Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 6 (1): 458-481, 2006
ISSN 1812-5654

Review of Methods for Integer Factorization Applied to Cryptography

Kefa Rabah
Department of Physics, Eastern Mediterranean University,
Gazimagusa, North Cyprus, via Mersin 10, Turkey

Abstract: The problem of finding the prime factors of large composite numbers has always been of mathematical interest for centuries. With the advent of public key cryptosystems it is also of practical importance, because of the security of these cryptosystems, such as the Rivest-Shamir-Adleman (RSA) systems, depends on the difficulty of factoring the public-keys. In recent years the best known integer factorization algorithms have improved greatly, to the point where it is now easy to factor a 100-decimal digit number and possible to factor larger than 250 decimal digits, given the availability of enough computing power. However, the problem of integer factorization still appears difficult, both in a practical sense (for numbers of more than over 100 decimal digits), in a theoretical sense (because none of the algorithms run in polynomial time). In this study we will outline some useful and recent integer factorization algorithms, including the Elliptic Curve Algorithm (ECM), Quadratic Sieve (QS), Number Field Sieve (NFS) and finally give some example of their usage.

Key words: Integer factorization, public-key cryptography, DLP, RSA, pollard rho, elliptic curve method, quardratic sieve, MPQS, index calculus, number field sieve

INTRODUCTION

Cryptography is an important building block of e-commerce systems. In particular, public key cryptography can be used for ensuring the confidentiality, authenticity and integrity of information in an organization. To protect the sensitive information in an organization, encryption can be applied to conceal sensitive data so that the encrypted data is completely meaningless except to the authorized individuals with the correct decryption key. To preserve the authenticity and integrity of data, digital signature can be performed on the data such that other people can neither impersonate the correct signer nor modify the signed data without being detected.

Modern asymmetric key cryptography uses mathematical operations that are fairly easy to do in one direction, but extremely hard to do in reverse. The standard example used (indeed, the one that is almost synonymous with public key encryption) is that of prime factorization. Large primes have at least one practical application-they can be used to construct public key cryptosystems (also known as asymmetric cryptosystems and open encryption key cryptosystems)[1]. Two basic types of public-key schemes emerged in the mid 1970s; Diffie-Hellman (DH) for key agreement protocol proposed in 1975 which relies on the hardness of the Discrete Logarithm Problem (DLP): given p, g and g^a, find a[2,3]. Two years later Rivest, Shamir and Alderman at MIT in the USA proposed the key transport and digital signature schemes known by their initials as RSA[4], which takes it security from the hardness of the Integer Factorization Problem (IFP): Given two large prime numbers p and q, it is a straightforward task to multiply them together and find the resulting multiplicand, $n = (p \cdot q)$. However, given a large composite integer that is a product of two large prime factors, it is extremely difficult to find those two primes[5].

Factorization was once primarily of academic interest. It gained in practical importance after the introduction of the RSA public-key cryptosystem[1,4,6]. It is one of the most popular public key crypto-algorithm, which is widely used today in hardware and software to secure electronic data transport on Internet especially the e-commerce to secure sensitive information such as credit card numbers.

As usual let us consider our usual communicating partners, Alice and Bob. Suppose Alice wants to use RSA to send an encrypted message to Bob. Let the public key of Bob be (e,n) and the private key of Bob be (d,n) where n is the product of two prime numbers p and q (with $ed = 1(\mathrm{mod}(p-1)(q-1))$. In this scenario, (e, n) is accessible to anyone (e.g. Alice) who wants to send encrypted messages to Bob while d is kept secretly by Bob, for detailed implementation procedure[1,4,5].

To encrypt a plaintext message M for Bob, Alice has to compute ciphertext: $C = M^e(\mathrm{mod}\ n)$. Bob can decrypt C by computing: $(C)^d = (M^e)^d = M(\mathrm{mod}\ n) = M$. No one except Bob can decrypt C since d is only known to Bob. To calculate d from e, it is required to factor n to get p and q. With q and q known, it is possible to calculate $(p-1)(q-1)$. By reversing the key generation procedure,

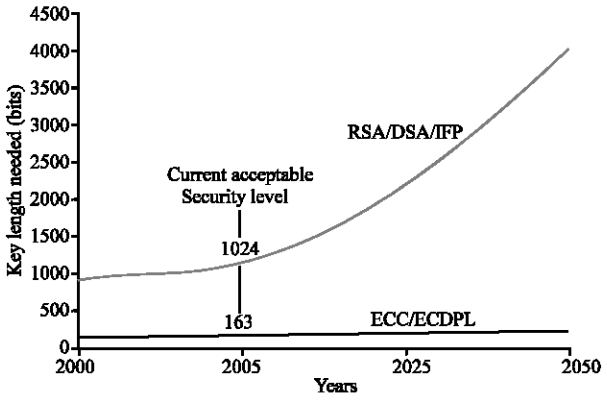


Fig. 1: Proposed the minimum key sizes (in bits) to be regarded as safe for RSA and ECC

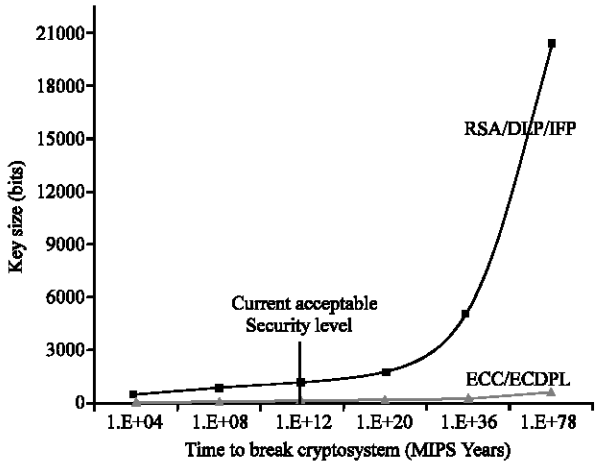


Fig. 2: Comparison of security levels of ECC and RSA/DSA

d can be calculated by computing: $e^{-1}(\text{mod}(p-1)(q-1))$[1,4] for detailed implemenation.

The product, n, is the modulus, e is the public exponent and d is the secret exponent. You can publish your public-key freely, because there are no known easy methods of calculating d, p or q given only your public-key (e,n). Authentication on the other hand, is not as easy to guarantee in public-key cryptography[7]. Since everybody knows everybody else's public-key, Eve can easily send message to Alice claiming to be Bob. The RSA crypto-algorithm can be broken by factoring n into p and q. If n is factored then (p–1)(q–1) can be found and from this d can be computed. Hence, any adversary that factors n can find the private-key d and with it decrypt any encrypted message.

Therefore, RSA crypto-algorithm is secure only if the factorization of the carefully chosen sufficiently large two prime numbers requires a super-polynomial amount of time with respect to the size of the modulus number, n. To date it has not been proved that the process of factorization of numbers requires an exponential amount of time. However, no classical polynomial time algorithm has been found and most researchers generally believe that none will ever been found! This is a practical motivation for the current interest in integer factorization algorithms. Currently, you need a crypto-keylength of 1024-bit number to get the same security you got from a 512-bit number in the early 1980s. If you want your keys to remain secure for 20 years, 1024 bits is probably too short (Fig. 1).

Because the security of RSA is so dependent on an adversary's inability to factor a large composite number, much more research has been done to find ways to quickly factor such numbers. The key question is: how large is sufficiently large to make this recovery virtually impossible? In the 1980s it was generally held that prime numbers of a fifty odd digits (i.e., 10^{50}) would suffice. As a case in point take, for example, when Rivest challenged the world in 1977 to factor RSA-129, a 129-digit number (from a special list), he estimated that on the basis of contemporary computational methods and computer systems of the day, this would take about 10^{16} years of the computing time. Seventeen years later it took only eight months in a worldwide cooperative effort to do the job[8]. This gave a false credence to a modulus number of 512-bit (155-digits) which in the mid 80s was a popular RSA encryption keylength used, for example, on the Internet and to secure transactions in the financial world: it too, was factored at CWI on August 22, 1999 using the Number Field Sieve factoring method (NFS)[9]. This was not only a new record at the time for factoring general numbers but in general, a big setback to the RSA crypto-community which relied heavily on 512-bit RSA crypto-keys so this factorization represented a breakthrough in research on RSA-based cryptosystems.

Computing power is measured in MIPS-years: a million-instructions-per-second computer running for one year or about 3×10^{13} instructions. A 100-MHz Pentium III is about a 50-MIPS machine; a 1600-node Intel Paragon is about 50,000 MIPS. In 1983, a Cray X-MP supercomputer factored a 71-digit number in 0.1 MIPS-years, using 9.5 CPU hours. That's expensive. Factoring the 129-digit number in 1994 required 5000 MIPS-years and used the idle time on 1600 computers around the world over an eight-month period. Although it took longer, it was essentially free. These two computations used what's called the quadratic sieve, but a newer, more powerful algorithm has arrived. The general number field sieve is faster than the quadratic sieve for numbers well below 116 digits and can factor a 512-bit number over 10 times faster-it would take less than a year to run on an 1800-node Intel Paragon. Figure 2 shows current acceptable security level involving MIPS-years estimation featuring public keys.

INTEGER FACTORIZATION - A HISTORICAL PERSPECTIVE

It has been known since before the days of Euclid that any natural number >2 can be represented uniquely as a product of primes, that is:

$$n = p_1 p_2 \ldots p_n \quad (1)$$

It is trivial to get the n from the primes, but how do we get the primes from n? This is the connerstone of the study.

In 1970, it was barely possible to factor a 20-digit number. In 1980, asymmetric cryptography had matured and was beginning to see widespread use in real-world applications. Factoring large integers suddenly became important work. The best algorithm of the time was Morrison-Brillhart continued fraction algorithm, based largely on Maurice Kraitchik's work during the 1920's improving Fermat's difference-of-squares method[10]. Their method was commonly used to factor 70-digit numbers, but no reports of any factorizations near 100 digits were made. After analyzing the complexity of the continued fraction algorithms, Richard Schroeppel discovered what was necessary to improve their efficiency and he began working on the linear sieve[11]. Carl Pomerance used some of the same ideas in devising the quadratic sieve, which still is the most efficient general factoring method for large numbers[12].

By 1990, with the use of the quadratic sieve factoring algorithm, the record factored number was 116 digits long[13]. The biggest break for the quadratic sieve and perhaps factoring in general, was the introduction of a multiple polynomial variant, first by Jim Davis and then Peter Montgomery[14]. This allowed for straightforward parallelization, followed by a distributed version from Robert Silverman. Arjen Lenstra and Mark Manasse brought the problem to the Internet and in 1994 the 129-digit RSA challenge number was factored using the idle time on over 1600 computers. It had been estimated in 1976, to be safe for 40 quadrillion years[15]. The quadratic sieve was replaced by Pollard's number field sieve in 1996[16]. Number Field Sieve (NFS) is currently at the cutting-edge of research into integer factoring algorithm capable of factoring large composite numbers over 100 digits[17]. The current record in factoring a generally hard integer is that of the 200 decimal digits challenge integer from RSA data Security, Inc., RSA-200, which was accomplished with General Number Field Sieve (GNFS) was factored by Bahr, *et al.*[18]. Among the Cunningham integers, the record is the factorization of 248 decimal digit integer by Special Number Field Sieve (SNFS) was factored by Aoki, Kida, Shimoyama, Sonoda and Ueda (CRYPTREC)[19,20].

Therefore, the baseline and trade-off, is the size of n should be chosen such that the time and cost for performing the factorization exceeds the value of the secured/encrypted information. But even then, great care must still be taken in the overall crypto-design, as current development in integer factorization have gone much faster than foreseen and it is a precarious matter for crypto-designers to venture upon quantitative forecasts in this field. Moreover, one should realize that it always remains possible that a new computational method could be invented from unsuspecting quarter, which makes factoring easy (e.g., quantum computing, if an operative quantum computer were to be realized in the not-so distance future): fortunately or unfortunately depending on which side you are on-no one knows how to build one yet[21,22]!

However, be warned that factoring large numbers is hard but not as hard as it used to be[21,22]. This has grave implications for the effectiveness of public-key cryptography, which relies on the difficulty of factoring long keys for its security. Today, the wise crypto-designer is ultraconservative when choosing key lengths for a public-key system. He/she must consider the intended security, the key's expected lifetime and the current state of the factoring art.

This quick historical perspectives show that the ability to factor huge numbers was not solely the result of advancements in computer technology, but instead was heavily based on the growth of mathematical algorithms. These advances began with a mathematician named Carl Friedrich Gauss[23].

THE INTEGER FACTORIZATION TECHNIQUES

For a long time factoring belonged to the realm of pure mathematics, without any practical application. Modern algorithms for factoring N fall into two major categories: these include trial division, Pollard Rho, p±1 and the Elliptic Curve Method (ECM). Algorithms in the second category factor a number regardless of the sizes of its prime factors, but cost much more when applied to larger integers. These algorithms include continued fraction, Quadratic Sieve (QS) and Number Field Sieve (NFS). In practice, algorithms in both categories are important. Given a large integer with no clue about the sizes of its prime factors, one typically tries algorithms in the first category until the cofactor (i.e., the quotient after dividing by known prime factors) of the original number

is sufficiently small. Then one tries an algorithm in the second category if the cofactor is not itself prime.

Base concepts

Factoring linque: The symbols $\mathbb{Q}, \mathbb{R}$ and $\mathbb{Z}$ denote the sets of rational numbers, real numbers and integers, respectively. If x and y are integers, then x | y (read: x divides y) means that y is a multiple of x. That is, x | y if and only if there exists $k \in \mathbb{Z}$ such that y = kx. The Greatest Common Divisor (GCD) of two integers x and y is denoted by gcd(x,y). The GCD is always positive unless x = y = 0. If gcd(x,y) = 1, then x and y are said to be coprime, i.e., they have no common factor except ±1.

Power-smoothness: A number N is B-smooth if it has no prime divisors larger than some bound B where B is a positive integer. A positive integer N is B-power-smooth if all prime powers dividing N are less than or equal to B. The power-smoothness of N is the largest B such that N is B-power-smooth. For example, $135 = 3^3 \cdot 5$ is 7-smooth and 5-smooth, but not 3-smooth or 2-smooth. Similarly, $123456789 = 2 \cdot 3^2 \cdot 5 \cdot 3607 \cdot 3803$ is 500-smooth and 3803-smooth, but not 3607-smooth or 5-smooth. Once B is found you can always compute m = B!. Alternatively one can simply compute: m = *lcm*(B).

Fermat's little theorem: Let n be a composite integer with prime factor p. By Fermat's little theorem, which states that:

$$a^{p-1} \equiv 1 \pmod{p} \text{ for } a \in (\mathbb{Z}/n\mathbb{Z})^* \qquad (2)$$

Where, a,p are integers that are relatively prime (i.e., gcd(a,p) = 1)

TRIAL DIVISION FACTORING METHOD

Almost all factoring programs attempt trial division by the smallest primes. Even if integer N is 100 decimal digits long, it takes only a few seconds to divide N by all primes up to 10^7. If N is composite, then at least one prime of N is at almost $\sqrt{N}$. To factor N, the trial division algorithm successively divides N by primes $2,3,5,7,\ldots,\sqrt{N}$ and check which primes divide the number to be factored. If p is the second largest prime factor of N, the trial division takes O(p) steps (or O(p/ln p)) steps if one does trial division only by the primes.

Sometimes the primes divisors of N are known to have a special form. For example, if $p \mid a^N - 1$ but $p \nmid a^k - 1$ for any k where k < N and k | N, then $p \equiv 1 \pmod{N}$. This information facilitates trial division, since it restricts the range of possible divisors. For such numbers, one might try trial division by all qualifying primes below 2^{32} or even higher. Hoever, unless N has a special form, trial division is impractical for finding prime divisor above 10^9. Furthermore, this method is not very economical, however and there are now far better methods available today.

The newest ones are the Quadratic Sieve (QS) and the Number Field Sieve (NFS), based on an old idea of Pierre de Fermat (1601-1665)[24] who observed that every composite number N can be written as the difference of two squares: $N = x^2 - y^2 = (x+y)(x-y)$. Thus, having found such numbers x and y, we also have found factors of N. The numbers are found by a sieving process, in which possible values of x and y are excluded. The process amounts to very efficiently collecting relations: pairs of numbers a and b possessing only small prime factors, such that N divides the difference: a – b. For large N, in general very many relations are required to find its prime factors (70 million for RSA-130, for example). In QS the relations consists of ordinary numbers, in NFS these are algebraic numbers (e.g., $3 + 2^2$).

Difficulty of factoring grows rapidly with the size, i.e., the number of digits, of a number we want to factor. To see this take a number N with L decimal digits ($N \approx 10^L$) and try factor it by dividing it by $2,3,\ldots,\sqrt{N}$ and checking the remainder. In the worst case scenario you may need approximately $\sqrt{N} = 10^{L/2}$ division to solve the problem-an exponential increase as a function of L. Now imagine a computer capable of performing 10^{10} per second. The computer can factor any number N, using the trial division method, in about $\sqrt{N}/10^{10}$ seconds. Take a 100-digit number N, so that $N \approx 10^{100}$. The computer will factor this number in about 10^{40} seconds, much longer than 3.8×10^{17} seconds (12 billion years)-the currently estimated age of the universe! Under this circumstances do we give up- Nop!-In comes the quantum computing number cruncher[21].

Shor[21] showed that if such machine could be built, integer factorization and discrete logs (including elliptic curve discrete logs) could be computed in polynomial time. This result has stimulated an explosion in research on quantum computers[22]. The one comforting factor is that all experts agree that even if quantum computers are eventually built, it will take many years to do so (at least for machines on a scale that will threaten modern public key cryptosystems) and so there will be advance warning about the need to develop and deploy alternative crypto-algorithms.

KRAITCHIK ALGORITHM

In the 1920's, Maurice Kraitchik improved Fermat's difference of squares technique for factorization ($x^2 - y^2$), which set the basis of most modern factorization algorithms[25]. Kraitchik determined that instead of looking

for a difference of squares equal to n, it would be enough to find a difference of squares, which is equal to a multiple of n. This can be restated as $x^2 \equiv y^2 \pmod n$. This congruence can have two different types of solutions: interesting and uninteresting solutions, respectively. In the case of an interesting solutions we have $x \neq \pm y \pmod n$; while uninteresting solution leads to $x \equiv \pm y \pmod n$.

What makes an interesting solution interesting and an uninteresting solution uninteresting? This can be understood by breaking down, $x^2 - y^2$ into two factors $(x + y)(x - y)$. In the case of an uninteresting solution, one of the two factors is a multiple of the number n we are trying to factor. This can be understood as follows: if $(a \cdot n)(b) = ab \cdot n$, where a and b are integers and $ab \cdot n$ is the value of $x^2 \equiv y^2 \pmod n$ and in this case the number n has not been broken into two factors. However, in the case of an interesting solution, the factorization ends up as $a \cdot b = c \cdot n$, or in other words, broken into two factors, say p and q, such that: $n = (p \cdot q)$.

Now, more importantly, how can the factors of n be pulled out once an interesting solution has been found? The answer to this can be found in Euclid's Greatest Common Divisor (GCD) algorithm. Applying this algorithm to either $x - y$ or $x + y$, an interesting solution will yield a factor of n. Euclid's algorithm works by repeatedly dividing the remainder into divisor of the previous function until the remainder is zero or one. If the remainder is zero, then the greatest common divisor has been found and; if the remainder is one, then the numbers are said to be relatively prime. Mathematically, we pull out the commpn factors using GCD as follows: $\gcd(x + y, n) = p$ or $\gcd(x - y, n) = q$ to achieve our desired aim.

The mechanics of Kraitchik algorithm: Now, to better understand this theory, an example will be presented using an overly simple composite number: $n = 1513$. The floor of the square root of this value is $\lfloor \sqrt{n} \rfloor = \lfloor \sqrt{1513} \rfloor = 39$. Using Fermat's method $(x^2 - n = y^2)$, the following is obtained:

j	x_j^2	$x_j^2 - n = y_j^2$
1	39^2	2^3
2	40^2	$3 \cdot 29$
3	41^2	$2^3 \cdot 3 \cdot 7$
4	42^2	251
5	43^2	$2^4 \cdot 3 \cdot 7$
6	44^2	$3^2 \cdot 47$
7	45^2	2^9
8	46^2	$3^2 \cdot 67$
9	47^2	$2^3 \cdot 3 \cdot 29$
10	48^2	$7 \cdot 113$
11	49^2	$2^3 \cdot 3 \cdot 37$
12	50^2	$3 \cdot 7 \cdot 47$
13	51^2	$2^6 \cdot 17$
14	52^2	$3 \cdot 397$
15	53^2	$2^4 \cdot 3^4$

Next we implement the Kraitchik's method which finds a solution by combining several of the results from the above table. The first suitable combination that is found is for $(x^2 = 39^2 \cdot 41^2 \cdot 43^2)$, since the multiplication of the results is a square $(y^2 = 2^8 \cdot 3^2 \cdot 7^2)$. Therefore, the solution $x = (39 \cdot 41 \cdot 43)$ and $y = (2^4 \cdot 3 \cdot 7)$ yields $\gcd(x - y, n) = \gcd(68757 - 336, n) = 1$ which leads to uninteresting solution. The next combination that is found is when $x = 39 \cdot 40 \cdot 47$ and $y = 2^3 \cdot 3 \cdot 29$) which is also uninteresting, since $(73320 - 696) \bmod n = 0$. Next, we check the combination, $x = 39 \cdot 45$ and $y = 2^6$ which is interesting, since $\gcd(1755\text{-}64, n) = 89$. In fact: $n = 1513 = 17 \cdot 89$. (Note: If the powers of all numbers are even, then a square has been found, as is the case with $j = 15$, with $x = 53$ and $y = 2^2 \cdot 3^2 = 36$, so that $\gcd(x - y, n) = \gcd(53 - 36, n) = 17$.

Pollard's (p – 1)-method: Pollard's p – 1 algorithm is a number theoretic integer factorization algorithm, invented by Pollard[26]. It is a special-purpose algorithm, meaning that it is only suitable for integers with specific types of factors. This algorithm is based on Fermat's Little Theorem which we came across in Eq. (2). The algorithm is also based on the insight that numbers of the form $a^b - 1$ tend to be highly composite when b is itself composite. Since it is computationally simple to evaluate numbers of this form in modular arithmetic, the algorithm allows one to quickly check many potential factors with great efficiency. In particular, the method will find a factor p if b is divisible by p – 1, hence the name. When p – 1 is smooth (the product of only small integers) then this algorithm is well-suited to discovering the factor p.

Base concepts: Let n be a composite integer with prime factor p. By invoking the Fermat's Theorem above, let us further assume that p–1 is B-powersmooth for some reasonably sized B. Recall that a positive integer m is called B-smooth if all prime factors p_i of m are such that $p_i \leq B$. Here, m is called B-powersmooth if all prime powers p_i^n dividing m are such that $p_i^n \leq B$. However, for n where there is no factor p for which p – 1 has only factors, in the worse case, this algorithm is no better than trial division. Therefore, Pollard's p – 1 algorithm can be considered as special purpose algorithm to factor integers efficiently provided n can satisfy the property discussed.

Let $p_1,\ldots,p_L$ be the primes less than B and let $e_1,\ldots,e_L$ be the exponents such that $p_i^{e_i} \leq B < p_i^{e_i+1}$.
Let:

$$m = \prod_{i=1}^{L} p_i^{e_i} \tag{3}$$

As a shortcut, note that $m = \text{lcm}(1,\ldots,B)$. As a consequence of this, (p – 1) divides m and also if p^e

divides m this implies that $p^e \leq B$. Since (p–1) divides m we know that $a^m \equiv 1 \pmod p$ and because p divides n this means $\gcd(a^m - 1, N) > 1$. Therefore if $\gcd(a^m - 1, N) \neq 1$, then the gcd is a non-trivial factor of N. Note that if p – 1 is not B-powersmooth, then $a^m \not\equiv 1 \pmod p$ for at least half of all a.

POLLARD CONCEPTS

Let N = pqr, where, p and q are distinct primes and r is an integer, such that p – 1 is B-powersmooth and q – 1 is not B-powersmooth. Now, $\gcd(a^m - 1, N)$ yields a proper factor of N. Note that in the case where, q – 1 is B-powersmooth, the gcd may yield a trivial factor because q divides $a^m - 1$. Note that this is what makes the algorithm specialized.

As an example let N = 172189 = 421·409. Such that: p – 1 = 421 - 1 = $2^3 \cdot 3 \cdot 5 \cdot 7$ and q – 1 = 409 – 1 = $2^3 \cdot 3 \cdot 17$. So, an appropriate value of B would be from 7 to 16. If B was selected less than 7 the gcd would have been 1 and if B was selected higher than 16 the gcd would have been N. Of course, we do not know what value of B is appropriate in advance, so this will factor into the algorithm's running time.

To speed up calculations, we also know that when taking the gcd we can reduce one part modulo the other, so $\gcd(a^m - 1, N) = \gcd(a^m - 1 \pmod N, N)$. This can be efficiently calculated using modular exponentiation and the Euclidean algorithm.

Algorithm and running time: The basic algorithm can be written as follows:

Algorithm - Pollard p-1 method

Inputs: *n*: a composite integer
Output: a non-trivial factor of *n* or failure

1. select a smoothness bound B
2. pick *a* randomly in $(\mathbb{Z}/n\mathbb{Z})^*$ (note: we can actually fix a, random selection here is not imperative)
3. for each prime $q \leq B$
   $e \leftarrow \lfloor \log B / \log q \rfloor$
   $a \leftarrow a^{q^e} \bmod n$ (note: this is a^m)
4. $d \leftarrow \gcd(a - 1, n)$
5. if $a < d < n$ then return d
6. if d = 1 then select a higher B and go to step 2 or return failure
7. if d = n then go to step 2 or return failure

If d = 1 in step 6, this indicates that for all p – 1 that none were B-powersmooth. If d = n in step 7, this usually indicates that all factors were B-powersmooth, but in rare cases it could indicate that a had a small order modulo p. The running time of this algorithm is $O(B \log B \cdot \log^2 n)$, so it is advantageous to pick a small value of B.

Large prime variant: A variant of the basic algorithm is sometimes used. Statistically, there is often a factor p of n such that p –1 = fq such that f is B-powersmooth and $B < q \leq B'$, where, q is a prime and B′ is called a semi-smoothness bound. As a starting point, this would work into the basic algorithm at step 6 if we encountered gcd = 1 but didn't want to increase B. For all primes $B < q_1,\ldots,q_L \leq B'$, we check if $\gcd(a^{q_i m} - 1, n) \neq 1$ to obtain a non-trivial factor of n. This is quickly accomplished, because if we let $c = a^m$ and $d_1 = q_1$ and $d_i = q_i - q_{i-1}$, then we can compute: $a^{q_1 m} = c^{d_1}$, $a^{q_2 m} = c^{d_1} c^{d_2} = a^{q_1 m} c^{d_2}, \ldots$. The running time of the algorithm with this variant then becomes $O(B' \log B' \cdot \log^2 n)$.

Additional information: Because of this algorithm's effectiveness on certain types of numbers the RSA specifications require that the primes, p and q, be such that they are non-B-powersmooth for small values of B.

The mechanics of Pollard's (p – 1)-Factoring Algorithm: We now illustrate the Pollard (p – 1) algorithm as follows:

- Choose m = B! and suppose that $(p - 1) | m$ for some bound $B \in \mathbb{N}$. (We choose B so that it is sufficiently large to capture the small prime factors of p – 1.)
- Apply Fermat's Little Theorem (FLT) to get $2^m \equiv 1 \pmod p$. (This is the reason for choosing B! in step 1.)
- Compute $d = \gcd(2^m - 1, n)$ using the Euclidean algorithm.
- If $n \nmid (2^m - 1)$, then d from step 3 provides a nontrivial factorization of n

In PARI, these computations are carried out as follows[27]:

Algorithm 4A: Pollard's (p–1) method with PARI - "Pollardp-1.gp"

```
\\ Implementation of Pollard's (p–1) method
\\ to run provide: integer N and B
\\ Copy and save as: "Pollardp-1.gp"

\\ Pollard p–1 method. Usage pollard(n,t) where n is the
\\ number to be factored and t is the size of the factor base
\\ i.e., the first t primes. Adjust to taste.
\\ By Felipe Voloch." http://www.ma.utexas.edu/users/voloch/mathinfo2.html"

{
pollard(n,t,
b,p,j,d,g,lgn)=
until(b,b=random()%n);
if(n<=1,error("Invalid input"));
g=gcd(b,n);
if(g>1,print(g," is a factor of ",n),
p=2;j=1;d=1;lgn=log(n);
while(d==1&&j<=t,
b=lift(Mod(b,n)^(p^floor(lgn/log(p))));
d=gcd(b–1,n);
if(d>1&&d<n,print(d," is a factor of ",n));
j=j++;p=nextprime(p++));
if(d==1||d==n,print("Try increasing t")))
}
```

ELLIPTIC CURVE METHOD (ECM) OF FACTORIZATION

Background of elliptic curve theory: Let K be the field. Here K will be either field $\mathbb{R}$ of real numbers, the field $\mathbb{Q}$ of rational numbers, the field $\mathbb{C}$ of complex numbers, or finite field F_q of $q = p^r$ elements. An elliptic curve over K is the set of points (x, y) with x,y ∈ K which satisfy the equation:

$$y^2 = x^3 + ax + b \quad (4)$$

together with a single element denoted by O and called the point at infinity, where a, b ∈ $\mathbb{F}_q$, $4a^3 + 27b^2 \neq 0$ and gcd (q, 6) = 1. We use the symbol O for the point at infinity since it turns out to be the additive identity, or zero element, in the additive abelian group on the elliptic curve, for detail implementation of elliptic curve theory[28].

In all cases the group used when implementing the ECM is the group of points on the curve over $\mathbb{F}_q$. If represented in affine coordinates, the points have the form: (x,y), where, x and y are in $\mathbb{F}_q$ and they satisfy the equation of the curve, Eq. 4, as well as a distinguished point O (called the point at infinity) which acts as the identity for the points on the curve.

Points of finite order: The order m of a point P on an elliptic curve is the smallest positive integer such that mP = O; of course, such a finite m need not exist. It is often of interest to find points P of finite order on an elliptic curve, especially for elliptic curve defined over $\mathbb{Q}$.

Additive of points on elliptic curve (mod p): Points are added using a geometric group law which can be expressed algebraically through rational functions involving x and y. That is, two points are added, forming P + Q, or a point is doubled, forming 2P[28].

Let E denote the points on the elliptic curve given by Eq. 4. For any two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in E, define:

$$P + Q = \begin{cases} O & \text{if } x_1 = x_2 \text{ and } y_1 = -y_2 \\ Q = Q + P & \text{if } P = O \\ (x_3, y_3) & \text{otherwise} \end{cases} \quad (5)$$

Next define x_3 and y_3 as:

$$x_3 = \lambda^2 - x_1 - x_2 \text{ and } y_3 = \lambda(x_1 - x_3) - y_1 \quad (6)$$

Where:

$$\lambda = \begin{cases} y_2 - y_1 / x_2 - x_1 & \text{If } P \neq Q, \text{ Point addition} \\ 3x_1^2 + a / 2y_1 & \text{If } P = Q, \text{ Point doubling} \end{cases} \quad (7)$$

As an example, let the elliptic curve be given by: E: $y^2 = x^3 + x + 188$ over the finite field $\mathbb{F}_7$. To find all the points on E, we need to solve the congruence, $x^3 + x + 188 \equiv y^2 \pmod 7$, for all $x \leq 6$. In other words, we need to find the quadratic residues. For each value of x, one needs to determine whether or not it is a quadratic residue. If it is the case, then there are two values in the elliptic group. If not, then the point is not in the elliptic group E_7 (1, 1880). So there will be a lot of points modulo 7. These are:

$$E(\mathbb{Z}/7\mathbb{Z}) = \{O, (2,3), (1,1), (2,4), (6,5), (4,5), (3,1), (3,6), (4,2), (6,2), (1,6)\}$$

Hence, E has eleven points, which is also known as the order of the group, defined as: # E ($\mathbb{F}_7$) = 11. We now show that this is a cyclic group. We need only find a nonzero element that is a generator. Choose P = (2, 3) as the generator point that generates all the other points. For example, to find 2P = (2,3) + (2,3) = (4,2), we find λ using Eq. 7, as follows:

$$\lambda = \frac{3x_1^2 + a}{2y_1} \bmod p = \frac{3(2)^2 + 1}{2(3)} \bmod 7$$

$$= \frac{13}{6} \bmod 7 = 13(6^{-1}) \bmod 7 = 1$$

Which we enter in the table below:

k	1	2	3	4	5	6
kP	(2,3)	(4,2)	(3,1)	(6,5)	(1,1)	(1,6)
λ	-	1	3	5	4	2

k	7	8	9	10	11
kP	(6,2)	(3,6)	(4,5)	(2,4)	[O]
λ	4	5	3	1	-

Hasse's bound for elliptic curve over F_r: If E is on an elliptic curve over $\mathbb{F}_p$ for a prime p>3 and A is the number of points on E, then, $|A - p - 1| < 2\sqrt{P}$ [28]. In the above example, we observe that the number points or group order is 11. The group structure of E over $\mathbb{F}_p$ is also known.

Now that we have explained the basic theory behind elliptic curve we will next look at how this theory motivated researchers to look at the possibility of using it to perform integer factorization or ECM.

Addition and reduction of points on elliptic curves (mod N) useful for ECM: Let N be an integer and let E be an elliptic curve over $\mathbb{Q}$ with equation:

$$y^2 = x^3 + ax + b, \; a, b \in \mathbb{Z} \text{ and}$$

$$\gcd(4a^2 + 27b^2, N) = 1 \quad (8)$$

Let P_1, P_2 be points on E where $P_1 + P_2 \neq O$ and the denominators of P_1, P_2 are relative prime to N. Then $P_1 + P_2$ is on E with coordinates having denominators prime to N if and only if there does not exist a prime p|N such that:

$$\overline{P_1}(\text{mod } p) + \overline{P_2}(\text{mod } p) = \overline{O}(\text{mod } p) \quad (9)$$

On the elliptic curve $\overline{E}(\text{mod } p)$ over F_p, with equation:

$$y^2 = x^3 + \overline{a}(\text{mod } p)x + \overline{b}(\text{mod } p) \quad (10)$$

Where, $\overline{P}(\text{mod } p)$ means $(\overline{x}(\text{mod } N), (\overline{y} \text{mod } N))$ where, P (x, y) is point on E with $x = \overline{x}(\text{mod } N)$ and $y = \overline{y}(\text{mod } N)$. Here, $\overline{E}(\text{mod } p)$ denotes the curve reduced modulo N. Now that we have sorted out the basic theory behind the elliptic curve, let us move into bigger picture of the elliptic curve method of factorization, i.e., ECM.

The basic idea and motivation of ECM: Recall that in Pollard's p – 1 method we needed to fix an integer B. Further, recall that if N = p·q with p and q prime and neither p – 1 nor q – 1 is B-power-smooth number, then the Pollard (p – 1) method; is extremely unlikely to work. For example, let B = 20 and suppose that N = 59·107 = 6313. Note that neither 59 – 1 = 2·29 nor 107 – 1 = 2·53 is B-power-smooth. With m = *lcm* (1,2,..., 20) = 232792560, we have:

$$2^m - 1 = 1755 \text{ (mod N) and gcd } (2^m - 1, N) = 1,$$

So we get nothing.

As remarked earlier, the problem is that p – 1 is not 30-power-smooth for either p = 59 or p = 107. However, notice that p – 2 = 3·19 is 20-power-smooth! If we could somehow replace the group $(\mathbb{Z}/p\mathbb{Z})^*$ which has order p – 1, by a group of order p – 2 and compute a^m for an element of this group, then we might easily split N. Roughly speaking, this is what Lenstra's elliptic curve factorization method does; it replaces $(\mathbb{Z}/p\mathbb{Z})^*$ by an elliptic curve E over $\mathbb{Z}/p\mathbb{Z}$. The order of the group $E(\mathbb{Z}/p\mathbb{Z})$ is $p + 1 \pm t$ for some nonnegative integer $t < 2\sqrt{p}$ (any t can occur). For example, if E is the elliptic curve, i.e.,:

$$E: y^2 = x^3 + x + 54$$

With $\mathbb{Z}/59\mathbb{Z}$ then $E(\mathbb{Z}/59\mathbb{Z})$ is cyclic of order 57. The set of numbers $59 + 1 \pm t$ for $t \leq 15$ contain numbers with very small power-smoothness. Hence, the beauty of the elliptic curve factoring method is that it demonstrates the use of the arithmetic theory of elliptic curves. This is a branch of number theory with roots going deeply into other areas and has been well-studied in the literature[28-31].

Lenstra's Elliptic Curve Factoring Method (ECM): The Elliptic Curve Method (ECM) was invented by Lenstra[32,33]. It is suited for finding small-say 9 to 30 digits-prime factors of large numbers. Among the different factorization algorithms whose complexity mainly depends on the size of the factor searched for (trial division, Pollard rho, Pollard p – 1, Williams p + 1), it is asymptotically the best method known. ECM can be viewed as generalization of Pollard's p – 1 method, just like ECPP generalizes the N – 1 primality test. ECM relies on Hasse's theorem: if p is prime, then an elliptic curve over $\mathbb{Z}/p\mathbb{Z}$ has group order $p + 1 - t$ with $|t| \leq 2\sqrt{p}$, where, t depends on the curve. If p + 1 – t is a smooth number, then ECM will most probably succeed and reveal the unknown factor p. The running time of ECM is comparable to that of the quadratic sieve[13].

The ECM factoring technique is an extension of Pollard's (p–1) method obtained by replacing the multiplicative group by the group of points on a random elliptic curve. To find a non-trivial divisor of an integer N>1, one begins by selecting an elliptic curve E over $\mathbb{Z}/N\mathbb{Z}$ and an integer k: k = *lcm* (2, 3, ..., b). Using the addition law of the curve, one next calculates the multiple k·p of p. One now hopes that there is a prime divisor of p of N for which k·P and the neutral element O of the curve become the same modulo p; if E is given by a homogenous Weierstrass equation: $y^2z = x^3 + axz^2 + bz^2$, with O = (0:1:0), then this is equivalent to the z-coordinate of k·P being divisible by p[12]. Hence, one hopes to find a non-trivial factor of N by calculating the greatest common divisor (gcd) of this z-coordinate with N.

If the above algorithm fails with a specific elliptic curve E, there is an option that is unavailable with Pollard's (P–1) method. We may repeat the above algorithm with a different choice of E. The number of points on E over $\mathbb{Z}/p\mathbb{Z}$ is of the form $p + 1 - t$ for some t with $|t| < 2\sqrt{p}$ and the algorithm is likely to succeed if p + 1 – t is B-power-smooth.

Now suppose that we wish to factor N. Choose an integer B. Next choose a random point P = (x, y) and a random elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{Z}/N\mathbb{Z}$ that goes through P. Let k = *lcm* (2, 3, ..., b). Try to compute mP (and hence, λ) working modulo N and at some point we can not compute λ because we can not compute the inverse modulo N from Eq. 7, then we (usually) find a nontrivial factor of N. Something wrong and not being able to divide is analogous to a^m being congruent to 1 modulo p.

The mechanics of ECM algorithm: Let n be an odd composite integer. The following is the algorithm for factoring n.

- In some random fashion, we generate pair (E, P), where, E is an elliptic curve over $\mathbb{Q}$ with equation:

$$y^2 = x^3 + ax + b,\ a, b \in \mathbb{Z} \text{ and P a point on E.}$$

- Check that gcd $(n, 4a^2 + 27\ b^2) = 1$. If not, then we have a factor of n, unless gcd $(n, 4a^2 + 27b^2)$ n, in which case we choose a different pair (E, P).
- Choose $m \in \mathbb{N}$ and bounds A,B, $\in \mathbb{N}$ such that the canonical prime factorization of m is:

$$m = \prod_{j=1}^{\ell} p_j^{a_{p_j}}$$

for small primes: $p_1 < p_2 < \ldots p_l \le B$

Where: $a_{p_j} = \left\lfloor \log(A)/\log(p_j) \right\rfloor$

is the largest exponent such that: $p_j^{a_{p_j}} \le A$ (where A is the number of points on our cureve E)

- Compute: $\overline{p_1^k P} \pmod n$, for $1 \le k \le a_{p_1}$

then $\overline{p_2^k p_1^{a_{p_1}} P} \pmod n$, for $1 \le k \le a_{p_2}$

and so on, until all primes P_j dividing m have been exhausted or the following occurs.

- If the calculation of either $(x_2 - x_1)^{-1}$ or $(2y_1)^{-1}$ in Eq. 7, for some s|m in step 4, determines one of them not prime to n, then there is a prime p|n such that $\overline{sP} = O \pmod p$, above. This will give us a nontrivial factor of n unless $\overline{sP} = O \pmod p$ for all primes p|n, in which case gcd (s, n) and go back and try the algorithm with different (E, P).

The implementation of Lenstra's EC factorization method: Here we use an overly small composite number for the purpose of testing the EC method. Let n = 963. Next choose a family of curves:

$$E: y^2 = x^3 + ax + 9 \qquad (11)$$

with the points P = (0, 3) and Q = (8, 23) falling on the curve E. Choose one of the points, say Q = (8, 23). We now choose successive natural number a until the process described above is successful in factoring n. We take B = 3 and since:

Table 1: Lenstra's ECC Factoring method using (E,P) pair $(y^2 = x^3 + x + 9, (8, 23))$ to factor n = 963

s	λ	$\bar{\lambda}$	x_3, y_3
$1 = 2^0$	---	---	(8,23)
$2 = 2^1$	193/46	67	(621,315)
$4 = 2^2$	578462/315	---	---

$$\left\lfloor \sqrt{n} \right\rfloor = \left\lfloor \sqrt{963} \right\rfloor = 31 = p$$

From Hasse's theorem, we find: $A = p + 1 + 2\left\lfloor \sqrt{963} \right\rfloor = 94$.

Next calculate: $\lfloor \log(94)/\log(2) \rfloor = 6$ and $\lfloor \log(94)/\log(3) \rfloor = 4$. Thus: $m = 2^6 \cdot 3^4$.

Now using Eq. 11, we tabulate the values of λ for a = 1 (Table 1). We therefore begin with the (E,P) pair $(y^2 = x^3 + x + 9, (8,23))$. With a relative on our first try, we get a solution since the process terminates with attempt to find a reduction of modulo n of λ. Here gcd (315, n) = 9. In fact $n = 9 \cdot 107$.

Since 1985, many improvements have been proposed to ECM Lenstra's original algorithm had no second phase[32]. Brent proposes in[34] a birthday paradox second phase and further more technical refinements. In, Montgomery[35] presents different variants of phase two of ECM and Pollard p–1 and introduces a parameterization with homogeneous coordinates, which avoids inversions modulo n, with only 6 and 5 modular multiplications per addition and duplication on E, respectively. It is also possible to choose elliptic curves with a group order divisible by 12 or 16[35,36].

ECM has been used to find the factors of Cunningham numbers $a^n \pm 1$ for a = 2,3,5,6,7,10,11,12). Fermat numbers $F_n = 2^{2^n} + 1$ are very good candidates for a≥10, since they are too large for general purpose factorization methods. Brent[37] completed the factorization of F_{10} and F_{11} using ECM in 1988. The largest factor found by the Elliptic Curve method has 66 digits, found by Dodson[38] in April 2005. Brent[37] maintains a list of the ten largest factors found by ECM his extrapolation from previous data would give an ECM record of 70 digits in year 2010, 85 digits in year 2018 and 100 digits in year 2025.

From the point of view of parallel implementations, the big deal with elliptic curve method is that we can use many parallel curves for many different parametric values of a and b. This is naïve parallelism, but it works. There are various ways in which one can determine, for a given N, the right number of curves to use in order to be assured that factors that can be found with method will in fact be found with this method.

PARI Implementaion of ECM: For simplicity, we use an elliptic curve of the form: $E: y^2 = x^3 + ax + 4$ with b = 4 and

a point P = (0, 2) already on it for any given value of a. The following tiny PARI function implements the ECM. Save it as ECM.gp. It generates an error message along with a usually nontrivial factor of N exactly when the ECM succeeds.

PARI Code: ECM.gp

```
{lcmfirst(B)=
local(L,i); L=1; for(i=2,B,L=lcm(L,i));
return(L);
}
{ECM(N,m) = local(E);
   E = ellinit([0,0,0,random(N),4]*Mod(1,N));
   print("E: y^2 = x^3 + ",lift(E[4]), "x+4, P=[0,2]");
   ellpow(E,[0,2]*Mod(1,N),m);    \\ this fails if and only if we win!
}
numpoints(a,p) = return(p+1 - ellap(ellinit([0,0,0,a,1]),p));
```

For simplicity we will implement the program on a small integer N. (ECM uses the random function, so the results of your run may differ from the one below.)

Running: ECM.gp

```
\\ Result with small N
> \r C:\ECM4.gp  \\ read PARI *.gp file
>
> m=lcmfirst(20) \\ B=20
%162 = 232792560
> N=105550217      \\ no. to be factored
%163 = 105550217
> Mod(2,n)^m-1
%164 = Mod(53337269, 105550217)
> gcd(53337269, 105550217)
%165 = 1
>
> ECM(N,m)
E: y^2 = x^3 + 89267312x+4, P=[0,2]
%166 = [Mod(24337003, 105550217), Mod(97898758, 105550217)]
>
> ECM(N,m)
E: y^2 = x^3 + 41879720x+4, P=[0,2]
%167 = [Mod(88635780, 105550217), Mod(17524161, 105550217)]
>
> ---------- \\ after several trials
>
> ECM(N,m)
E: y^2 = x^3 + 49137188x+4, P=[0,2]
%181 = [Mod(90537630, 105550217), Mod(59393357, 105550217)]
> ECM(N,m)
E: y^2 = x^3 + 86857918x+4, P=[0,2]
  *** ellpow: impossible inverse modulo: Mod(35166067, 105550217).
> \\ try gcd with our impossible modulo
> gcd(35166067, 105550217)
%182 = 3251

Yap! We have a factor of N = 3251x32467
```

Elliptic curve primality test: Here we will extend Lenstra's algorithm and a method for modifying it in order to give a primality test, i.e., to tell us if a given number is prime or not. Let $m \in \mathbb{N}$ with gcd (n, 6) and let E be an elliptic curve over rational field $\mathbb{Q}$. Suppose that:

- $n+1-2\sqrt{n} \le |\bar{E}(\mathrm{mod}\, n)| \le n+1+2\sqrt{n}$
- $|\bar{E}\ (\mathrm{mod}\ n)| = 2p$, where, p>2 is prime
- If $P \neq O$ is a point on E and $\overline{pP}$ on $|\bar{E}\ (\mathrm{mod}\ N)|$ then N is prime.

Using the above theorem and picking randomly chosen points $P_1, P_2, ..., P_m$; $m \in \mathbb{N}$ on E and calculating $\overline{pP_j}$ for each j = 1, 2, ..., m. If $\overline{pP_j} = O$ for some j = 1, 2, ...,m, then n is prime. As a final illustration, we again choose an overly simplistic value of n, which we use to illustrate the primality test without excessive calculation.

As an example, let n = 1103 with our earlier elliptic curve pair: $(E, P) = (y^2 = x^3 + x + 9, (5, 310))$: One calculates: $|\bar{E}\ (\mathrm{mod}\ n)| = 1084 = 2^2 \cdot 271$ and

$$1038 < n+1-2\sqrt{n} \le |\bar{E}(\mathrm{mod}\, n)| \le n+1+2\sqrt{n} < 1170$$

and so both parts 1 and 2 of the theorem are satisfied. If n were to be prime, then we might be able to generate enough points for testing from a primitive element. Thus, we proceed as with the Lenstra's method, namely by successive doubling and reduction of P with powers of 2. Notice that:

$$271 = 2^8 + 2^3 + 2^2 + 2^1 + 2^0$$

so we will first go for 2^8 and test $\overline{271P}$, the results are shown in Table 2.

Hence, from the primality test above, we prove that n = 1103 is a primenumber as we are unable to split it. Moreover, we have been very lucky that the test worked after a few try, largely due to the small value of N chosen. However, if part 1, of the theorem fails, then we have to test for compositeness by Hasse's theorem. Part 2 of the theorem, however, is very special and does not hold for many elliptic curves. Furthermore, although we were able to calculate the value of $|\bar{E}\ (\mathrm{mod}\ n)|$ above, this cardinality gets large as N gets large, so we may not even be able to determine what is in general. Calculating it could be as difficulty as proving that N is prime. These problems were overcome in a primality test by Goldwasser and Kilian[39].

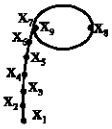
Pollard's rho method: Pollard[40] introduced the Monte Carlo method for factoring. He so named it because it rests on randomly chosen integers. Today, we refer to Pollard's method as the Pollard-rho method. Given that $n \in \mathbb{N}$ composite and p an (as yet unknown) prime divisor of it, Pollard's rho method seeks to find the two prime factors. First, we illustrate the reason behind the name Pollard rho-method. We take n = 31 as the modulus and $x_0 = 2$ as the seed and then we proceed through the Pollard rho-method, by placing the values of x_i's

Table 2: Elliptic curve primality testing method using (E, P) pair ($y^2 = x^3 + x + 9$, (5, 310)) to factor n = 1103 (tested using Sect. 7.3.3 part 5:) sP = (x,y)

s	λ	$\overline{\lambda} = \lambda (\mathrm{mod}\ n)$	(x, y)
$1 = 2^0$	-	-	(5, 310)
$2 = 2^1$	19/155	299	(48, 69)
$4 = 2^2$	6913/138	154	(457, 919)
$8 = 2^3$	313274/919	971	(1068, 317)
$16 = 2^4$	3421873/634	820	(734, 360)
$32 = 2^5$	414037/180	468	(247, 138)
$64 = 2^6$	45757/69	823	(696, 943)
$128 = 2^7$	1453249/1886	915	(826, 484)
$256 = 2^8$	2229133/968	613	(128, 537)
$264 = 2^8 + 2^3$	-11/47	563	(315, 70)
$268 = 264 + 2^2$	849/142	1008	(532, 691)
270 = 268 + 2	311/242	97	(5, 793)
271	-	-	O = (0, 0)

Table 3: Pollard rho-method simulation

i	$x_i = f(x_{i-1}) (\mathrm{mod}\ n)$	$x_j = f(x_i) (\mathrm{mod}\ n)$
1	5	26
2	26	10
3	10	14
4	14	23
5	23	8
6	8	7
7	7	21
8	21	7
9	7	21



Pollard rho-the ρ-symbol

(Table 3), which we map to achieve the rho-simple diagram shown next to the table.

From the Table 3, we can observe that when we reach x_9, then we are in the period that takes us back and forth between the residue system of 7 and that of 21 modulo 31. This is the significance of the left pointing arrow from the position of x_8 back to position of x_7, which is the same as the reside system of x_9. This completes the circuit. The diagram mapped depict the shape of the symbol of the Greek letter ρ, rho.

Modified pollard rho method: The rho algorithm is based on Floyd's cycle-finding algorithm and the birthday paradox. It is based on the observation that, by the birthday paradox, two numbers x and y are congruent modulo p with probability 0.5 after $1.77\sqrt{p}$ numbers have been randomly chosen. If p is a factor of n, the integer we are aiming to factor, then: gcd ($|x - y|$, n) = p, since $x - y \equiv 0 \pmod{p}$.

The rho algorithm therefore uses a function modulo n as a generator of a pseudo-random sequence. It runs one sequence twice as fast as the other; i.e., for every iteration made by one copy of the sequence, the other copy makes two iterations. Let x be the current state of one sequence and y be the current state of the other. The GCD of $|x - y|$ and n is taken at each step. If this GCD ever comes to n, then the algorithm terminates with failure, since this means that x = y and therefore, by Floyd's cycle-finding algorithm, the sequence has cycled and continuing any further would only be repeating previous study.

The Pollard's rho algorithm

Inputs: n, the integer to be factored; and f(x), a pseudo-random function modulo n

Output: a non-trivial factor of n, or failure.

1. x ← 2, y ← 2; d ← 1
2. While d = 1
1. x ← f(x)
2. y ← f(f(y))
3. d ← GCD(|x - y|, n)
4. If 1 < d < n, then return d.
5. If d = n, return failure.

Note that this algorithm will return failure for all prime n, but it can also fail for composite n. In that case, use a different f (x) and try again.

To illustrate the modified Pollard's rho method we follow the following procedure. Given $n \in \mathbb{N}$ composite and p an (as yet unknown) prime divisor of it, perform the following steps:

- Choose an integral polynomial f with deg (f)≥2, usually $f(x) = x^2 + 1$, is chosen for simplicity.
- Choose randomly generated $x = x_0$, the seed and compute $x_1 = f(x_0)$, $x_2 = f(x_1)$, ..., $x_{j+1} = f(x_j)$ for j = 0,1, ...,B, where the bound B is determined by step 3.
- Sieve through all differences $x_i - x_j$ modulo n until we find $x_B \neq x_j$ but $x_B \equiv x_j \pmod{p}$ for some natural number $B > j \geq 1$. Then gcd ($x_B = x_j$, n) is a nontrivial divisor of n.

As an example we use an overly small value of n = 5561 and $x_0 = 2 = y_0$ as the seed with $f(x) = x^2 + 1$ we generate the values of x_i's and y_i's as per the table below until gcd ($x_i - y_i$, n) = d > 1, where d is our factored integer.

i	$x_i = f(x_{i-1})$	$y_i = f(f(y_{i-1}))$	gcd ($\lvert x_i - y_i \rvert$, n)
1	5	26	1
2	26	2328	1
3	667	954	1
4	2328	1730	1
5	3171	5080	83

We find that gcd ($x_i - y_i$, n) for i≤ 4 until d = gcd ($|x_5 - y_5|$, n) = gcd (5080 – 3171, n) which is factor of 5561. In fact n = 5561 = 67·83.

Richard Brent's rho variant: Eldershan and Brent[41] published a faster variant of the rho algorithm. He used the same core ideas as Pollard[40] however, with a different method of cycle detection that was faster than Floyd's original algorithm. In Real practice, the algorithm is very fast for numbers with small factors. For example, on a 733 Mhz workstation, an implementation of the rho algorithm, without any optimizations, found the factor 274177 of the sixth Fermat number in about half a second. The sixth Fermat number is 18446744073709551617 (20 decimal digits). However, for a semiprime of the same size, the same workstation took around 9 seconds to find a factor of 10023859281455311421 (the product of 2 each of 10-digit primes). The rho algorithm's most remarkable success has been the factorization of the eighth Fermat number by Brent and Pollard[42]. They used Brent's variant of the algorithm, which found a previously unknown prime factor. The complete factorization of F_8 took, in total, 2 h on a Univac 1100/42.

THE QUADRATIC SIEVE FACTORING ALGORITHM

Background: Pollard's two methods discussed above may be invoked when trial division fails to be useful. However, if the methods of Pollard fail to be useful, which they will for large prime factors, say with the number of digits in the high tens, then we need more powerful number cruncher algorithm e.g., the quadratic sieving technique[13]. The Quadratic Sieve (QS) factoring algorithm was the most effective general purpose factoring algorithm of the 1980's and early 1990's. This algorithm is an improvement over the Fermat's method and Dixon's random square method for performing factorization. It is the method of choice for factoring integers between 50 and 100 digits. Unlike the Integer Factorization Problem (IFP) and the likes in which the running time depends mainly on the size of p and q (the factors of n = (p·q)), the running time of Quadratic Sieve (QS) and its advance counterpart Number Field Sieve (NSF) depends mainly on the size of n.

The basic idea behind QS algorithm is to find solutions where: $x^2 \equiv y^2 \pmod n$. This would imply $(x+y)(x-y) \equiv 0 \pmod n$. By calculating gcd((x + y), n) and gcd((x – y), n), it may be possible to find a non-trivial divisor of n. In particular, when n is the product of two prime numbers p and q, the probability of finding a non-trivial factor is 2/3.

Recall that the Kraitchik[13] method searches for combinations to produce a square by incrementing from the ceiling of the square root until an interesting combination is found. The quadratic sieve aims at finding a more efficient method of searching for these combinations by using a modified version of the Sieve of Eratosthenes. The Sieve of Eratosthenes searches for new primes by crossing off all multiples of primes below the square root of the upper bound of the search. Instead of crossing off the numbers, the quadratic sieve uses the search to find numbers, which fall in a range of primes. The quadratic sieve would search for results, which were completely factorable by all primes underneath a certain value (a combination with primes under a certain value is called B-Smooth, where y is the largest prime) by using division, instead of simply crossing off the values. However, it must be noted, that the larger the number being factored, the harder it becomes to find values under a low y-smooth range.

The mechanics of quadratic sieve algorithm: Quadratic Sieving (QS) technique is a process whereby we search numbers up to a prescribed bound and eliminate candidates along the way, leaving only the solution. Recall Fermat's method, which essentially had as an end-goal to solve $x^2 \equiv y^2 \pmod n$ in an attempt to find a nontrivial factor of n. The QS method similarly has this same end-goal (which is essentially factoring using congruences) was introduced by Pomerance[13]. To describe it, we begin as usual with a composite $n \in \mathbb{N}$ and proceed as follows:

- Choose a factor base $\mathfrak{S} = \{p_1, p_2, ..., p_k\}$, where p_j are the primes for $j = 1,2,..., k \in \mathbb{N}$. (Evidence in the literature suggests that the optimal k is one which is chosen to be approximately $\sqrt{\exp(\sqrt{\log(n)\log\log(n)})}$ where exp(x) means e^x as a convenient approximation.)
- Find all values of $x_i = j + \lfloor \sqrt{n} \rfloor$ such that x_j^2 factors modulo n over the factor base. In other words, find those $x_j^2 \equiv \prod_{j=1}^{k} p_{(i,j)}^{a(i,j)} = y_{(i,j)} \pmod n$, with $a_{(i,j)} \geq 0$ and $p_{(i,j)} \in \mathfrak{S}$ for all i,j.
- If we find t such congruences as in part (2) with $\prod_{j=1}^{i} y_j \equiv y^2 \pmod n$, then $x^2 \equiv y^2 \pmod n$ and $\gcd(x \pm y, n)$ provides a nontrivial factor of n if $x \neq \pm y \pmod n$.

By virtue of part (2) of the algorithm, it makes no sense to have any odd primes $p \in \mathfrak{S}$ which do not have any solutions $x^2 \equiv n \pmod p$ for any $x \in \mathbb{Z}$. Hence, we choose primes in $\mathfrak{S}$ to ensure the maximal possibility of finding solutions to $x_j^2 \equiv n \pmod p$. When n fails to satisfy $x^2 \equiv n \pmod p$ for any $x \in \mathbb{Z}$, n is a quadratic nonresidue modulo p and if there is a solution of the congruence, then n is a quadratic residue modulo p.

Table 4: Computation of $Q(x_j)$ and extracton of primes in the factor base

j	x_j^2	$Q(x_j)$	$Q(x_j)$	–1	2	3	7	11	17	19	23	37	41	
0	200^2	–5313	$-1\cdot3\cdot7\cdot11\cdot23$	√		√	√	√			√			
1	201^2	–4912	$-1\cdot2^4\cdot307$	√	√									X
2	202^2	–4509	$-1\cdot3^3\cdot167$	√		√								X
3	203^2	–4104	$-1\cdot2^3\cdot3^3\cdot19$	√	√	√				√				
4	204^2	–3697	$-1\cdot3697$	√										X
5	205^2	–3288	$-1\cdot2^3\cdot3\cdot137$	√	√	√								X
6	206^2	–2877	$-1\cdot3\cdot7\cdot137$	√		√	√							X
7	207^2	–2464	$-1\cdot2^5\cdot7\cdot11$	√	√		√	√						
8	208^2	–2049	$-1\cdot3\cdot683$	√		√								X
9	209^2	–1632	$-1\cdot2^5\cdot3\cdot17$	√	√	√			√					
10	210^2	–1213	$-1\cdot1213$	√										X
11	211^2	–792	$-1\cdot2^3\cdot3^2\cdot11$	√	√	√		√						
12	212^2	–369	$-1\cdot3^2\cdot41$	√		√							√	
13	213^2	56	$2^3\cdot7$		√		√							
14	214^2	483	$3\cdot7\cdot23$			√	√				√			
15	215^2	912	$2^4\cdot3\cdot19$		√	√				√				
16	216^2	1343	$17\cdot79$						√					X
17	217^2	1776	$2^4\cdot3\cdot37$		√	√						√		
18	218^2	2211	$3\cdot11\cdot67$			√		√						X
19	219^2	2648	$2^3\cdot331$		√									X
20	220^2	3087	$3^2\cdot7^3$			√	√							
21	221^2	3528	$2^3\cdot3^2\cdot7^2$		√	√	√							
22	222^2	3971	$11\cdot19^2$					√		√				
23	223^2	4416	$2^6\cdot3\cdot23$		√	√					√			
24	224^2	4863	$3\cdot1621$			√								X
				–1	2	3	7	11	17	19	23	37	41	

Note: X implies not fully factored over $\mathcal{S}$

As an example and again using an overly small value composite number; let n = 45313. Notice that the optimal number of primes in our factor base should be twelve, by part (1) of the above algorithm. To find a factor base consider the values of (n/p), which we compute using the Legendre symbol:

p	2	3	5	7	11	13	17
(n/p)	1	1	–1	1	1	–1	1
x = msqrt(p,n)	1,	1,2	Fail	3, 4	2, 9	Fail	5, 12

p	19	23	29	31	37	41
(n/p)	1	1	–1	–1	1	1
x = msqrt(p,n)	6, 13	7, 16	Fail	Fail	5, 32	7, 34

Example Maple: x:=msqrt(7,n), =±, ⇒ x = 2, or 5. PARI: kronecker(n, p)

Therefore, our selected factor base: $\mathcal{S}$ = {2,3,7,11,17,19, 23,37,41}. That is, to factor n = 45313, we start by considering only 41-smooth numbers. Next we look at the polynomial $Q(x) = x^2 - n$. In order for a prime p to divide a value of Q(x), it is necessary that $x^2 \equiv n \pmod p$, which we compute as per (Table 4) (using the values of the factor base); and at the same time extract the primes that appear in our factor base, $\mathcal{S}$.

Now we use the results from Table 4 to speed up the factorization of $Q(x) = x^2 - n$. Below we list the values of x from 200 to 225 (these values are chosen to be roughly centered from about 212, the square root of n, i.e., $\lfloor \sqrt{n} \rfloor = \lfloor \sqrt{45313} \rfloor = 212$, which gives us a sieving range, M = 25, which we use to further develop (Table 4). That's for each j find: $x_j = j + \lfloor \sqrt{n} \rfloor$, x_j^2 and finally $x^2 \equiv n \pmod p$. That is, we tick-mark which primes divides Q(x).

Since most of the entries in columns 5-14 in Table 4 (~75 % to be precise) are empty, we have saved a lot of work in implementing the sieving method of building the table to tell us which primes we need to consider (tick marked) plus QS technique requires us to use only the rows that are completely split and cross out those that are but their prime factors are outside our factor base (rows marked with X in the last column). This effectively leads to a great saving. (For larger numbers, the savings are even greater - much greater!)

Now we can start to factor $Q(x) = x^2 - n$, for each value of x, using Table 4 to tell us which primes will factor Q(x). Scanning Table 4 we can see that we have a possible solution for j = 14, j = 20 and j = 23

Hence, we have: $214^2 \equiv 3\cdot7\cdot23 \pmod n$, $220^2 \equiv 3^2\cdot7^3 \pmod n$ and $223^2 \equiv 2^6\cdot3\cdot23 \pmod n$ so $(214\cdot220\cdot223)^2 \equiv (2^3\cdot3^2\cdot7^2\cdot23)^2 \pmod n$, namely $31537^2 \equiv 35831^2 \pmod{45313}$ and gcd(31537 – 35831, 45313) = 113, or gcd(31537 + 35831, 45313) = 401. In fact: $n = 45313 = 113\cdot401$.

Sometimes we get lucky in our use of the quadratic sieve since we may find, in the sieving process, that a single $x_j^2 - n$ is itself a square. In other words, the value t = 1 in part (3) of the above algorithm. In this particular example we were not that lucky!

Now let's implememt a more effective and robust binary technique which gives a definitive solution and

Table 5: Implementation of an effective and robust binary technique for speeding up QS

j	x_j^2	Q(x_j)	Q(x_j)	–1	2	3	7	11	17	19	23	37	41	r_i
0	200^2	–5313	$-1\cdot3\cdot7\cdot11\cdot23$	1	0	1	1	1	0	0	1	0	0	1
3	203^2	–4104	$-1\cdot2^3\cdot3^3\cdot19$	1	1	1	0	0	0	1	0	0	0	2
7	207^2	–2464	$-1\cdot2^5\cdot7\cdot11$	1	1	0	1	1	0	0	0	0	0	3
9	209^2	–1632	$-1\cdot2^5\cdot3\cdot17$	1	1	1	0	0	1	0	0	0	0	4
11	211^2	–792	$-1\cdot2^3\cdot3^2\cdot11$	1	1	0	0	1	0	0	0	0	0	5
12	212^2	–369	$-1\cdot3^2\cdot41$	1	0	0	0	0	0	0	0	0	1	6
13	213^2	56	$2^3\cdot7$	0	1	0	1	0	0	0	0	0	0	7
14	214^2	483	$3\cdot7\cdot23$	0	0	1	1	0	0	0	1	0	0	8
15	215^2	912	$2^4\cdot3\cdot19$	0	0	1	0	0	0	1	0	0	0	9
17	217^2	1776	$2^4\cdot3\cdot37$	0	0	1	0	0	0	0	0	1	0	10
20	220^2	3087	$3^2\cdot7^3$	0	0	0	1	0	0	0	0	0	0	11
21	221^2	3528	$2^3\cdot3^2\cdot7^2$	0	1	0	0	0	0	0	0	0	0	12
22	222^2	3971	$11\cdot19^2$	0	0	0	0	1	0	0	0	0	0	13
23	223^2	4416	$2^6\cdot3\cdot2$	0	0	1	0	0	0	0	1	0	0	14
				–1	2	3	7	11	17	19	23	37	41	

avoids the trial lookout performed above, which can be very difficulty if our composite number is large!. To achieve this, we transfer our data for the values for which, $Q(x) = x^2 - n$, splits completely over $\mathbb{S}$ and take their exponent vector modulo 2 which we enter in Table 5, columns 5-14:

The matrix A extracted from Table 5, columns 5-14 is:

$$A := \begin{bmatrix} 1&0&1&1&1&0&0&1&0&0\\ 1&1&1&0&0&0&1&0&0&0\\ 1&1&0&1&1&0&0&0&0&0\\ 1&1&1&0&0&1&0&0&0&0\\ 1&1&0&0&1&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&1\\ 0&1&0&1&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&1&0&0\\ 0&0&1&0&0&0&1&0&0&0\\ 0&0&1&0&0&0&0&0&1&0\\ 0&0&0&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0&1&0&0 \end{bmatrix}$$

$$A^T\cdot\vec{e} = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0\\ 0&1&1&1&1&0&0&0&0&0&0&1&0&0\\ 1&1&0&1&0&0&0&1&1&1&0&0&0&1\\ 1&0&1&0&0&0&1&1&0&0&1&0&0&0\\ 1&0&1&0&1&0&0&0&0&0&0&0&1&0\\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&0&0&0&0&0\\ 1&0&0&0&0&0&0&1&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} \vec{e}_1\\ \vec{e}_2\\ \vec{e}_3\\ \vec{e}_4\\ \vec{e}_5\\ \vec{e}_6\\ \vec{e}_7\\ \vec{e}_8\\ \vec{e}_9\\ \vec{e}_{10}\\ \vec{e}_{11}\\ \vec{e}_{12}\\ \vec{e}_{13}\\ \vec{e}_{14} \end{bmatrix} = \vec{0} \quad (12)$$

Equation 12 has the form, $A^T\cdot\vec{e} = \vec{0}(\mathrm{mod}\,2)$, where $\vec{e}$ is the exponent vector and A^T is the transpose of the 14×10 exponent matrix A extracted from the right side of Table 5 but reduced modulo 2. The fourteen column vectors must be linearly dependent since most are in a space of dimension at most 10. This is equivalent to saying that there exists a nonzero $\vec{e}\in GF(2)^{14}$ such that $A^T\cdot\vec{e} = \vec{0}$.

In this example we have solved (the matrix A, is transposed to A^T, as we solve $A^T\cdot\vec{e} = \vec{0}$ and not $\vec{e}\cdot A = \vec{0}$) using Maple, which gives five set of possible solutions as:

$$\vec{e} = \begin{Bmatrix} [0,0,0,0,0,0,1,0,0,0,1,1,0,0],\\ [1,0,0,0,1,0,0,0,0,0,1,1,0,1],\\ [0,0,1,0,1,0,0,0,0,1,0,0,0],\\ [0,0,0,0,0,0,0,1,0,0,1,0,0,1],\\ [0,1,0,0,1,0,0,0,1,0,0,0,1,0] \end{Bmatrix}$$

Which we shall identify as follows:

$$\vec{e} = \{[\vec{e}_{s1}],\ [\vec{e}_{s2}],\ [\vec{e}_{s3}],\ [\vec{e}_{s4}],\ [\vec{e}_{s5}]\}$$

Here $\vec{e}_{s1}$ and $\vec{e}_{s5}$ give no factors of n = 45313 while $\vec{e}_{s2}$, $\vec{e}_{s3}$ and $\vec{e}_{s4}$ are able to factor our composite number n. For example:

$$\vec{e}_{s3} = [0,0,1,0,1,0,0,0,0,0,1,0,0,0] \equiv [r_3, r_5, r_{11}]$$

Where, we read the values of x_j's corresponding to values of r_i's as listed by rows in the last column of (Table 5), which gives:

$$x = 207\cdot211\cdot220 = 9608940 \equiv 2584 \bmod 45313$$

$$y = \sqrt{(207^2-n)\cdot(211^2-n)\cdot(220^2-n)}$$
$$= 2^4\cdot3^2\cdot7^2\cdot11 = 77616 \equiv 32303(\mathrm{mod}\,n)$$

This yields the gcds:

$$\gcd(x - y, n) = 113 \quad \text{and} \quad \gcd(x - y, n) = 401$$

Which gives a factorization n=45313=113·401. This is the same value as we had found earlier via a primitive look-up method.

Traditionally, one solves the system $A^T \cdot \vec{e} = \vec{0}$ by a Gaussian elimination. However, recently some iterative methods[15,43,44] have been found. The iterative methods are superior when the matrix is large, since they require less storage (matrices arising from integer factorization problems are very sparse as noted earlier). For these large, sparse, matrices, the iterative methods are also faster-if A is an $n \times n$ matrix, then Gaussian elimination uses $O(n^3)$ bit operations but the iterative methods take $O(n)$ applications of the matrix A, which is time $O(n^2)$ if the number of nonzero entries per column remains bounded as n grows.

The term quadratic sieve comes from the above process of sieving over all quadratic congruences $x_j^2 \equiv n \pmod p$ for $p \in \mathfrak{B}$. Unlike p–1 method, as k (the number of primes in our factor base) gets large in the QS method, then we may naturally expect to find more integers x_j that factor over $\mathfrak{B}$. However, the negative side here is the increased number of congruences that we need to list before a solution leads us naturally to its complexity, naturally $O(\exp(\sqrt{\log(n)\log\log(n)}))$.

Complexity of the quadratic sieve algorithm

- Conjectured time for optimal choice of B: $\exp((c + o(1))\cdot(\log n)^{1/2}\cdot(\log \log n)^{1/2})$ with c = 1, compared to $c = \sqrt{2}$ for Dixon's algorithm. Hence, it can factor integers that are twice longer.
- Variant-self initializing multiple polynomial quadratic sieve.
- Subexponential time, subexponential space but can practically factor integers up to ~400 bits.
- Can we decrease the $(\log n)^{1/2}$ term in the exponent?

On April 2, 1994, the factorization of RSA-129 was completed using QS. It was a 129-digit number, the product of two large primes, one of 64 digits and the other of 65[8]. The factor base for this factorization contained 524339 primes. The data collection phase took 5000 mips-years, done in distributed fashion over the Internet. The data collected totaled 2GB. The data processing phase took 45 h on Bellcore's MasPar (massively parallel) supercomputer. This was the largest published factorization by a general-purpose algorithm, until the arrival of NFS which is currently the champion factoring algorithm.

THE MULTIPLE POLYNOMIAL QUADRATIC SIEVE (MPQS)

The Quadratic Sieve (QS) is much efficient algorithm, however, for large n it suffers from limitations. To generate enough relations, the length of the sieve interval M must be large. But the value of Q(x) grows with M, making it less likely that we'll find values of x which Q(x) factor fully over the factor base $\mathfrak{B}$. In general, the number of relations returned per unit time spend sieving decreases as M increases. If we sieve the 2M values of Q(x) for $|x - \sqrt{N}| \le M$, then the largest residue is about $2M\sqrt{N}$ (assuming $M \ll \sqrt{N}$). Montgomery[45] found a way to stunt this growth as M grows. His variation, which is an improvement to QS technique, is called the Multiple Polynomial Quadratic Sieve, or MPQS.

As the name suggests, the MPQS uses several polynomials instead of just a single polynomial Q(x) as in QS and although discovered independently by several researchers, it was first suggested by Peter Montgomery. But it is also important to note that MPQS goes back to Kraitchik technique[8]; however, Pomerance[12] was the first to describe and analyze it in its modern form. Davis and Holdridge[46] and independently, Montgomery[45] proposed the use of the multiple polynomials in the quadratic sieve algorithm. The new Multiple Polynomial Quadratic Sieve (MPQS) is significantly faster than the single polynomial version but requires expensive multi-precision and modular inverse operations for each new polynomial. The algorithm spends much time calculating the new zeros for each polynomial when compared to the sieving time.

In the integer factoring world, the MPQS method is usually considered as a complementary to ECM since the computing time of MPQS depends on the size of the smallest prime factor of the number to be factorized. At the present, numbers with smallest prime divisor up to 30 decimal digits may be best factorized with the help of ECM, whereas numbers with smallest divisor more than 30 digits may be best factorized with the help of MPQS, provided that the size of the number to be factorized does not exceed 90 decimal digits. In real practice, however, we usually do not have such knowledge about the size of the prime divisors we are seeking. In best practice, it is always advisable to try ECM before MPQS, in order to eliminate the smaller prime divisors first.

The mechanics of MPQS-algorithm: Let n be the (large) number, which is known by Fermat's little theorem and

which we wish to factorize. Further recall that the Quadratic Sieve (QS) method belongs to a class of algorithm, which have common aim to find two integers X and Y, such that:

$$X^2 = Y^2 (\text{mod } n) \quad (13)$$

If gcd(X–Y, n) = d satisfies 1<d<n, then d is, a proper divisor of n. In order to find such an (X,Y)-pair, one may try to find triples (U_i, a_i, H_i), where, i = 1,2,... such that:

$$H(x_i) \equiv a_i \cdot f(x_i) \equiv U^2(x_i) (\text{mod } n) \quad (14)$$

Where, $H(x_i) = a_i \cdot f(x_i)$ is easy to factor, or at least easier than n. If sufficiently many congruences have been found these can be combined, by multiplying together a subset of them, in order to get a relation of the form of Eq. 13, i.e.:

$$\prod U^2(x_i) \equiv \prod H(x_i) (\text{mod } n) \quad (15)$$

In this study we will implement the version suggested by Montgomery. Multiple polynomials keeps the value of Q(x) smaller and therefore more likely to be smooth over ℜ. The Montgomery's refinement to MPQS sieves over shorter interval but with several different quadratic polynomials in x. Instead of just:

$$Q(x) = x^2 - n \quad (16)$$

The Montgomery option starts with a general polynomial of the form:

$$f(x) = ax^2 + 2bx + c \quad (17)$$

Where, a, b and c are chosen according to certain guidelines below. The motivation for this is that by using several polynomials, we can make the sieving interval much smaller, which makes f(x) smaller, which in turn will mean that a greater proportion of values of f(x) completely factor over the factor base.

In choosing our coefficients, let a be a square. Then choose $0 \le b < a$ so that $b^2 \equiv n (\text{mod } a)$. This can only be true if n is a square mod p for every prime p|a. So we wish to choose a with a known factorization such that (n/p) = 1 for every p|a.

In order for f(x) to generate quadratic residues, the determinant $b^2 - 4ac$ must equal n. Since the determinant will be equal to 0 or 1 (mod 4) and n will be 1 or 3 (mod 4), if n≡3 (mod 4) we must multiply it by a small constant k (called the multiplier) in order to make the product kn equal to 1(mod 4) so we factor kn instead of n.

Lastly, we choose c so that $b^2 - 4ac = n$. When we find a f(x) that factors well, notice that:

$$H(x) = a \cdot f(x) = (ax)^2 + 2abx + ac + (ax + b)^2 - n \quad (18)$$

Where:

$$f(x) = ax^2 + 2bx + c, \; H(x) = a \cdot f(x) \text{ and}$$

$$U(x) = (ax + b) \quad (19)$$

So that:

$$U^2(x_i) \equiv H(x_i) (\text{mod } n) \quad (20)$$

Also as before, if a prime p divides, a·f(x), then n is a quadratic residue modulo p, so that we do not need to change our factor base.

The number to be factored hits its minimum at x = –b/a. We want to choose a, b and c so as to minimize both: –Q(–b/a) = n/a and the extreme values at the edge of the sieved interval:

$$Q(-M-b/a) = Q(M-b/a) = aM^2 - n/a$$

Where, M is half of the sieving interval. If M is prescribed, this is accomplished by setting these values equal, i.e., at $a \approx \sqrt{2n}/M$.

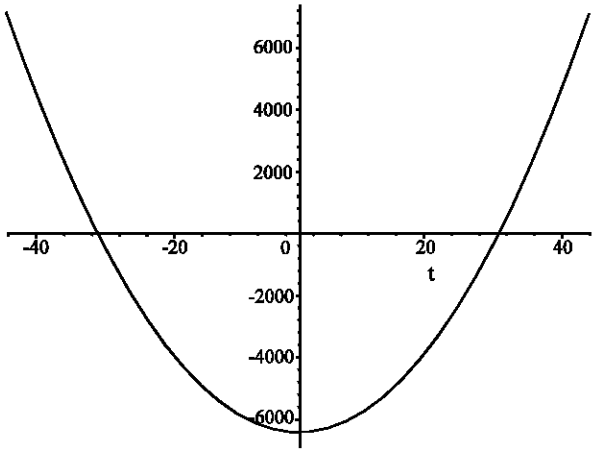
If a is chosen to be a prime, then we know how to solve the congruence, $x^2 \equiv n (\text{mod } a)$. Finally, we choose b to be a solution of this congruence and c to be: $c = (b^2 - n)/a$.

The MPQS described above has a number of nice features. For example, if we do not get enough completely factored Q(x)'s in the chosen short sieving range then we just generate a new polynomial and sieve again over our shorter interval. However, one of the nicest features is that the sieving parallelizes perfectly. With N processors, one can assign a different polynomial to each processor and the algorithm runs N times as fast. In Lenstra and Manasse[15] used this procedure with a dramatic effect to produce the first factorization of a hard 100-digit integer into a product of two primes and accomplished the factorization by farming out their polynomials to roughly 400 computers around the world.

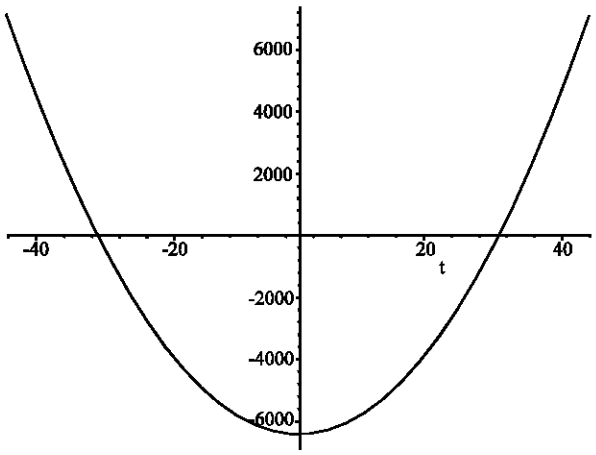
A Simple implementation of MPQS: Here we present an overly simple implementation of MPQS. Note that in real practice we are talking of integers in the range of 50-100 digits. It has been shown that if the number of decimal digits to be factorized is increased by three, then the amount of CPU-time needed is roughly doubled.

Appendix 1: \\ Maple implementation of MPQS

```
> restart;
> with(numtheory):
Warning, the protected name order has been redefined and unprotected
>
> n:=45313;
                                                    n: = 45313
> msqrt(n,7);
                                                    3
> a:=7;
                                                    a: = 7
> b:=3;
                                                    b: = 3
> c:=(b^2–n)/a;
                                                    c: = –6472
> F:=proc(r)c+r*(a*r+2*b)end;
                                                    F: = proc(r) c+r×(a×r+2×b) end proc
> evalf(–b/a);
                                                    –.4285714286
> evalf(F(–b/a));
                                                    –6473.285714
> evalf(sqrt(2*n)/a);
                                                    43.00593221
> M:=44;
                                                    M: = 44
> plot(F(t),t=–M–b/a..M–b/a);
```




```
> begining:=floor(–M–b/a);
                                                    beginning = –45
> ending:=floor(M–b/a);
                                                    ending = 43
> svals:=[seq([r,F(r)],r=begining..ending)];
svals: = [[–45, 7433], [–44, 6816], [–43, 6213], [–42, 5624], [–41, 5049], [–40, 4488],
        [–39, 3941], [–38, 3402], [–37, 2889], [–36, 2384], [–35, 1893], [–34, 1416],
        [–33, 953], [–32, 504], [31, 69], [-30, -352], [-29, -759], [-28, -1152], [-27, -1531],
        [-26, -1896], [-25, -2247], [-24, -2584], [-23, -2907], [-22, -3216], [-21, -3511],
        [-20, -3792], [-19, -4059], [-18, -5312], [-17, -4551], [-16, 4776], [-15, -4987],
        [-14, 5184], [-13, -5367], [-12, -5536], [-11, -5691], [-10, -5832], [-9, -5959],
        [-8, 6072], [-7, 6171], [-6,6256], [-5, 6327], [-1, 6384], [-3, 6427],
        [-2, 6456], [-1, 6471], [0, 6472], [1, -6459], [2, -6432], [3, -6391], [4, -6336],
        [5, 6267], [6, -6184], [7, -6087], [8, -5976], [9, -5851], [10, -5712], [11, -5559],
        [18, -4096], [19, -3831], [20, -3552], [21, -3259], [22, -2952], [23, -2631],
        [24, 2296], [25, -1947], [26, -1584], [27, -1207], [28, -816], [29, -411], [30, 8],
        [31, 441], [32, 888], [33, 1349], [34, 1824], [35, 2313], [36, 2816], [37, 3333],
        [38, 3864], [39, 4409], [40, 4968], [41, 5541], [42, 6128], [43, 6729]]
>
> plot(svals);
```




```
>
> tvals:=map(u->[u[1],ifactors(u[2])],svals);
tvals: = [[-45, [1, [[7433, 1]]]], [-44, [1 [[2, 5], [3, 1], [71, 1]]]],
        [-43, [1, [[3, 1], [19, 1], [109, 1]]]], [-42, [1[[2, 3][19, 1], [37, 1]]],
        [-41, [1, [[3, 3], [11, 1], [17, 1]]]], [-40, [1, [[2, 3], [3, 1], [11, 1], [17, 1]]]],
        [-39, [1, [[7, 1][563, 1]]]], [-38, [1,[[2, 4], [3, 1], [71, 1]]]],
        [-37, [1, [[3, 3], [107, 1]]]], [-36, [1, [[2, 4], [149, 1]]]],
        [-35, [1, [[3, 1], [631, 1]]]], [-34, [1, [[2, 3], [3, 1], [59,. 1]]]]],
        [-33, [1, [[953, 1]]]], [-32, [1, [[2, 3], [3, 2], [7, 1]]]],
        [-31, [1, [[3, 1], [23, 1]]]], [-30, [-1, [[2, 5], [11, 1]]]],
        [-29, [-1, [[3, 1], [11, 1], [23, 1]]]], [-28, [-1, [[2, 7], [3, 2]]]],
        [-27, [-1, [[1539, 1]]]], [-26, [-1, [[2, 3], [3, 1], [79, 1]]]]],
        [-25, [-1, [[3,1], [7, 1], [107, 1]]]], [-24, [-1, [[2,3], [17, 1], [19, 1]]]],
        [-23, [-1, [[3, 2], [17, 1], [19, 1]]]], [-22, [–1, [[2, 4], [3, 1], [67, 1]]]],
        [–21, [–1, [[3511, 1]]]], [–20, [-1, [[2, 4], [3, 1], [79, 1]]]],
        [-19, [-1, [[3, 2], [11, 1], [41, 1]]]], [-18, [-1, [[2, 3], [7, 2], [11, 1]]]],
        [-17, [-1, [[3, 1], [37, 1], [41, 1]]]], [-16, [-1, [[2, 3], [3,1], [199, 1]]]],
        [-15, [-1, [[4987, 1]]]], [-414, [-1, [[2, 6], [3, 4]]]], [–13, [–1, [[3, 1], [1789, 1]]]],
        [–12, [–1, [[2, 5], [173, 1]]]], [–11, [–1, [[3, 1], [7, 1], [271, 1]]]],
        [–10, [–1, [[2, 3], [2, 6]]]], [–9, [–1, [[59, 1], [101, 1]]]],
        [–8, [–1, [[2, 3], [3,1], [11, 1], [23, 1]]]], [–7, [–1, [[3, 1], [11, 2], [17, 1]]]],
        [–6, [–1, [[2, 4], [17, 1], [23, 1]]]], [–5, [–1, [[3, 2], [19, 1], [37, 1]]]],
        [–4,[–1,[[2,4],[3,1],[7,1],[19,1]]]],[–3,[–1,[[6427,1]]]],
        [–2,[–1,[[2,3],[3,1],[269,1]]]],[–1,[–1,[[3,2],719,1]]]],
        [0,[–0,[[2,3],[8,9,1]]]],[1,[–1,[[3,1],[2153,1]]]],
        [2,[–1,[[2,5],[3,1],[67,1]]]],[3,[–1,[[7,1],[11,1],[83,1]]]],
        [4,[–1,[[2,6],3,2],[11,1]]]],[5,[–1,[[3,1],[2089,1]]]],
        [6,[–1,[[2,3],[773,1]]]],[7,[–1,[[3,1],[2029,1]]]],
        [8,[–1,[[2,3],[3,2],[83,1]]]],[9,[–1,[[5851,1]]]],
        [10,[–1,[[2,4],[3,1],[7,1],[17,1]]]],[11,[–1[[3,1],17,1],109,1]]]],
        [12,[–1,[[2,4],[337,1]]]],[13,[–1,[[3,3],[193,1]]]],
        [14,[–1,[[2,3],[3,1],[11,1],[19,1]]]],[15,[–1,[[11,1],[19,1],[23,1]]]],
        [16,[–1,[[2,3],[3,1],[191,1]]]],[17,[–1,[[3,3],[7,1],[23,1]]]],
        [18,[–1,[[2,12]]]],[19,[–1,[[3,1],[1277,1]]]],
        [20,[–0,[[2,5],[3,1],[37,1]]]],[21,[–1,[[3259,1]]]],
        [22,[–1,[[2,3],[3,2],[41,1]]]],[23,[–1,[[3,1],[877,1]]]],
        [24,[–1,[[2,3],[7,1],[41,1]]]],[25,[–1,[[3,1],[11,1],[59,1]]]],
        [26,[–1,[[2,4],[3,2],[11,1]]]],[27,[–1,[[17,1],[71,1]]]],
        [28,[–1,[[2,4],[3,1],[17,1]]]],[29,[–1,[[3,1],[137,1]]]],[30,[1,[[2,3]]]],
        [31,[1,[[3,2],[7,2]]]],[32,[1,[[2,3],[3,1],[37,1]]]],
        [33,[1,[[19,1],[71,1]]]],[34,[1,[[2,5],[3,1],[19,1]]]],
        [35,[1,[[3,2],[257,1]]]],[36,[1,[[2,8],[11,1]]]],
        [37,[1,[[3,1],[11,1],[101,1]]]],[38,[1,[[2,3],[3,1],[701],[23,1]]]],
        [39,[1,[[4409,1]]]],[40,[1,[[2,3],[3,3],23,1]]]],
        [41,[1,[[3,1],[1847,1]]]],[42,[1,[[2,4],[383,1]]]],
        [43,[1,[[3,1],[2243,1]]]]
>

>pvals:=map (u–>[u[1] ,u[2] [1] ,map(v–>v[1] , u[2] [2])] ,tvals) ;
>
pvlas :=[[–45,1,[7433]],[–44,1,[2,3,71]],[–43,1,[3,19,109]],[–42,1,[2,19,37]],
        [–41,1,[3,11,17]],[–40,1,[2,3,11,17]],[–39,1,[7,563]],[–38,1,[2,3,71]],
```

```
      [–37,1,[3,107]],[–36,1,[2,149]],[–35,1,[3,631]],[–34,1,[2,3,59]],
      [–33,1,[953,]],[–32,1,[2,3,7]],[–31,1,[3,23]],[–31,–1,[2,11]],
      [–29,–1,[3,,11,23]],[–28,–1,[2,3]],[–27,–1,[1531]],[–26,–1,[2,3,79]],
      [–25,–1,[3,7,107]],[–24,–1,[2,17,19]],[–12,–1,[3,17,19]],
      [–22,–1,[2,3,67]],[–21,–1,[3511]],[–20,–1,[2,3,79]],[–19,–1,[3,11,41]],
      [–18,–1,[2,7,11]],[–17,–1,[3,37,41]],[–16,–1,[2,3,199]],[–15,–1,[4987]],
      [–14,–1,[2,3]],[–13,–1,[3,1789]],[–12,–1,[2,173]],[–11,–1,[3,7,271]],
      [–10,–1,[2,3]],[–9,–1,[59,101]],[–8,–1,[2,3,11,23]],[–7,–1,[3,11,17]],
      [–6,–1,[2,17,23]],[–5,–1,[3,19,37]],[–4,–1,[2,3,7,19]],[–3,–1,[6427]],
      [–2,–1,[2,3,269]],[–1,–1,[3,719]],[0,–1,[2,809]],[1,–1,[3,2153]],
      [2,–1,[2,3,67]],[3,–1,[7,11,83]],[4,–1,[2,3,11]],[5,–1,[3,2089]],
      [6,–1,[2,773]],[7,–1,[3,2029]],[8,–1,[2,3,83]],[9,–1,[5851]],
      [10,–1,[2,3,7,17]],[11,–1,[3,17,109]],[12,–1,[2,337]],[13,–1,[3,193]],
      [14,–1,[2,3,11,19]],[15,–1,[11,19,23]],[16,–1,[2,3,191]],
      [17,–1,[3,7,23]],[18,–1,[2]],[19,–1,[3,1277]],[20,–1,[2,3,37]],
      [21,–1,[3259]],[22,–1,[2,3,41]],[23,–1,[3,877]],[24,–1,[2,7,41]],
      [25,–1,[3,11,59]],[26,–1,[2,3,11]],[27,–1,[17,71]],[28,–1,[2,3,17]],
      [29,–1,[3,137]],[30,1,[2]]][31,1,[3,7]],[32,1,[2,3,37]],33,1,[19,71]],
      [34,1,[2,3,19]],[35,1,[3,257]],[36,1,[2,11]],[37,1,[3,11,101]],
      38,1,[2,3,7,23]],[39,1,[4409]],[40,1,[203023]],[41,1,[3,1847]],
      [42,1,[2,383]],[43,1,[3,2243]]]
> chooser:=proc(x) local
m,y,n;: =x[3] ;n :=nops (y) ;m:=op(n,y) ;evalb (m,36)end;
  chooser:=
       proc(x) local m,y,n;y:=x[3];n:= nops(y); m:=op(n,y); evalb(m<36) end proc
>
>select  (chooser , pvals) ;
[[–41,1,[3,11,17]],[–40,1,[2,3,11,17]],[–32,1,[2,3,7]],[–31,1,[3,23]],
[–30,–1,[2,11]],[–29,–1,[3,11,23]],[–28,–1,[2,3]],[–24,–1,[2,17,19]],
[–23,–1,[3,17,19]],[–18,–1,[2,7,11]],[–14,–1,[2,3]],[–10,–1,[2,3]],
[–8,–1,[2,3,11,23]],[–7,–1,[3,11,17]],[–6,–1,[2,17,23]],
[–4,–1,[2,3,7,19]],[4,–1,[2,3,11]],[10,–1,[2,3,7,17]],
[14,–1,[2,3,11,19]],[15,–1,[11,19,23]],[17,–1,[3,7,23]],[18,–1,[2]],
[26,–1,[2,3,11]],[28,–1,[2,3,17]],[30,1,[2]],[31,1,[3,7]],
[34,1,[2,3,19]],[36,1,[2,11]],[38,1,[2,3,7,23]],[4,1,[2,3,23]]]
>
>nops (tvals);
                                                    98
>tvals [1];
                                                    [–45,[1,[[7433,1]]]]
tvals [10];
                                                    [–36,[1,[[2,4],[149,1]]]]
tvals [46];
                                                    [0,[–1,[[2,3],[809,1]]]]
tvals [48];
                                                    [2,[[–1,[[2,5],[3,1],[67,1]]]]
tvals [70];
                                                    [24,[–1,[[2,3],[7,1],[41,1]]]]
tvals [81];
                                                    [35,[1,[[3,2],[257,1]]]]
tvals [84];
                                                    [38,[1,[[2,3],[3,1],[7,1],[23,1]]]]
Note: we read tvals as follows:
tvals [10]=f(–36)= 2^4*149; tvals [48] = f(2) = –1*2^5*3*67;
and tvals[84] = f(38) = 2^3*3*7*23.
```

Let n = 45313 giving $n \equiv 1 \pmod 4$ and k = 1 and let's pick a = 7 from our earlier factor base $\mathfrak{F}$, so that:

$$b = msqrt(n,a) = msqrt(45313,7) = 3$$
$$c = (b^2 - n)/a = -6472 \quad -45 \le M \le 43$$

Hence, from Eq. (19):

$$f(x) = 7x^2 = 6x - 6472 \quad H(x) = 7 \cdot f(x) \quad U(x) = 7x + 3$$

This can be combined with values of $Q(x_j)$ from QS algorithm to produce a product which allows for complete factorization of n. One such product which we can derive from is:

$$Q(200) = -1 \cdot 3 \cdot 7 \cdot 11 \cdot 23 \qquad Q(207) = -1 \cdot 2^5 \cdot 7 \cdot 11$$

and $f(38) = 2^3 \cdot 3 \cdot 7 \cdot 23$ $H(38) = 7 \cdot f(38) = 2^3 \cdot 3 \cdot 7^2 \cdot 23$
$U(38) = 269$ giving:

$$Q(200)Q(207)U(38) \equiv (200\cdot207\cdot269)^2 \equiv (2^3\cdot3\cdot7^2\cdot23)(-1\cdot3\cdot7\cdot11\cdot23)(-1\cdot2^5\cdot7\cdot11) \equiv 2^8\cdot3^2\cdot7^4\cdot11^2\cdot23^2 \equiv (2^4\cdot3\cdot7^2\cdot11\cdot23)^2$$

Which we can write as follows:

$$x \equiv 200\cdot207\cdot269 \equiv 34915 \bmod n \quad \text{and}$$

$$y \equiv 2^4\cdot3\cdot7^2\cdot11\cdot23 \equiv 5987 \bmod n$$

This yields the gcds as:

$$\gcd(x - y, n) = 113 \quad \text{and} \quad \gcd(x + y, n) = 401$$

Which give a factorization of n = 45313 = 113·401 which is the same as found with QS.

The above solutions is found using the values read from the Appendix 1. Read tvals as: M = –45 or Q(–45) to M = 43 or Q(43). These values gives the sieving range, $-45 \le M \le 43$, which can be merged with other data from Table 4 from the QS scheme to produce squares on both sides of Eq. 13.

MPQS large prime variations technique: The sieving procedure in QS looks for values of x such that f(x) is smooth with respect to the factor base $\mathcal{B}$. The algorithm is easily modified to also find values of x for which f(x) is a smooth number times a prime not much larger than the factor base bound, by adjusting the threshold used when inspecting logarithms after sieving. The extra prime in the factorization of f(x) is called a large prime. If one finds two values of x for which f(x) has the same large prime, then the corresponding congruences can be multiplied together and treated as a pair for the rest of the algorithm.

The solution is to allow separate large prime factors. The chances that another $Q(x_j)$ has the same large prime factor are good when an even number of such $Q(x_j)$ are combined, the large factor appears with even exponent and does not need to be considered in the matrix.

This procedure, called the large prime variation, is compatible with the use of multiple polynomials described above. For example, both Q(218) = 3·11·67 and f(2) = –1·2^5·3·67 have 67 as the only prime exceeding 41. After doing linear algebra phase, we decide to combine these with two entries from Table 4 from QS method. One such product which we can derive from are:

$$Q(211) = -1\cdot2^3\cdot3^2\cdot11 \quad Q(218) = 3\cdot11\cdot67 \quad Q(220) = 3^2\cdot7^3$$

and

$$f(2) = -1\cdot2^5\cdot3\cdot67 \qquad H(2) = 7\cdot f(2) = -1\cdot2^5\cdot3\cdot7\cdot67$$

U(2) = 17 is:

$$Q(211)Q(218)Q(220)U(2) \equiv (211\cdot218\cdot220\cdot17)^2 \equiv (-1\cdot2^3\cdot3^2\cdot11)(3\cdot11\cdot67)(3^2\cdot7^3)(-1\cdot2^5\cdot3\cdot67) \equiv 2^8\cdot3^6\cdot7^4\cdot11^2\cdot67^2 \equiv (2^4\cdot3^3\cdot7^2\cdot11\cdot67)2$$

Which we can write as follows:

$$x \equiv 17\cdot211\cdot218\cdot220 \equiv 24372 \bmod n \quad \text{and}$$

$$y \equiv 2^4\cdot3^3\cdot7^2\cdot11\cdot67 \equiv 13144 \bmod n$$

This yields the gcds as:

$$\gcd(x - y, n) = 401 \quad \text{and} \quad \gcd(x + y, n) = 113$$

Which give a factorization of n = 45313 = 113·401, which is the same as found with QS.

INDEX CALCULUS METHOD

The integer factorization problem has been the subject of intense research, especially in the years since the invention of RSA[4]. Recall that the most basic attack on RSA consists of factoring the modulus N = pq. Let N be an n-bit integer. Most of the subexponential-time algorithms-those that take fewer than 2^{n^c}-steps with c < 1-are of index calculus type. We now describe index calculus algorithm to factor N.

The method is based on the elementary observations that if $x^2 \equiv y^2 \pmod N$, then $N = pq \mid (x + y)(x - y)$ and so p and q each must divide either x + y or x – y. If x and y were formed independently of one another then one expects that 50% of the time the two primes will divide different factors, say $p \mid x + y$, $q \mid (x - y$. In that case and as before, we can factor N by using the Euclidean algorithm to compute gcd(x + y, N) = p.

We start the index calculus factoring algorithm by choosing a factor base $\mathcal{B}$ consisting of all primes less than some bound B along with the number –1, i.e., $\mathcal{B} = \{p_0, p_1,...,p_r\}$, where $p_0 = -1$, $p_1 = 2$, $p_3 = 3$. We next choose positive integers a < N (either randomly or according to some convenient criteria) and compute the least absolute residue of a^2. If this residue cannot be written as a product of numbers in our factor base, we choose another value of a. We finally arrive at a system of mod N relations of the form:

$$a_i^2 \equiv \prod_{j=0}^{r} p_j^{a_{i,j}}$$

Where, i = 1,2,..,s. We try the product :

$$\prod_i a_i^{2v_i} \equiv \prod_{i,j} p_j^{v_i\alpha_{i,j}}$$

Where, $v_i \in \{0,1\}$ in such a way that we get a perfect square on the right. In other words, we need each prime on the right to occur to an even power; i.e., $\Sigma_i v_i a_{i,j}$ must be even for each j = 0,1,...,r. This amounts to solving a system of r + 1 simultaneously equations in s unknowns over the field $\mathbb{F}_2 = \{0,1\}$. Once we have such a product, we can set:

$$x = \prod_i a_i^{v_i} \text{ and } y = \prod_i p_i^{\mu_i} \text{ with } \mu_j = \frac{1}{2}(\sum_i v_i \alpha_{i,j})$$

Then $x^2 \equiv y^2 \pmod N$ and there is a 50% chance that we can immediately factor N, we find another solution to the simultaneous equations over $\mathbb{F}_2$ and try again.

As an example, again using an overly simple composite number, let N = 403 = (13·31) and choose $\mathfrak{F}$ = {–1, 2, 3, 5, 7, 11, 13}. After squaring some 2-digit numbers, we find that we can take a_i, $1 \le i \le 7$ which we use to complete the table below:

i	a_i^2	a_i^2 mod n	–1	2	3	5	7	11	13
1	19^2	$-1\cdot2\cdot3\cdot7$	1	1	1	1	0	0	0
2	22^2	3^4	0	0	0	0	0	0	0
3	26^2	$-1\cdot2\cdot5\cdot13$	1	1	0	1	0	0	1
4	28^2	$-1\cdot2\cdot11$	1	1	0	0	0	1	0
5	33^2	$-1\cdot2^3\cdot3\cdot5$	1	1	1	1	0	0	0
6	34^2	2^4	0	0	0	0	0	0	0
7	38^2	$-1\cdot2^3\cdot3\cdot7$	1	1	1	0	1	0	0

As with the case with Quadratic Sieve (QS), we extract the exponent information from $\prod_i p_i^{\mu_i}$, to form a matrix A, which we transpose to get A^T,

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and then solve the equation: $A^T \cdot v = 0 \pmod 2$, where v is the exponent vector, which gives three set of possible solutions as:

$$v = \{[0,1,0,0,0,0,0], [0,0,0,0,0,1,0], [1,0,0,0,1,0,0]\}$$

Which we shall identify as follows: $v = \{[v_{s1}], [v_{s2}], [v_{s3}]\}$. Here v_{s2} and v_{s3} give no factors of N = 403, while v_{s1} is able to factor our composite number N. For example, $v_{s1} = [0,1,0,0,0,0,0]$, gives, $x \equiv 22 \pmod{403} = 22$ and $y \equiv 3^2 \pmod{403} = 9$. These yield the gcds, gcd(x – y, n) = 13 and gcd(x + y, n) = 31, which gives a factorization N = 403 = 13·31.

It can be shown that the time required to factor an n-bit integer by the above index calculus factorization method is of the order $2^{n^{1/2+\epsilon}}$ for any $\epsilon > 0$ (More precisely, the number of steps is $(O(\sqrt{n \log n}))$.) Throughout the 1980's modifications and generalizations were introduced that improved upon the performance of index calculus methods; however, no one was able to reduce the exponent of n below 1/2 + ϵ. Even when Lenstra[32] developed an exciting and conceptually very different factorization methods based on elliptic curves, asymptotically his method required roughly the same amount of time as the index calculus algorithms. Some people wondered whether the exponent 1/2 + ϵ might be best possible for a general integer factorization algorithm.

However, ideas of Pollard[16] led to a major breakthrough in factorization, called the Number Field Sieve (NFS). By carrying over index calculus to algebraic filed, it was possible to factor an arbitrary n-bit integer in time bounded by $2^{n^{1/3+\epsilon}}$ for any $\epsilon > 0$. (More precisely, $\exp(O(\sqrt[3]{n \log^2 n}))$. The number field sieve is at present the fastest method for factoring an RSA modulus; the current record is a number of 248 decimal digits. The reduction in the exponent of n from 1/2+ϵ to 1/3+ϵ has important consequences in the long run. It means that even modest improvements in hardware and software can increase the size of the numbers that can be factored. For this reason the current recommendation for implementation of RSA is to use numbers of at least n = 1024 bits (Fig. 1).

THE GENERAL NUMBER FIELD SIEVE (GNFS)

The general number field sieve (GNFS) algorithm is the fastest known method for factoring large number of between 100 and 248 digits. GNFS is an improvement on quadratic sieve[47-51]. The algorithm uses idea from diverse field of mathematics such as algebraic number theory, graph theory, finite fields and linear algebra. One of the improvements is that the polynomial being used is not only limited to quadratic, but may be cubic or even higher degree polynomials. Implementation of the number field sieve can be written such that the sieving process can take place on several computers. The results of the sieving processare then combined at a later stage.

In GNFS, as compared to QS or its variant, we have two polynomials f and g that are related modulo n, but

now the relation is more subtle: f and g both have known m modulo n:

$$f(m) \equiv g(m) \equiv 0 (\text{mod } n)$$

Also, suppose f and g are monic and irreducible. Let α be a complex root of f. Consider the ring $\mathbb{Z}[\alpha]$ (equivalent, $\mathbb{Z}[x]/(f(x))$). The members of this ring are of the form:

$$q(\alpha) = q_0 + q_1\alpha + q_2\alpha^2 + ... + q + q_d\alpha^{\deg f-1}$$

Operations in this ring are done modulo $f(\alpha)$, simply because $f(\alpha) = 0$. Similarly, define $\mathbb{Z}[\beta]$, where, β is a complex root of g.

Consider the ring homomorphism $\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}_n$ $\phi : \alpha \mapsto m$ defined by (i.e., replacing all occurrences of α with m). Like wise, $\psi : \mathbb{Z}[\beta] \to \mathbb{Z}$, $\psi : \beta \mapsto m$.

Suppose we found a setoff integer pairs $S \subset \mathbb{Z} \times \mathbb{Z}$ and also $q(\alpha) \in \mathbb{Z}[\alpha]$ and $t(\beta) \in \mathbb{Z}[\beta]$, such that:

$$q(\alpha)^2 = \prod_{(a,b)\in S}(a - b\alpha) \text{ over } \mathbb{Z}[\alpha]$$
$$t(\beta)^2 = \prod_{(a,b)\in S}(a - b\beta) \text{ over } \mathbb{Z}[\beta]$$

Then mod n gives:

$$\left.\begin{aligned}\phi(q(\alpha))^2 &\equiv \phi\left(\prod_{(a,b)\in S}(a - b\alpha)\right)\\ \phi(t(\beta))^2 &\equiv \psi\left(\prod_{(a,b)\in S}(a - b\beta)\right)\end{aligned}\right\} \equiv \prod_{(a,b)\in S}(a - bm) \text{mod } n$$

Let $F(a,b) = b^{\deg f} f(a/b)$. It turns out that:

$$\prod_{(a,b)\in S}(a - b\alpha) \text{ is a square in } \mathbb{Z}[\alpha]$$

Implies:

$$\prod_{(a,b)\in S} F(a,b) \text{ is a square in } \mathbb{Z}$$

Moreover, the converse almost holds. Therefore, we can work as before: find (a,b) pairs such that: F(a,b) is B-smooth, compute their exponent vectors and find dependencies. To find pairs, we fix values of b and sieve over a. We do the same for g and Z[β] and find S that satisfies both conditions. Mips years required to factor a number with the GNFS:

Bits	Mips-years
512	30,000
768	2×10^8
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

Complexity of the Number Field Sieve (NFS): The best known classical method for factoring numbers is currently the Number Field Sieve (NFS)[47]. It has an asymptotic run time of approximately $\exp(c\,(\ln n)^{1/3}(\ln n \ln \ln n)^{2/3})$, reducing the exponent over the continued fraction factorization algorithm and quadratic sieve. There are three values of c relevant to different flavors of the method[36]. For the special case of the algorithm applied to numbers near a large power, $c = (32/9)^{1/3} = 1.526285$ for the general case applicable to any odd positive number, which is not a power, $c = (64/9)^{1/3} = 1.922999$ and for a version using many polynomials (Coppersmith[43]), $c = \frac{1}{3}(92 + 26\sqrt{13})^{1/3} = 1.901883$, where n is the number to be factored. As the length of the number n is O(ln n) this means that the factoring algorithm is exponential in the length of n. The factorization of a 512-bit number using NFS method requires approximately 10^{19} steps. Using quantum oracle, one can achieve a better time complexity[21,22].

- The point: wisely choose f and g so that there values near 0 are small and therefore likely to be smooth.
- Specifically, we choose f and g of degree: $d \approx (\log n/\log \log n)^{1/3}$ and the F(a,b) and G(a,b) values we test for smoothness have size roughly $n^{2/d}$ (versus $n^{1/2}$ for QS).
- Conjectured time for optimal parameter choice: $e^{(c+o(1))\cdot(\log n)^{1/3}\cdot(\log\log n)^{2/3}}$ with $c \approx 2$
- Successfully factored 512-bit and 524-bit composites, at considerable effort.
- Appears scalable to 1024-bit composites using custom-built hardware, thereby threatening the security reliability of RSA 1024 public key cryptosystems, which rely on the hardness factoring composite integer of that order.

A recent research trend has been to design special-purpose hardware on which factoring algorithm such as the number field sieve might be faster or more cost-effective than on conventional general-purpose computers. Among the noteworthy proposals are Shamir's TWINKLE machine[52], Bernstein's circuits[53] and the TWIRL machine of Shamir and Tromer[54]. Shamir and Tromer[53] estimate that the relation-generation stage of the number field sieve for factoring a 1024-bit RSA modulus can be completed in less than a year by a machine that would cost about US$ 10 million to build and that the linear algebra stage is easier. Such special-purpose hardware has yet to be built (unless it has been built in secret), so it remains to be seen if this work will have any impact on the size of RSA moduli used in practice.

CONCLUSIONS

In this review paper attempt has been made to present in a simple manner a very difficult topic of integer factorization that is the connerstone of the security behind RSA cryptosystems.As a review paper most of the work presented here have been sourced from variant research works that came to our notice. We hope it will be of good use to the future budding cryptanalyst and number theorists. It has been pointed out that the security of RSA depends on the difficulty of factorizing n into its constituent primes p and q. With the increase in the speed of computers today and the improvement of factorization methods, it becomes increasingly feasible to factorize large numbers. From this perspective, choosing a longer key is essential to safeguard against attacks based on factorization. However, choosing a longer key may incur higher overhead in performing encryption and decryption to secure your data. The question for the crypto designer to ponder, is given a certain amount of budget and time, what should be the length of the key that is reasonably secure without causing unnecessary performance degradation?

REFERENCES

1. Rabah, K., 2004. Data Security and Cryptographic Techniques: A Review ITJ., 3: 106-132.
2. Diffie, W. and M.E. Hellman, 1966. New Directions in Cryptography. IEEE Transaction on Information Theory, pp: 644-654.
3. Rabah, K., 2005. Security of the Cryptographic Protocols Based on Discrete Logarithm Problem (DLP). In Press.
4. Rivest, R., A. Shamir and L. Adleman, 1978. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Comm. of the ACM., 21: 120-126.
5. Menezes, A., P. Van Oorschot and S. Vanstone, 1997. Handbook of Applied Cryptography. CRC Press.
6. Rabin, M.O., 1979. Digital Signature and public-key functions as intractable as factorization, MIT Laboratory of Computer Science, Technical report, MIT/LCS/TR-212,
7. Rabah, K., 2005. Secure implementation of message digest, authentication and digital signature. ITJ., 4: 204-221.
8. Leutwyler, K., 1994. Superhack: Forty quadrillion years early, a 129-Digit Code Is Broken. Sci. Am., 271: 17-20.
9. Cowie, J., B. Dodson, R. Marije Elkenbracht-Huizing, A.K. Lenstra, P.L. Montgomery and J. Zayer, 1996. A World Wide Number Field Sieve Factoring Record: On to 512 bits. In: Kim, K. and T. Matsumoto (Eds.), Advances in Cryptology-Asiacrypt '96, Lecture Notes in Computer Science # 1163, Springer-Verlag, Berlin, pp: 382-394. http://ftp.cwi.nl/herman/NFSrecords/RSA-155
10. Morrison, M.A. and J. Brillhart, 1975. A method of factorization and the factorization of F7. Math. Comp., 29: 183-205.
11. Richard Schroeppel linear sieve-http://cboyer.club.fr/multimagie/English/Schroeppel.htm
12. Pomerance, C., 1982. Analysis and comparison of some integer factoring algorithms, Computational Methods in Number Theory: Part 1, Lenstra, H.W. Jr. and R. Tijdeman, (Eds.), Math. Centre Tract 154, Math. Centre Amsterdam, pp: 89-139.
13. Pomerance, C., 1985.The quadratic sieve factoring algorithm, Advances in Cryptology Eurocrypt'84, Springer-Verlag, Lectures Notes in Computer Science, 209: 169-182.
14. Pomerance, C., 1996. A Tale of Two Sieves. Not. Amer. Math. Soc., 43: 1473-1485.
15. Lenstra, A.K. and M.S. Manasse, 1994. Factoring with two large primes. Mathematics of Computation, 63: 785-798.
16. Pollard, J., 1993. Factoring with cubic integers. The Development of the number field sieve. Lecture Notes in Mathematics, 1554: 4-10.
17. Dodson, B. and A.K. Lenstra, 1995. NFS with four large primes: An explosive experiment, Proceedings Crypto 95, Lecture Notes in Comput. Sci., 963: 372-385.
18. Bundesamt für Sicherheit in der Informationstechnik. Forscher stellen neuen Weltrekord auf: Zahl RSA200 zerlegt. May 9, 2005
20. A 248-digit factorization using SNFS, http://www.rkmath.rikkyo.ac.jp/~kida/snfs248e.htm. Victor Miller's number theory mailing list archive, available at _ http://www.listserv.nodak.edu
21. Shor, P.W., 1994. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, Goldwasser, S., (Ed.) (IEEE Computer Society Press, Los Alamitos, CA), pp: 124-134.
22. Shor, P.W., 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Computing, 26: 1484-1509.

23. Bell, E.T., 1986. The Prince of Mathematicians: Gauss. Ch. 14 in Men of Mathematics: The Lives and Achievements of the Great Mathematicians from Zeno to Poincaré. New York: Simon and Schuster, pp: 218-269.
24. Mahoney, M.S., 1994. The Mathematical Career of Pierre de Fermat, 2nd rev. (Edn.) Princeton, NJ: Princeton University Press., pp: 1601-1665.
25. Kraitchik, M., 1929. Recherches sur la th'eorie des nombres, volume II. Gauthier Villar, Paris.
26. Pollard, J.M., 1974. Theorems on Factorization and Primality Testing. Proc. Cambridge Phil. Soc. 76: 521-528.
27. PARI/GP-(Computational Number Theory): http://pari.math.u-bordeaux.fr/
28. Rabah, K., 2005. Theory and implementation of elliptic curve cryptography. J. Applied Sci., 5: 604-633.
29. Silverman, J.H., 1985. The Arithmetic of Elliptic Curves. Springer-Verlag.
30. Silverman, J.H. and J. Tate, 1992. Rational Points on Elliptic Curves. Springer-Verlag.
31. Koblitz, N., 1993. Introduction to Elliptic Curves and Modular Forms. Springer-Verlag.
32. Lenstra, H.W. Jr., 1987. Factoring Integers with Elliptic Curves. Ann. Math., 126: 649-673.
33. Dixon, B. and A.K. Lenstra, 1993. Massively parallel elliptic curve factoring, Proc. Eurocrypt '92, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 658: 183-193.
34. Brent, R.P., 1986. Some Integer Factorization algorithms using elliptic curves. Austral. Comp. Sci. Comm., 8: 149-163.
35. Montgomery, P.L., 1987. Speeding the pollard and elliptic curve methods of factorization. Math. Comput., 48: 243-264.
36. Lenstra, A.K. and H.W. Jr. Lenstra, 1990. Algorithms in Number Theory. In Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (J. van Leeuwen, Ed.). Amsterdam: Netherlands, Elsevier, pp: 673-715.
37. Brent, R.P., 1999. Factorization of the tenth Fermat number, Math. Comp., 68: 429-451.
38. Dodson, B. Elliptic Curve Factoring Record, http://www.loria.fr/~zimmerma/records/factor.html
39. Goldwasser and J. Kilian. 1986. Almost all primes can be quickly certified. In Proc. 18th STOC. ACM., pp: 316-329.
40. Pollard, J.M., 1978. Monte Carlo methods for index computation (mod p), Mathematics of Computation, 143: 918-924.
41. Eldershaw, C. and R.P. Brent, 1995. Factorization of large integers on some vector and parallel computers, Proceedings of Neural, Parallel and Scientific Computations, 1: 143-148.
42. Brent, R.P. and J.M. Pollard, 1981. Factorization of the eighth fermat number, Mathematics of Computation, 36: 627-630. MR 83h:10014.
43. Don Coppersmith, 1994. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. Mathematics of Computation, 62: 333-350.
44. Wiedemann, D.H., 1986. Solving sparse linear equations over finite fields, IEEE Trans. Information Theory , 32: 54-62.
45. Montgomery, P.L., 1994. A Survey of Modern Integer Factorization Algorithm, 7: 337-38.
46. Davis, J.A. and D.B. Holdridge, 1984. Factorization using the quadratic sieve algorithm, in D. Chaum, Ed., Advances in Cryptology: proceedings of CRYPTO 83, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 22-24, 1983, New York: Plenum Press, pp: 103-113.
47. Dodson, B., A.K. Lenstra, 1995. NFS with four large primes: An explosive experiment, Springer-Verlag, pp: 372-385.
48. Lenstra, A.K., H.W. Jr. Lenstra, M.S. Manasse and J.M. Pollard, 1990. The number field sieve, Proc. 22nd ACM Symp. Theory of Computing, pp: 564-572.
49. SIAM News, 1990. Number field sieve produces factoring breakthrough, SIAM News, 23: 4.
50. Murphy, B.A., 1998. Modelling the yield of number field sieve polynomials, Algorithmic Number Theory. ANTS III, LNCS, Springer-Verlag, Berlin, 1443: 137-150.
51. Couveignes, J.M., 1993. Computing a square root for the number feld sieve. In: Lenstra, A.K. and H.W. Jr. Lenstra. The development of the Number Field Sieve, vol. 1554 of Lecture Notes in Mathematics. Springer-Verlag, pp: 95-102.
52. Lenstra, A.K. and A. Shamir, 2000. Analysis and optimization of the TWINKLE factoring device, Advances in Cryptology, EUROCRYPT 2000, Lecture Notes in Computer Science, Springer-Verlag, 1807: 35-52.
53. Bernstein, D., 2001. Circuits for integer factorization: a proposal, preprint.
54. Shamir, A. and E. Tromer, 2003. Factoring large numbers with the TWIRL device, Advances in Cryptology | CRYPTO 2003, Lecture Notes in Computer Science, 2729 (2003), Springer-Verlag, pp: 1-26.