Security Level of Cryptography —
Integer Factoring Problem (Factoring $N = p^2q$)

December 2001

Contents

Summary **2**

Detailed Evaluation **3**

1 The Elliptic Curve Method **3**

1.1 The ECM applied to $N = p^dq$ 4

1.2 Running time estimates 5

2 The Lattice Factoring Method **7**

3 Factoring Using Jacobi Symbols **8**

4 Conclusions: Recommended parameter sizes for $N = p^dq$ **9**

References **10**

Summary

Integers of the form $N = p^d q$ have found many application in cryptography. For example,

- Okamoto et al. use $N = p^2 q$ to give a fast digital signature scheme [12, 10, 3],
- Okamoto and Uchiyama [13] use $N = p^2 q$ to build an elegant public-key system, and
- Takagi [16] shows how to use $N = p^d q$ to speed-up RSA decryption.

This report analyzes the effectiveness of various factoring techniques for factoring integers of the form $N = p^2 q$ (and more generally integers of the form $N = p^d q$). We study the effectiveness of three factoring techniques:

1. The Elliptic Curve Method (ECM) [7] augmented by Okamoto and Peralta [11],
2. The Lattice Factoring Method (LFM) [1], and
3. Factoring $N = p^2 q$ using Jacobi symbols.

Our conclusion is that the LFM method and the Jacobi symbol method cannot currently be used to attack $N = p^2 q$ when N is at least 1024-bits long. On the other hand, the ECM method with the improvements of Okamoto and Peralta poses a threat that should be addressed. When using $N = p^2 q$ one would like to ensure that the modulus size is sufficiently large to provide the same level of security as 1024-bit standard RSA. Hence, our conclusions are three fold:

1. 1024-bit integers of the form $N = p^2 q$ provide adequate security in the short term. However, within 5 to 10 years the safety margins provided by such integers will likely become unacceptable.
2. To provide security comparable to standard 1024-bit RSA it is recommended to use at least a 1280-bit $N = p^2 q$ modulus. We estimate that the running time for ECM on such a modulus is comparable to the running time of the Number Field Sieve on 1024-bit RSA (note that we are ignoring memory-space considerations).
3. 1024-bit integers of the form $N = p^3 q$ should be considered insecure and should not be used in deployed systems. They are susceptible to a factoring attack using the ECM method.

We also show that integers of the form $N = p^2 q$ have more structure than standard RSA integers. We show that this structure may lead to new factoring algorithms specifically targeted at $N = p^2 q$. Hence, factoring $N = p^2 q$ might be much easier than factoring $N = pq$ of the same size. Currently we cannot exploit this extra structure of $N = p^2 q$ to design new factoring algorithms. Therefore this should be viewed as a theoretical threat rather than a real attack.

Detailed Evaluation

The report is organized as follows. We begin by analyzing the effectiveness of the Elliptic Curve Method (ECM) for factoring integers of the form $N = p^d q$. We mostly focus on square integers $N = p^2 q$. We then estimate the running time of the ECM and deduce some bounds on the size of $N = p^2 q$ for proper security. Next, we consider the Lattice Factoring Method (LFM) and show that it does not pose a threat to sufficiently large square integers. Finally, in Section 3 we discuss a hypothetical approach to factoring square integers using Jacobi symbols and conclude that factoring such numbers could be easier than factoring general RSA integers.

Note that we do not explicitly analyze the performance of the Number Field Sieve (NFS) on integers of the form $N = p^d q$. The NFS will take the same amount of time to factor $N = p^2 q$ as it will to factor $N = pq$ of the same size. Hence, the NFS does not pose a threat to $N = p^2 q$ beyond the threat that it poses to standard RSA integers. Throughout the report we use an estimated running time of the NFS on a standard 1024-bit RSA integer to assess the security of integers $N = p^2 q$ and $N = p^3 q$ for various sizes of N.

Notation: Whenever we write $N = p^2 q$ we assume that p and q are primes of approximately the same size, i.e. $p, q \approx \sqrt[3]{N}$. Similarly, when we write $N = p^3 q$ we assume p and q are primes of approximately the same size, i.e. $p, q \approx \sqrt[4]{N}$.

1 The Elliptic Curve Method

The Elliptic Curve factoring Method (ECM) [7] is a general-purpose factoring algorithm that performs well when the smallest prime factor is relatively small. Hence, it is natural to analyze its performance on $N = p^2 q$ since p and q are relatively small, i.e. $p, q \ll \sqrt{N}$. When p is the smallest prime factor of N the asymptotic running time of ECM is:

$$L_{1/2,\sqrt{2}}(p) = \exp\left((1.41 + o(1)) \ln(p)^{1/2} (\ln \ln(p))^{1/2}\right) \tag{1}$$

The term $o(1)$ tends to 0 as p tends to infinity. Several practical speed-ups to the ECM give an (approximately) constant factor improvement [8] over this asymptotic running time.

The ECM requires little memory-space to run. Asymptotically ECM only requires linear space. Other factoring methods such as the NFS require large amounts of memory size (though sub-exponential in the size of the number being factored). Hence, a pure time comparison between ECM and NFS is a bit inaccurate since for a large factoring project (e.g. factoring RSA-1024) the space requirements of NFS are prohibitive. In our estimates we make a note of the difference in space requirements, but mostly rely on running-time for comparison between different techniques.

The current record in ECM factoring belongs to Miyamoto [15] who succeeded in finding a 183-bit factor of a 372-bit number. This project took about 13 days using a single 800MhZ Pentium III. We will use this result as a base point for estimating the running-time of ECM on large integers $N = p^d q$.

1.1 The ECM applied to $N = p^d q$

We briefly describe a simple variant of the ECM method and then discuss its adaptation to integers of the form $N = p^d q$.

Let B_1, B_2 be two smoothness bounds. Typically B_1 is set to $B_1 = L_{1/2,\sqrt{2}/2}(p)$ and B_2 is chosen as some larger value, e.g. $B_2 = 10B_1$. The ECM algorithm factors an integer $N = p^d q$ by iterating the following steps until N is factored:

Init: Pick a random point $T = (x_0, y_0) \in Z_N^2$ and choose random $a, b \in \mathbb{Z}_N$ such that

$$y_0^2 = x_0^3 + ax_0 + b \pmod{N}$$

We refer to the set of points $(x, y) \in \mathbb{Z}_N^2$ satisfying $y^2 = x^3 + ax + b \pmod{N}$ as an elliptic curve $E/\mathbb{Z}_N$. If we reduce a, b modulo q we get an elliptic curve $E/\mathbb{F}_q$. It is well known that the set of points on $E/\mathbb{F}_q$ forms an Abelian group. There is a natural map $\phi_q : E/\mathbb{Z}_N \to E/\mathbb{F}_q$ defined by $\phi_q((x, y)) = (x \bmod q, y \bmod q)$. Let m_q be the (unknown) order of $\phi_q(T)$ in the group $E/\mathbb{F}_q$. Similarly, let m_p be the (unknown) order of $\phi_p(T)$ in the group $E/\mathbb{F}_p$.

Power: Let $L \in \mathbb{Z}$ be the product of all prime powers less than B_1. Compute $R = L \cdot T \in E/\mathbb{Z}_N$ by using the addition law on the curve $E : y^2 = x^3 + ax + b$. This step takes time $O(B_1 \log B_1 \log^2 N)$.

Search: Suppose m_q is a small multiple of a B_1-smooth number. More precisely, suppose $m_q = M \cdot m$ where M has no prime factor larger than B_1 and $m < B_2$. Then the point $\phi_q(R) \in E/\mathbb{F}_q$ has order dividing m on the curve $E/\mathbb{F}_q$. Since $m < B_2$ we know that $\phi_q(R)$ has order less than B_2 in $E/\mathbb{F}_q$. We assume that $\phi_p(R) \in E/\mathbb{F}_p$ has order much larger than B_2 on $E/\mathbb{F}_p$.

Now, suppose we had two integers $\alpha_1, \alpha_2 \in [0, B_2]$ such that $\alpha_1 = \alpha_2 \bmod m$. Let $R_1 = (x_1, y_1) = \alpha_1 R$ and $R_2 = (x_2, y_2) = \alpha_2 R$. Then $\phi_q(R_1) = \phi_q(R_2)$, but $\phi_p(R_1) \neq \pm\phi_p(R_2)$. Hence, $x_1 = x_2 \bmod q$, but $x_1 \neq x_2 \bmod p$. We then obtain the factorization of N by simply computing $\gcd(x_1 - x_2, N) = q$.

Since m is not known to us the question is how to find such α_1, α_2. The simplest approach is to pick $k = 2\lceil\sqrt{B_2}\,\rceil$ random integers $\alpha_1, \ldots, \alpha_k \in [0, B_2]$. Then the birthday paradox implies that with high probability there exists $0 \leq u, v \leq k$ such that $\alpha_u = \alpha_v \bmod m$. To find the pair u, v we compute $R_i = (x_i, y_i) = \alpha_i R$ for all $i = 1, \ldots, k$. We know that $\gcd(x_u - x_v, N) = q$. Hence, given $x_1, \ldots, x_k \in \mathbb{Z}_N$ we need to find u, v satisfying this gcd relation. Naively we can do this by trying all pairs $0 \leq u, v \leq k$. This takes time $O(B_2 \log^2 N)$.

Montgomery and Silverman [9] and Pollard proposed better algorithms for this last step using FFT. Their algorithm takes $x_1, \ldots, x_k \in \mathbb{Z}_N$ as input and finds a pair u, v so that $\gcd(x_u - x_v, N) = q$ in time $O(\sqrt{B_2} \log B_2 \log^2 N)$. Asymptotically this is much faster than the naive method. Unfortunately, the constants hidden by the big-O are relatively large making this method less attractive.

The three steps above are repeated until N is factored. The expected number of iterations is $L_{1/2,\sqrt{2}/2}(p)$.

The Okamoto-Peralta improvement: When $N = p^2 q$ Okamoto and Peralta [11] propose an improvement to the Search step in the above algorithm. Recall that the problem to solve is given $x_1, \ldots, x_k \in \mathbb{Z}_N$ find a pair u, v satisfying $\gcd(x_u - x_v, N) = q$. To do so, define the signature of an element $x \in \mathbb{F}_q$ as

$$\text{sig}(x) = \left[\left(\frac{x}{q}\right), \left(\frac{x+1}{q}\right), \ldots, \left(\frac{x+b}{q}\right) \right] \in \{\pm 1, 0\}^{b+1}$$

where $b = \lfloor \log_2 q \rfloor$ and $\left(\frac{x}{q}\right)$ is the Legendre symbol of $x \bmod q$. It is conjectured that $\text{sig}(x)$ uniquely defines $x \in \mathbb{F}_q$, namely if $x \neq y$ then $\text{sig}(x) \neq \text{sig}(y)$.

The main point of the Okamoto-Peralta improvement is that when $N = p^2 q$ we can efficiently compute the signature of $x_1, \ldots, x_k \in \mathbb{Z}_N$ even though q is unknown. To see this observe that when $N = p^2 q$ the Jacobi symbol of $x \in \mathbb{Z}_N^*$ satisfies

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p^2}\right) \cdot \left(\frac{x}{q}\right) = \left(\frac{x}{p}\right)^2 \cdot \left(\frac{x}{q}\right) = \left(\frac{x}{q}\right)$$

Hence, we can compute the signature of $x_i \bmod q$ by computing the Jacobi symbols $\left(\frac{x_i + j}{N}\right)$ for $j = 1, \ldots, b$. Once we compute the signatures of all the x_i's modulo q we can easily find u, v such that $\text{sig}(x_u) = \text{sig}(x_v) \bmod q$ in time $O(k \log k)$. Then $x_u = x_v \bmod q$. Hence, when $N = p^2 q$ the third step of ECM now takes time $O(\sqrt{B_2} \log B_2 \log^2 N)$ where the constant in the big-O notation is quite small. This means we can use a much larger value of B_2. Okamoto and Peralta claim that this method speeds up the ECM by approximately a factor of 50 for the parameter sizes we are interested in.

Potential defense against Okamoto-Peralta: We note that the Okamoto-Peralta improvement only applies when ECM is close to finding the prime factor q of $N = p^2 q$. Hence, for proper security it may make sense to increase the size of q by a few bits at the cost of reducing the size of p. For example, it may make sense to increase the size of q by 10 bits and reduce the size of p by 5 bits. This has no effect on the size of $N = p^2 q$, but it nullifies the Okamoto-Peralta improvement to ECM since finding q is now harder.

ECM improvements for $N = p^d q$: The Okamoto-Peralta improvement for $N = p^2 q$ generalizes to $N = p^d q$. To do so one defines the signature of $x \bmod N$ using the d'th power residue symbol [4, p. 204]. For a given $x \in \mathbb{Z}_N$ one computes the signature of $x \bmod q$ using the Eisenstein reciprocity law [4, p. 207]. We do not give the details here.

1.2 Running time estimates

As mentioned earlier, the current record in ECM factoring belongs to Miyamoto [15] who succeeded in finding a 183-bit factor of a general 372-bit number in 13 days using a single 800MhZ Pentium III. We will use this result as a base point for estimating the running-time of ECM on large integers $N = p^d q$.

We estimate the running time of ECM for factoring (1) 1024-bit $N = p^2q$, (2) 1280-bit $N = p^2q$, and (3) 1024-bit $N = p^3q$. Our estimates show that 1024-bit $N = p^3q$ is insecure by today's standards and should not be used in deployed systems. 1024-bit $N = p^2q$ is not as strong as standard 1024-bit RSA, but may leave sufficient safety margins for short term use. However, within five to ten years 1024-bit $N = p^2q$ is likely to become unacceptable. On the other hand 1280-bit $N = p^2q$ seems to offer similar security to 1024-bit standard RSA and can be adequately deployed.

1. Factoring 1024-bit $N_1 = p^2q$: Here the prime factor q is of order $1024/3 = 342$-bits. We use Equation 1 to compare the running for factoring such N_1 with the result of Miyamoto. The direct ratio of running times is:

$$\frac{L_{1/2,\sqrt{2}}(2^{342})}{L_{1/2,\sqrt{2}}(2^{183})} = 7.76 \cdot 10^6$$

Note that we are ignoring the $o(1)$ term in Equation 1. Instead, we manually adjust our estimate to take into account that arithmetic modulo a 1024-bit number takes longer than arithmetic modulo a 372-bit number (the size of the modulus used by Miyamoto). Assuming all arithmetic operations take quadratic time, we need to add a factor of $(1024/372)^2 = 7.57$ to the estimate. This implies that finding a 342-bit factor of a 1024-bit modulus should take approximately $5.88 \cdot 10^7$ times the work of finding a 183-bit factor of a 372-bit modulus. To be conservative we include the factor of 50 improvement in ECM running time due to Okamoto-Peralta. Hence, factoring a 1024-bit $N_1 = p^2q$ should take approximately 42000 years on a single 800MhZ Pentium III.

We compare the effort of factoring N_1 using ECM to the effort of factoring RSA-155, a 512-bit integer [2]. RSA-155 was factored using the Number Field Sieve in 3.7 months using about 300 machines. We see that factoring N_1 using ECM takes about 460 times the work of factoring RSA-155 using NFS. These numbers are only an estimate based on the asymptotic formulas and could be off by as much as a factor of 10. So, suppose that finding a 342-bit factor using ECM is 4000 times the effort of factoring RSA-155 using NFS. This safety margin might be acceptable today, but may not be acceptable in 5 to 10 years given the rate of advance in both hardware performance and hardware availability. We note that factoring a regular 1024-bit RSA modulus $N = pq$ would take $3.1 \cdot 10^6$ times the work of factoring RSA-155 (assuming no memory-space constraints) which is a much larger safety margin than we have for 1024-bit $N = p^2q$.

2. Factoring 1024-bit $N_2 = p^3q$: Here the prime factor q is of order $1024/4 = 256$-bits. We use Equation 1 to compare the running time for factoring such N_2 with the result of Miyamoto. The direct ratio of running times is:

$$\frac{L_{1/2,\sqrt{2}}(2^{256})}{L_{1/2,\sqrt{2}}(2^{183})} = 2486$$

As before we adjust our estimate by a factor of 7.57 to take into account that arithmetic modulo a 1024 bit number takes longer than arithmetic modulo a 372-bit number. Hence, factoring a 1024-bit $N_2 = p^3q$ should take approximately 670 years on a single 800MhZ Pentium III.

This comes out to be approximately 7.2 times the effort it took to factor RSA-155 (a 512 bit integer) using NFS. These numbers are only an estimate based on the asymptotic formulas and could be off by as much as a factor of 10. So, suppose that finding a 256-bit factor using ECM is 72 times the effort of factoring RSA-155 using NFS. This safety margin is far smaller than the margin provided by comparable standards. In fact, these calculations show that factoring 1024-bit $N_2 = p^3q$ is within reach of a large factoring project today. Hence, 1024-bit numbers of the form $N_2 = p^3q$ should not be used in deployed systems.

3. Factoring 1280-bit $N_3 = p^2q$: Here the prime factor q is of order $1280/3 = 426$-bits. We use Equation 1 to compare the running for factoring such N_3 with the result of Miyamoto. The direct ratio of running times is:

$$\frac{L_{1/2,\sqrt{2}}(2^{426})}{L_{1/2,\sqrt{2}}(2^{183})} = 8.8 \cdot 10^9$$

As before we adjust our estimate by a factor of $(1280/372)^2 = 12$ to take into account that arithmetic modulo a 1280-bit number takes longer than arithmetic modulo a 372-bit number. To be conservative we also include the factor of 50 improvement in ECM running time due to Okamoto-Peralta. Hence, factoring a 1280-bit $N_3 = p^2q$ should take approximately $7.6 \cdot 10^7$ years on a single 800MhZ Pentium III using ECM. This is approximately $8 \cdot 10^5$ times the work of factoring RSA-512 which provides comparable security to that provided by standard 1024-bit RSA $N = pq$.

Remark: We note that our estimates in this section are based on asymptotic behavior. We neglected some low order terms, but tried to be conservative where appropriate. The effect of the Okamoto-Peralta improvement is a bit difficult to predict. We used a factor of 50 to estimate the improvement for 1024-bit $N = p^2q$ since that seems to be a conservative estimate.

2 The Lattice Factoring Method

The special structure of $N = p^2q$ makes it susceptible to another factoring method called the Lattice Factoring Method (LFM) [1]. The LFM is designed to factor very large integers of the form $N = p^dq$ for large value of d, e.g. $d \approx 20$. However, the method has some implications to integers of the form $N = p^2q$ and $N = p^3q$. We state the following theorem that easily follows from Lemma 3.3 of [1]:

Theorem 2.1 *Let $N = p^dq$ be an n-bit integer where $\lfloor \log_2 p \rfloor = \lfloor \log_2 q \rfloor = n/(d+1)$. Then the LFM factoring algorithm will factor N in time $O(2^{n/(d+1)^2} n^4)$.*

For square integers $N = p^2q$ we get a factoring algorithm whose running time is dominated by the term $2^{n/9}$. For $N = p^3q$ we get an algorithm whose running time is dominated by the term $2^{n/16}$. These dominant terms determine the number of iterations needed for the algorithm to work. In each iteration the algorithm runs the LLL lattice bases reduction algorithm.

Therefore, the running time for each iteration is non-trivial. We use the experiments given in [1] and assume that each iteration takes one hour on a 400MhZ Pentium II. As before we analyze the running time of the LFM algorithm for (1) 1024-bit $N = p^2q$, (2) 1024-bit $N = p^3q$, and (3) 1280-bit $N = p^2q$.

1024-bit $N = p^2q$: the expected number of iterations is $2^{1024/9} = 2^{113}$. The number of iterations alone is already too large to carry out. Considering that each iteration takes one hour makes this algorithm completely infeasible.

1024-bit $N = p^3q$: the expected number of iterations is 2^{64}. Taking one hour per iteration the algorithm will take 10^{16} years on a single machine. This number is beyond the reach of computers for the next 20 years. Hence, the LFM method is ineffective against 1024-bit $N = p^3q$.

1280-bit $N = p^2q$: Here again the number of iterations is too large to carry out. The algorithm will require an expected 2^{142} iterations. Again this is beyond the reach of computers for the next 20 years.

In summary, the LFM method works well for very large numbers of the form $N = p^dq$ with large d, but is ineffective against 1024-bit integers (or longer) of the form $N = p^2q$ and $N = p^3q$.

3 Factoring Using Jacobi Symbols

We conclude this report with an observation that suggests that integers of the form $N = p^2q$ might be easier to factor than factoring $N = pq$. In other words, we show that $N = p^2q$ might be susceptible to a class of integer factorization algorithms that do not apply to $N = pq$.

We begin with the observation of Okamoto-Peralta [11]. Let $N = p^2q$ and $x \in \mathbb{Z}_N^*$. Okamoto and Peralta showed that we can easily determine whether x is a quadratic residue modulo q without knowing q. This follows from the fact that $\left(\frac{x}{q}\right) = \left(\frac{x}{N}\right)$. Hence, x is a quadratic residue in $\mathbb{F}_q$ if and only if the Jacobi symbol $\left(\frac{x}{N}\right)$ is 1. Furthermore, the Jacobi symbol $\left(\frac{x}{N}\right)$ is easy to compute.

Using our ability to determine whether x is a quadratic residue modulo q can potentially give rise to a new class of factoring algorithms for $N = p^2q$. Let $\ell_1, \ell_2, \ldots, \ell_k$ be the k smallest odd integer primes $(3, 5, \ldots)$. If ℓ_i is a quadratic residue modulo q then q satisfies certain congruence relations modulo $4\ell_i$. For example, if 3 is a quadratic residue modulo q then $q \bmod 12 \in \{1, -1\}$. If 5 is a quadratic residue modulo q then $q \bmod 20 \in \{1, 9, 11, 19\}$. Generally, every time we learn the Legendre symbol of $\ell \bmod q$ we learn that $q \bmod 4\ell$ is one of $\ell - 1$ possible values. By using many small primes in this way we collect more and more congruence information about q. The question is whether it is possible to recover q from all this information. One can show that all this congruence information uniquely determines q, but it is currently an open problem to find q given this information.

We can collect this information about q into a simple univariate modular polynomial as follows. The one bit of information we learn from the Legendre symbol of $\ell \bmod q$ can be

converted via quadratic reciprocity into an equation in q of the form

$$q^{(\ell-1)/2} = (-1)^b \pmod{\ell}$$

where $b \in \{0, 1\}$ is known to us. We get one such equation for each small ℓ_i. This leads to the following system of equations in q:

$$\begin{cases} q^{(\ell_1-1)/2} &= (-1)^{b_1} \pmod{\ell_1} \\ & \vdots \\ q^{(\ell_k-1)/2} &= (-1)^{b_k} \pmod{\ell_k} \end{cases}$$

Let L be the product of all the ℓ_i's. Then we can apply the Chinese Remainder Theorem (CRT) to all these equations to get a polynomial $f(x) \in \mathbb{Z}_L[x]$ of degree $(\ell_k - 1)/2$ for which we know that $f(q) = 0 \bmod L$. When L is sufficiently large we know that q is a small root of this polynomial ($q < \sqrt[3]{N}$). Since the polynomial $f(x)$ is easy to construct the question is whether it is possible to recover all integers $|y| < \sqrt[3]{N}$ such that $f(y) = 0 \bmod L$. One of these roots of $f(x)$ will be the desired q. Currently, the best techniques for finding small roots of modular equations (due to Coppersmith) do not seem capable of solving this problem.

This approach to factoring $N = p^2q$ does not apply to standard RSA moduli $N = pq$. We are clearly using the special structure of p^2q to deduce information about q. Currently we do not know how to make use of this information. At any rate, this approach shows that factoring $N = p^2q$ is potentially easier than factoring $N = pq$. The special structure of $N = p^2q$ opens the door to new techniques that do not apply to $N = pq$.

Remark: As a final comment we note that the discussion in this section generalizes to $N = p^dq$ for $d > 2$ by using the d'th power residue symbol (rather than the Legendre symbol). This symbol can be computed using the Eisenstein reciprocity theorem [4, p. 207]. We also note that an efficient factoring algorithm for $N = p^2q$ using the techniques of this section would imply that computing quadratic residousity modulo a Blum integer is as hard as factoring the integer — a long standing open problem.

4 Conclusions: Recommended parameter sizes for $N = p^dq$

This report studies the effectiveness of various factoring techniques for factoring integers of the form $N = p^2q$ and more generally integers of the form $N = p^dq$. We studied the effectiveness of three factoring techniques:

1. The Elliptic Curve Method [7] augmented by [11],
2. The Lattice Factoring Method [1], and
3. Factoring $N = p^2q$ using Jacobi symbols.

Our conclusion is that the LFM method and the Jacobi symbol method cannot currently be used to attack $N = p^2q$ when N is 1024-bits long. On the other hand, the ECM method with the improvements of Okamoto and Peralta poses a threat that should be addressed. When using $N = p^dq$ one would like ensure that the modulus size is sufficiently large to provide the same level of security as 1024-bit standard RSA. Our conclusions are three fold:

1. 1024-bit integers of the form $N = p^2q$ provide adequate security in the short term. However, within 5 to 10 years the safety margins provided by such integers is likely to become unacceptable.

2. To provide security comparable to standard 1024-bit RSA it is recommended to use at least a 1280-bit $N = p^2q$ modulus. We estimate that the running time for ECM on such a modulus is comparable to the running time of the Number Field Sieve on 1024-bit RSA (note that we are ignoring memory-space constraints).

3. 1024-bit integers of the form $N = p^3q$ should be considered insecure since today they are susceptible to attack using the ECM method. The ECM can be adapted to $N = p^3q$ using a generalized Okamoto-Peralta improvement (generalized to make use of the cubic symbol rather than the Legendre symbol).

We also pointed out in section 3 that integers of the form $N = p^2q$ have more structure than standard RSA moduli. We showed that this may lead to new factoring algorithms specifically targeted at $N = p^2q$. Hence, factoring $N = p^2q$ might be easier than factoring $N = pq$ of the same size. However, we cannot currently exploit the extra structure of $N = p^2q$ to design faster factoring algorithms.

References

[1] D. Boneh, G. Durfee, and N. Howgrave-Graham. "Factoring $N = p^rq$ for Large r." *Proceedings of Crypto '99*, vol. 1666 of *LNCS*, pp. 326–337. Springer-Verlag, 1999.

[2] S. Cavalar et al. "Factorization of a 512-bit RSA Modulus", Proc. of Eurocrypt '2000.

[3] A. Fujioke, T. Okamoto, and S. Miyaguchi. "ESIGN: an efficient digital signature implementation for smartcards", In. proc. Eurocrypt '91, pp. 446–457, 1991.

[4] K. Ireland and M. Rosen "A classical introduction to modern number theory", 2nd edition, GTM 84, Springer, 1990.

[5] A. Lenstra and H.W. Lenstra Jr.. "Algorithms in Number Theory", in Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity), ch. 12, pp. 673–715, 1990.

[6] A. Lenstra and H.W. Lenstra Jr.. "The development of the number field sieve", Lecture Notes in Mathematics, Vol. 1554, Springer-Verlag, 1994.

[7] H.W. Lenstra Jr.. "Factoring integers with elliptic curves", Annuals of Mathematics, 126:649-673, 1987.

[8] P. Montgomery. "Speeding up the Pollard and Elliptic Curve methods of factorization", Math. Comp. Vol. 48 (1987), pp. 243–264.

[9] P. Montgomery and R. Silverman. "An FFT extension to the $p-1$ factoring algorithm", Math. Comp. Vol. 54 (1990), pp. 839–854.

[10] T.Okamoto. "A Fast Signature Scheme Based on Congruential Polynomial Operations", IEEE Trans. on Inform. Theory, IT-36,1 (1990) 47–5

[11] E.Okamoto and R.Peralta. "Faster Factoring of Integers of a Special Form", IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences, E79-A, n.4 (1996).

[12] T.Okamoto and A.Shiraishi. "A Fast Signature Scheme Based on Quadratic Inequalities", Proc. of the ACM Symp. Security and Privacy, ACM Press (1985)

[13] T. Okamoto and S. Uchiyama. "A new public key cryptosystem as secure as factoring", in Proc. Eurocrypt '98, pp. 310–318, 1998.

[14] R. Silverman and Wagstaff Jr.. "A Practical analysis of the elliptic curve factoring algorithm", Math. Comp. Vol 61, 1993.

[15] I. Miyamoto. Report on ECM-net, Oct. 2001.
`http://www.loria.fr/ zimmerma/records/ecmnet.html`

[16] T. Takagi. "Fast RSA-type cryptosystem modulo $p^k q$", in Proc. Crypto '98, pp. 318–326, 1998.