Factoring Algorithms

Pollard’s $p - 1$ Method

This method discovers a prime factor $p$ of an integer $n$ whenever $p - 1$ has only small prime factors.

Input: $n$ (to factor) and a limit $B$
Output: a proper factor of $n$ or "fail"

$a = 2$
for (i = 2 to B) {
    a = a\^i \bmod n
    if ( ( g = \gcd(a - 1, n) ) > 1) {
        print "g divides n"
        stop
    }
}
print "fail"
Example of Pollard’s $p - 1$ factoring method.

Factor $n = 6437$. Use $B = 5$ and $a = 2$.

For $i = 2$, we have $a = 2^2 \mod n = 4$. 
$\gcd(4 - 1, n) = 1$.

For $i = 3$, we have $a = 4^3 \mod n = 64$. 
$\gcd(64 - 1, n) = 1$.

For $i = 4$, we have $a = 64^4 \mod n = 2394$. 
$\gcd(2394 - 1, n) = 1$.

For $i = 5$, we have $a = 2394^5 \mod n = 4306$. 
$\gcd(4306 - 1, n) = 41$.

Thus, 41 divides $n$. In fact, $n = 41 \cdot 157$
Note that at the end of the $i$-th iteration of the loop we have $a \equiv 2^i! \pmod{n}$, so $a \equiv 2^i! \pmod{p}$ if $p$ divides $n$.

When $i$ is large enough so that $p - 1$ divides $i!$, say, $i! = (p - 1)m$ for some $m$, we will have

$$a \equiv 2^i! \equiv (2^{p-1})^m \equiv 1^m \equiv 1 \pmod{p},$$

by Fermat’s little theorem, so $p$ divides $a - 1$. If $p$ also divides $n$, then $p$ divides $g = \gcd(a - 1, n)$.

Occasionally, Pollard’s $p-1$ method has a spectacular success, but it is unlikely to factor an RSA public modulus $n$.

However, when generating a large prime $p$ for RSA one should factor $p - 1$ and be sure it contains a large prime factor. (A prime factor $q$ of $p - 1$ is “large” if no adversary can do $q$ operations.)
Quadratic Sieve Method

Recall this theorem:

**Theorem.** If \( n = pq \) is the product of two distinct primes, and if \( x^2 \equiv y^2 \pmod{n} \), but \( x \not\equiv \pm y \pmod{n} \), then \( \gcd(x + y, n) = p \) or \( q \).

**Proof:** We are given that \( n \) divides \((x + y)(x - y)\) but not \((x + y)\) or \((x - y)\). Hence, one of \( p, q \) must divide \((x + y)\) and the other must divide \((x - y)\).

In fact, if \( n \) has more than two prime factors and the congruence conditions of the theorem hold, then \( \gcd(x + y, n) \) and \( \gcd(x - y, n) \) will be proper factors of \( n \) even if they are not prime. The conditions fail to lead to a proper factor of \( n \) only in case \( n \) is a power of a prime.
The quadratic sieve algorithm tries to factor $n$ simply by finding $x$ and $y$ with $x^2 \equiv y^2 \pmod{n}$, ignoring the conditions $x \not\equiv \pm y \pmod{n}$. (It just hopes for the best. Usually, it finds several such pairs $x, y$. Each pair succeeds in factoring $n$ with probability at least 1/2.)
**Definition.** An integer \( k \) is a *square* if there exists an integer \( x \) so that \( k = x^2 \).

The quadratic sieve method tries to factor \( n \) by finding two congruent squares modulo \( n \).

How can one recognize a square?

Multiple choice question:

Which of these numbers is a square?

a. 21
b. 23
c. 25
d. 27
e. 29
Which of these numbers is a square?

a. 431641
b. 431643
c. 431645
d. 431647
e. 431649

This is harder.
Suppose I give you the prime factorizations of the numbers.

Which of these numbers is a square?

a. $431641 = 7^2 \cdot 23 \cdot 383$

b. $431643 = 3 \cdot 143881$

c. $431645 = 5 \cdot 131 \cdot 659$

d. $431647 = 17 \cdot 25391$

e. $431649 = 3^4 \cdot 73^2$

**Theorem.** If $n = \prod_{i=1}^{k} p_i^{e_i}$ is the prime factorization of $n$ into the product of powers of distinct primes, then $n$ is square if and only if all exponents $e_i$ are even numbers.
The quadratic sieve factoring algorithm finds congruences $x^2 \equiv y^2 \pmod{n}$ as follows.

Generate many “relations” $j^2 \equiv m \pmod{n}$, where $m$ is small and therefore easy to factor. One way to do this is to try $j$ slightly larger than $\sqrt{n}$. Then $j^2 \pmod{n} = j^2 - n$ will be small compared to $n$.

Factor (some of) the numbers $m$ and match their prime factors to form a product of some $ms$ in which each prime occurs as a factor an even number of times, so it is a square. Let $y^2$ be the product of these $ms$. Let $x$ be the product of the $j$s in the relations used to make $y^2$. Then $x^2$ is the product of the $j^2$s, which is congruent to the product of the $ms$. This product is $y^2$ by the choice of relations.
Example. Let us factor $n = 1649$. Note that $\sqrt{n} \approx 40.6$, so the numbers $41^2 \mod n$, $42^2 \mod n$, \ldots, will be fairly small compared to $n$. We have

$$41^2 \equiv 1681 \equiv 32 = 2^5 \pmod{1649},$$

$$42^2 \equiv 1764 \equiv 115 = 5 \cdot 23 \pmod{1649},$$

$$43^2 \equiv 1849 \equiv 200 = 2^3 \cdot 5^2 \pmod{1649}.$$

Now $32 \cdot 200 = 2^8 \cdot 5^2 = 80^2$ is a square. Therefore,

$$(41 \cdot 43)^2 \equiv 80^2 \pmod{1649}.$$  

Note that $41 \cdot 43 = 1763 \equiv 114 \pmod{1649}$ and that $114 \not\equiv \pm 80 \pmod{1649}$. We get the factors of 1649 from $\gcd(114 - 80, 1649) = 17$ and $\gcd(114 + 80, 1649) = 97$, so $1649 = 17 \cdot 97$. 
In a real application of the quadratic sieve there may be millions of relations \( j^2 \equiv m \pmod{n} \) with \( m \) factored. How can we efficiently match the prime factors of the \( ms \) to make each prime occur an even number of times?

Answer: Use linear algebra over the field \( \mathbb{F}_2 \) with 2 elements.

Let \( p_1, p_2, \ldots, p_b \) be all of the prime numbers that occur as factors of any of the \( ms \).

If \( m = \prod_{i=1}^{b} p_i^{e_i} \), where each exponent \( e_i \geq 0 \), associate \( m \) to the vector

\[
v(m) = (e_1, e_2, \ldots, e_b).
\]

Multiplying \( ms \) corresponds to adding their associated vectors. If \( S \subseteq \{1, 2, \ldots, r\} \), where \( r \) is the total number of relations, then \( \prod_{i \in S} m_i \) is a square if and only if \( \sum_{i \in S} v(m_i) \) has all even coordinates.
Reduce the exponent vectors $v(m)$ modulo 2 and think of them as vectors in the $b$-dimensional vector space $\mathbb{F}_2^b$ over $\mathbb{F}_2 = \{0, 1\}$.

Linear combinations of distinct vectors $v(m)$ correspond to subset sums. Finding a nonempty subset of integers whose product is a square is reduced to finding a linear dependency among the vectors $v(m)$.

We know from linear algebra that if we have more vectors than the dimension $b$ of the vector space ($r > b$), then there will be linear dependencies among the vectors.

Also from linear algebra we have efficient algorithms, such as matrix reduction, for finding linear dependencies. Row reduction over $\mathbb{F}_2$ is especially efficient because adding (or subtracting) two rows is the same as finding their exclusive-or.
The analysis of the quadratic sieve algorithm shows that its time complexity to factor $n$ is about

$$e^{\sqrt{(\ln n)(\ln \ln n)}}$$

bit operations.

To understand what this means, consider

$$e^{\sqrt{(\ln n)(\ln \ln n)}} \leq e^{\sqrt{(\ln n)(\ln n)}} = e^{\ln n} = n$$

and

$$e^{\sqrt{(\ln n)(\ln \ln n)}} \geq e^{\sqrt{(\ln \ln n)(\ln \ln n)}} = e^{\ln \ln n} = \ln n.$$

Thus, $e^{\sqrt{(\ln n)(\ln \ln n)}} \leq n^\varepsilon$ for any $\varepsilon > 0$ and $e^{\sqrt{(\ln n)(\ln \ln n)}} \geq (\ln n)^c$ for any constant $c > 0$. That is, the time complexity is subexponential but not polynomial time.
Discrete Logarithms via Index Calculus

There is a faster way to solve \( a^x \equiv b \pmod{p} \) using a method similar to the integer factoring algorithm QS. It is called the \textbf{index calculus method}.

If \( a^x \equiv b \pmod{p} \), then we write \( x = \log_a(b) \). Note that \( \log_a(b) \) is an integer determined modulo \( p - 1 \) because of Fermat’s theorem: \( a^{p-1} \equiv 1 \pmod{p} \).

\( \log_a(b) \) is called the discrete logarithm of \( b \) to base \( a \). (The modulus \( p \) is usually supressed.)
Choose a factor base of primes $p_1, \ldots, p_k$, usually all primes $\leq B$. Perform the following pre-computation which depends on $a$ and $p$ but not on $b$. For many random values of $x$, try to factor $a^x \mod p$ using the primes in the factor base.

Save at least $k + 20$ of the factored residues:

$$a^{x_j} \equiv \prod_{i=1}^{k} p_i^{e_{ij}} \pmod{p} \text{ for } 1 \leq j \leq k + 20,$$

or equivalently

$$x_j \equiv \sum_{i=1}^{k} e_{ij} \log_a p_i \pmod{p-1} \text{ for } 1 \leq j \leq k + 20.$$
Use linear algebra to solve for the $\text{Log}_{a}p_{i}$.

When $b$ is given, perform the following main computation to find $\text{Log}_{a}b$. Try many random values for $s$ until one is found for which $ba^{s} \pmod{p}$ can be factored using only the primes in the factor base.

Write it as

$$ba^{s} \equiv \prod_{i=1}^{k} p_{i}^{c_{i}} \pmod{p}$$

or

$$(\text{Log}_{a}b) + s \equiv \sum_{i=1}^{k} c_{i}\text{Log}_{a}p_{i} \pmod{p - 1}.$$

Substitute the values of $\text{Log}_{a}p_{i}$ found in the precomputation to get $\text{Log}_{a}b$. 

16
Using arguments like those for the running time of the quadratic sieve factoring algorithms, one can prove that the precomputation takes time

\[ \exp \left( \sqrt{2 \log p \log \log p} \right), \]

while the main computation takes time

\[ \exp \left( \sqrt{\log p \log \log p} \right). \]