907, 1982.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[18] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Cryptographic communications system and method. US Patent #4,405,829, Dec. 1977.

[19] Adi Shamir. RSA for paranoids. RSA Laboratories' CryptoBytes, vol 1, no 3, pp 1–4, 1995.

[20] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[21] Tsuyoshi Takagi. Fast RSA-type cryptosystems using n-adic expansion. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 372–384. Springer, 1997.

[22] Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 318–326. Springer, 1998.