Abstract. This paper presents two algorithms that, given an \( n \)-bit positive integer \( m \in 1 + 8\mathbb{Z} \) that is not a square, find an element of \( \mathbb{Z}/m \) that is a non-square or a nonzero non-unit. Under a standard conjecture, the first algorithm takes time \( O(n(\lg n)^3 \lg \lg n) \). Under a new but plausible conjecture, the second algorithm takes expected time \( O(n) \).

Consider the problem of finding a nonzero element of \( \mathbb{Z}/m \) that is not a square in \( (\mathbb{Z}/m)^\ast \), given an odd positive integer \( m \) that is not a square in \( \mathbb{Z} \): in other words, finding an integer \( r \) that is not congruent to a square modulo \( m \), or that has a factor in common with \( m \) without being divisible by \( m \).

There are two standard solutions to this problem. One is a randomized algorithm that takes essentially linear expected time on, for example, a multitape Turing machine. The other is a deterministic algorithm that, under a standard conjecture, takes essentially quadratic time.

This paper presents an improved deterministic algorithm that, under the same conjecture, takes essentially linear time; and an improved randomized algorithm that, under a new but plausible conjecture, takes linear expected time.

In practice, people use—and should continue to use—the original deterministic algorithm: its average time over typical distributions of \( m \) appears to be linear, with a very small constant.

Strategy. One can trivially handle certain cases by inspecting a few bits of \( m \). If \( m \in 3 + 4\mathbb{Z} \) then one can take \( r = -1 \). If \( m \in 5 + 8\mathbb{Z} \) then one can take \( r = 2 \).

Assume, from now on, that \( m \in 1 + 8\mathbb{Z} \). Write \( n = \lceil \lg m \rceil \).

The standard way to find \( r \) is to compute the Jacobi symbol of \( r \) modulo \( m \) for various candidate \( r \)'s:

- If the Jacobi symbol is 0 or 1, try the next \( r \).
- If the Jacobi symbol is \(-1\), stop: \( r \) is a non-square modulo \( m \).
- If the Jacobi symbol is undefined, stop: \( r \) is a nonzero non-unit modulo \( m \).

Even better, the Jacobi-symbol computation has found a factor of \( m \).

Schönhage’s fast-gcd algorithm in [7] computes the Jacobi symbol of two \( O(n) \)-bit inputs in time \( O(n(\lg n)^2 \lg \lg n) \).

There are two popular sequences of candidate \( r \)'s, as described below.

Deterministic algorithms. One popular sequence is the sequence of odd primes: try \( r = 3 \), then \( r = 5 \), then \( r = 7 \), then \( r = 11 \), etc. It is well known that the Jacobi-symbol computation of \( r \) modulo \( m \) becomes simpler and faster when \( r \) is
small; see, e.g., the proof of [2, Theorem 7.8.2]. Most of the work is a single division, computing \( m \mod r \).

How many \( r \)'s are needed? A standard conjecture is that the number of \( r \)'s is \( n + o(n) \) for worst-case moduli \( m \). See, e.g., [1] and [5].

If there are \( n \) Jacobi-symbol computations, and one Jacobi-symbol computation takes time at least \( n \), then the total time is at least \( n^2 \), right? Wrong! One can use the Moenck-Borodin multipoint-evaluation algorithm to compute \( m \mod r \) for many \( r \)'s simultaneously. If there are \( O(n) \) primes \( r \), each having \( O(\lg n) \) bits, then this computation takes time \( O(n(\lg n)^3\lg\lg n) \). See, e.g., [3, Theorem 3.4]. It is easy to complete the Jacobi-symbol computations, and enumerate the primes in the first place, within the same time bound.

It is already standard practice to compute \( m \mod r \) for a few \( r \)'s simultaneously: one reduces \( m \) modulo a single-word product of \( r \)'s, then reduces the result modulo each \( r \). See, e.g., [6, page 146]. The Moenck-Borodin algorithm uses the same idea recursively on a larger scale.

Of course, for typical moduli \( m \), the first few \( r \)'s suffice. One could try a smaller set of primes \( r \) as a preliminary step. The prime 3 suffices for half of all moduli; it can be tried in time \( O(n) \). The prime 5 suffices for half of the remaining moduli; it can be tried in time \( O(n) \). The primes below \( (\lg n)/\lg\lg n \) suffice for most moduli; they can all be tried together in time \( O(n) \), as explained below. The primes below \( n/\lg n \) suffice for practically all moduli; they can be tried in total time \( O(n\lg n\lg\lg n) \).

**Randomized algorithms.** The other popular sequence of \( r \)'s is a sequence of independent uniform random odd integers between 0 and \( m – 1 \); actually, between 0 and \( 2^n – 1 \), so that one can generate each candidate \( r \) by generating \( n – 1 \) random bits. This algorithm finds \( r \) in expected time \( O(n(\lg n)^2\lg\lg n) \): there are at least \( (m – 1)/4 \geq 2^n/8 \) qualifying values of \( r \), so the expected number of Jacobi-symbol computations is at most 4. (This is not the optimal constant.)

Even better, take \( r \) to be a uniform random odd integer between 0 and \( 2^k – 1 \), where \( k \) is much smaller than \( n \); say \( k = 2 \lceil \lg n \rceil \). The bottleneck in the Jacobi-symbol computation is then an \( n \)-bit-by-\( k \)-bit division, which takes time \( O(n) \) by an adaptation of Kaminski's algorithm in [4]. I conjecture that the expected number of Jacobi-symbol computations is bounded.

I should add a table of numerical evidence for this conjecture. I should probably explain Kaminski's algorithm; my recollection is that Kaminski focused entirely on the function-field case. Perhaps I should just tell people to select \( r \) as a sum of \( 10 \) random powers of 2, with exponents bounded by \( n/(\lg n)^3 \), although I have to be a bit more careful with constants in this case to make the conjecture plausible.

**References**


Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

Email address: djb@cr.yp.to