Improvements to Lucas-sequence modular square roots and primality testing

Mike Hamburg

Rambus Inc.

September 12, 2026

Abstract. Lucas sequences are a helpful tool in mathematical and cryptographic calculations, providing in particular an efficient way to exponentiate in a quotient ring $R[x]/(x^2 - Px + Q)$. As with exponentiation in other finite rings and fields, we can use the periodic nature of these sequences to find roots of polynomials. Since they behave differently in the ring $\mathbb{Z}/N$ depending on whether N is prime, Lucas sequences are also useful for primality testing. In this paper, we discuss improvements to Lucas-sequence algorithms for square roots and heuristic primality testing.

Our first application is modular square roots. It is straightforward to take square roots modulo primes $p \equiv \{3, 5, 7\} \bmod 8$. When $p \equiv 1 \bmod 8$, and especially when $p - 1$ is divisible by many powers of 2, Müller's algorithm and Kim-Koo-Kwon are attractive options. Both of these use Lucas sequences. Here we show how to simplify and speed up Kim-Koo-Kwon. We also show a variant on Müller's algorithm which works even when $p \equiv 3 \bmod 4$, which would be useful if p were secret.

Our second application is heuristic primality testing. The Baillie-PSW primality test combines a strong Fermat test with a strong Lucas test. The recent Baillie-Fiori-Wagstaff variant strengthens Baillie-PSW. Here we show an improved variant, `SuperBFPSW`, which is stronger than Baillie-Fiori-Wagstaff, but also faster than the original Baillie-PSW.

1 Introduction

The Lucas sequences $U_n(P, Q)$ and $V_n(P, Q)$ are defined by the recurrence:

$$U_{n+2}(P, Q) = PU_{n+1}(P, Q) - QU_n(P, Q)$$
$$V_{n+2}(P, Q) = PV_{n+1}(P, Q) - QV_n(P, Q)$$

with starting conditions:

$$U_0(P, Q) = 0 \qquad V_0(P, Q) = 2$$
$$U_1(P, Q) = 1 \qquad V_1(P, Q) = P.$$

These sequences are defined on any ring R with unit, but we will assume that R is commutative. When Q is invertible, they can be extended to negative n as well. The sequence $V_n(P, 1)$ is especially fast and simple to compute, and uses a Montgomery-like ladder which may be helpful for side-channel defense.

1.1 Outline

Here we study two applications of Lucas sequences, using related techniques. In Section 2, we discuss notation, as well as related work on square root algorithms. In Section 3 we exploit identities on the Lucas sequences as well as batch calculations to simplify and speed up the Kim-Koo-Kwon square root algorithm. We present our improved algorithm, `Plucky`, in Section 3.1. In Section 4 we then use these and other identities to simplify and speed up the Baillie-Fiori-Wagstaff primality test, allowing it to use sequences with $Q = 1$ and to skip certain steps.

This concludes our main results. In Appendix A we discuss variants on `Plucky`. In Appendix B we discuss running Müller's algorithm modulo 3-mod-4 primes as well, and how to keep the modulus secret. In Appendix C we prove our lemmas for the Baillie-Fiori-Wagstaff test, and in Appendix D we spell out the quadratic Hensel-lifting algorithm to find the square root of an integer, which is used in the primality tests.

2 Notation and related work

2.1 Modular square roots

Let x be an element of a ring R. Then $y \leftarrow \sqrt{x}$ indicates any element of R such that $y^2 = x$. We are particularly interested in the case of a prime-order finite field $\mathbb{F}_p$. Our algorithm `Plucky` also works for extension fields $\mathbb{F}_q$, where q is a power of a prime p, but there are dedicated algorithms for these fields — see e.g. [Adj2013][1] and its references. `Plucky` can compute inverse square roots with minimal modifications, for applications such as point decompression on Edwards curves [Ber2012; Ham2015].

Let $\chi(x) := x^{(q-1)/2}$ be the quadratic character on $\mathbb{F}_q$. When $q = p$ this is the Jacobi symbol $\left(\frac{x}{p}\right)$, and otherwise it is $\left(\frac{\text{Norm}_{q/p}(x)}{p}\right)$, and implementations typically compute $\chi(x)$ this way instead of calculating an exponentiation. An element $x \in \mathbb{F}_q$ has exactly $1 + \chi(x) \in \{0, 1, 2\}$ square roots. If x has any square roots in $\mathbb{F}_q$ it is said to be a *quadratic residue* in that field, and otherwise a *quadratic nonresidue*. Call the set of quadratic residues $\mathcal{Q}(\mathbb{F}_q)$.

For primality testing of an odd positive integer N define $\chi_N(x) := x^{(N-1)/2} \in \mathbb{Z}/N$. This value might not be ± 1, but if it is not (and if N doesn't divide x) then N cannot be prime. Even if $\chi_N(x) = \pm 1$ it doesn't always equal $\left(\frac{x}{N}\right)$; see Lemma 2 in Appendix C for a case analysis.

2.2 Batching inversions and symbol calculations

Some of our performance improvements rely on batching, meaning that several slow operations can be computed for only slightly more than the price of one. Montgomery's batch inversion trick is well-known: one can calculate (x^{-1}, y^{-1})

[1] We abbreviate our references with a 4-digit year because several of them are more than 100 years old.

as $(y, x) \cdot (xy)^{-1}$, and this extends to a batch of n inversions at the cost of one inversion and $3(n-1)$ multiplications. It is also possible to batch $\chi(x)$ or $\chi_N(x)$ with inversion. When computing $\chi_N(x)$ with exponentiation (e.g. when testing N for primality, or in a small base field $\mathbb{F}_p$), one can compute

$$t :\equiv x^{\frac{N-3}{2}}; \qquad \chi_N(x) \equiv tx; \qquad x^{-1} = t^2 x \text{ if } \chi_N(x) = \pm 1.$$

For large N or p, it is faster to compute inversion or the Jacobi symbol using the Euclidean algorithm and related algorithms such as Bernstein-Yang [Ber2019; Ham2021; Ara2023]. The reduction steps are the same for Jacobi symbols and for inversion, so both can be computed in a batch.

We can also batch $\left(\frac{x}{N}\right)$ with inverses of one or more y_i (where x and each y_i is invertible) by calculating the symbol and inverse of $x \cdot \prod y_i^2$ in a batch. This approach does not necessarily work for χ_N when N is composite: we know that $\left(\frac{y_i}{N}\right)^2 = 1$ but not that $\chi_N(y_i)^2 = 1$.

2.3 Atkin, Tonelli-Shanks and its variants

Next we describe prior work on square root algorithms. The performance of these algorithms depends on $\nu(q-1)$, where $\nu(n)$ is the largest integer such that $2^{\nu(n)}$ divides n. Let $\mathbf{S}, \mathbf{M}, \mathbf{m}, \mathbf{I}, \mathbf{L}$ be the costs of **S**quaring, **M**ultiplication, **m**ultiplication by a small integer, **I**nversion, and **L**egendre symbols respectively.

When building cryptographic primitives that need square roots, usually we select $q = p \equiv 3 \pmod 4$. In this case $\sqrt{x} \equiv \pm x^{(q+1)/4}$. In the case of Ed25519 [Ber2012], instead $q = p \equiv 5 \pmod 8$. Atkin's algorithm [Atk1992] covers this case, and is fast and deterministic.

When $q \equiv 1 \pmod 8$, or equivalently $\nu(q-1) \geq 3$, the Tonelli-Shanks algorithm [Ton1891; Sha1973] is a common choice. Tonelli-Shanks costs $\Omega(\log q + \nu(q-1)^2) \cdot \mathbf{S}$ in expectation, which makes it slow when $\nu(q-1)$ is large, most notably for the NIST secp224r1 elliptic curve modulus $2^{224} - 2^{96} + 1$ where $\nu(q-1) = 96$. It can be sped up to $O\left(\log(q) + \frac{\nu(q-1)\log\nu(q-1)}{\log\log\nu(q-1)}\right) \cdot \mathbf{S}$ with precomputed information about $\mathbb{F}_q$ [Ber2001; Sut2011; Sar2020; Por2023].

Without precomputation, Tonelli-Shanks usually also needs to find a quadratic nonresidue, which costs $2\mathbf{L}$ in expectation, or in a field $\mathbb{F}_p$, at most $(\log p)^2 \cdot \mathbf{L}$ assuming the generalized Riemann hypothesis [Lam2016].

2.4 Müller's algorithm

Müller's algorithm [Mül2004] may be seen as a Lucas-sequence variant of the Cipolla-Lehmer algorithm [Cip1903; Leh1969]. The algorithm uses the Lucas sequence $V_n(P, 1)$, which can be calculated especially efficiently using Algorithm 1. Müller's algorithm is shown in Algorithm 2.

The search for a suitable seed a is nondeterministic, succeeding on each attempt with probability $\frac{1}{2} + O(1/q)$, it costs $\approx 2\mathbf{L}$ in expectation. Evaluating the Lucas sequence costs $\ell\mathbf{S} + (\ell - \nu(q-1))\mathbf{M}$. The inversion at the end can

Algorithm 1 Computing a norm-1 Lucas sequence out to $V_n(P, 1)$

Require: Inputs $P \in \mathbb{F}_q$ and positive $n \in \mathbb{Z}$
Ensure: Return $V_n(P, 1)$ and optionally $V_{n+1}(P, 1)$
1: Write n in binary as $n =: \sum_{i=0}^{\ell-1} 2^i \cdot n_i$.
2: Set $(V, W) \leftarrow (2, P)$
3: **for** $i = \ell - 1$ **down to** 0 **do**
4: **if** $n_i = 0$ **then** $(V, W) \leftarrow (V^2 - 2, V \cdot W - P)$
5: **else** $(V, W) \leftarrow (V \cdot W - P, W^2 - 2)$
6: **return** $V_n(P, 1) = V$, and optionally $V_{n+1}(P, 1) = W$

Algorithm 2 Müller's square root algorithm

Require: Quadratic residue $x \in \mathbb{F}_q$ where $q \equiv 1 \pmod 4$
Ensure: Return $y = \pm\sqrt{x}$
1: **if** $x = 0$ **then return** $y = 0$
2: Set $a \leftarrow 1$ ▷ *If x is secret, this must be random instead.*
3: **while** $\chi(a^2 x - 4) \neq -1$ **do**
4: Randomly resample $a \xleftarrow{R} \mathbb{F}_q$.
5: Use Algorithm 1 to compute $V \leftarrow V_{(q-1)/4}(a^2 x - 2, 1)$
6: **return** $y \leftarrow V/a$

be cheaper if an easier-to-invert a is used (such as $a = 1$, a small number or a power of 2). If x is secret then these cannot be used, but the inversion can be batched with the calculation of $\chi(a^2 x - 4)$.

2.5 Pocklington-Peralta and Kim-Koo-Kwon

Square roots can also be computed using the Pocklington-Peralta algorithm [Poc1917; Per1986], which succeeds with probability $1 - (1/2)^{\nu(q-1)-1} + O(1/q)$. If $\nu(q-1) \geq 3$, this is a higher probability than Müller's 1/2. For the NIST secp224r1 modulus $q = 2^{224} - 2^{96} + 1$, this algorithm almost always succeeds in one attempt. But it is slower than Müller's algorithm on average, because it requires exponentiation in a ring $R := \mathbb{F}_q[t]/(t^2 + x)$. The main case of Sze's algorithm [Sze2011] is similar but uses an even slower ring.

Kim et al. [Kim2022] have improved the Pocklington-Peralta algorithm by replacing exponentiation in R with a Lucas sequence. Their algorithm from [Kim2022, Algorithm 3] is shown in Algorithm 3 with slight notation changes. As written, this algorithm assumes x is a nonzero square in $\mathbb{F}_q$.

The Kim-Koo-Kwon algorithm uses $1\mathbf{I} + (\ell + O(1))(\mathbf{S} + \mathbf{M})$ in each pass, plus $1\mathbf{I}$ in the final pass. Its performance is thus competitive with Müller's algorithm, depending on q and on the arithmetic implementation.

Algorithm 3 The Kim-Koo-Kwon square root algorithm

Require: $x \in \mathbb{F}_q$ with $q \equiv 1 \pmod 4$ and $\chi(x) = 1$
Ensure: Return $y = \pm\sqrt{x}$
1: Factor $q-1$ as $q_o \cdot 2^{\nu(q-1)}$

2: Choose a random $a \xleftarrow{R} \mathbb{F}_q$
3: **if** $a^2 + x = 0$ **then** go to step 2

4: Calculate $(a^2+x)^{-1}$
5: Set $P \leftarrow 2(a^2-x)(a^2+x)^{-1}$ and $b \leftarrow 2a(a^2+x)^{-1}$
6: Calculate $(V, W) \leftarrow (V_{q_o}(P,1), V_{q_o+1}(P,1))$ using Algorithm 1
7: **if** $V = \pm 2$ **then** go to step 2 ▷ *Retry with probability* $\approx 2^{1-\nu(q-1)}$
8: **else if** $V = 0$ **then return** $y \leftarrow 2bx/W$

9: **while** $V \neq 0$ **do** ▷ *Calculate* $V_{q_o \cdot 2^i}$*; always finishes in* $< \nu(q-1)$ *steps*
10: $\quad (V_{\text{prev}}, W_{\text{prev}}, V, W) \leftarrow (V, W, V^2 - 2, V \cdot W - P)$
11: **return** $y \leftarrow \frac{V_{\text{prev}} \cdot bx}{V_{\text{prev}} P/2 - W_{\text{prev}}}$

3 Extracting square roots from Lucas sequences

In this section we briefly review the properties of Lucas sequences that allow the Kim et al. and Müller square root algorithms to work. Most of these properties generalize to finite commutative rings, which we will need for the primality testing section. Lucas sequences satisfy the useful identities:

$$V_{2n} = V_n^2 - 2Q^n \tag{1}$$

$$V_{2n+1} = V_n \cdot V_{n+1} - PQ^n \tag{2}$$

which enable Algorithm 1. Letting the discriminant $D := P^2 - 4Q$, it is easily shown by induction on n that:

$$V_{n+1}^2 - PV_nV_{n+1} + V_n^2 Q = -Q^n D. \tag{3}$$

Lucas sequences $U_n(P,Q)$ and $V_n(P,Q)$ in finite rings are periodic when Q is invertible. In most cases, we will set $Q = 1$ and use the sequence $V_n(P,1)$, which simplifies these identities. We will sometimes omit the arguments $(P,1)$ for brevity. Sequences with $Q = 1$ are symmetric, meaning that $V_{-n} = V_n$.

Over a field $\mathbb{F}_q$, we can determine more about the period. Let $\delta := \chi(D)$ and assume that $\delta \neq 0$. Then the period of $V_n(P,1)$ must divide $q - \delta$, and is the least positive n for which $V_n = 2$. The sequence might also be anti-periodic: if $V_m = -2$ for some m, then also $V_{m+n} = -V_n$ for all n.[2] Furthermore, if $P+2$ is a quadratic residue in $\mathbb{F}_q$, then the period divides $(q-\delta)/2$, and otherwise it does not.

[2] These follow because if $V_n = \pm 2$, then $V_{n+1}^2 \mp 2PV_{n+1} + 4 = -D = 4 - P^2$, which has a unique double root at $V_{n+1} = \pm P$.

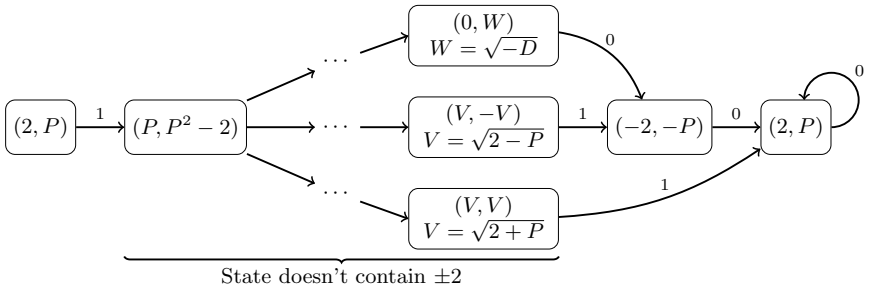


Fig. 1. In a field $\mathbb{F}_q$ where P and $D = P^2 - 4$ are invertible, we have $V_{(q-\chi(D))/2}(P, 1) = 2 \cdot \chi(P + 2) = \pm 2$. This diagram shows the path Algorithm 1 takes to reach $V_n = \pm 2$. Each node shows the state (V, W). A 0 or 1 above an edge indicates the bit n_i. When not in a field, the listed square-root relations still hold if 2 is invertible, but other paths are possible.

When computing $V_n(P, 1)$, Algorithm 1 can only reach the repeat point in a few specific ways. These are described in the following lemma, and the field cases are shown in Figure 1.

Lemma 1 (How V sequences reach 2). *Let R be a commutative ring with unit. Take any $P \in R$ and $n \in \mathbb{N}$, and let $(V, W) := (V_n(P, 1), V_{n+1}(P, 1))$. We have $V_{2n+1}(P, 1) = VW - P$. If $V_{2n+1}(P, 1) = \pm 2$, then*

$$(W \mp V)^2 = 0 \qquad \textit{and} \qquad (W \pm V)^2 = 4 \cdot (2 \pm P)$$

where all the $\pm$ terms have the same sign as the ± 2 in the assumption. If R is a field, or if $R = \mathbb{Z}/N\mathbb{Z}$ where N is square-free, it follows that $V = \pm W$ and $V^2 = W^2 = 2 \pm P$.

Conversely, if $V = \pm W$ and if $P \mp 2$ is invertible in R, then $V^2 = 2 \pm P$ and $V_{2n+1}(P, 1) = \pm 2$.

We prove this lemma in Appendix C.

Now consider using Algorithm 1 to compute $(V_n(P, 1), V_{n+1}(P, 1))$ in a ring where $2PD$ is invertible. If $n > 0$ there must be at least one iteration, and the first iteration's state is $(P, P^2 - 2)$, which cannot contain ± 2.

But suppose that either V_n or V_{n+1} is ± 2. Let $\mathcal{S} := (V, W)$ be the algorithm state immediately before this happens, meaning that $\mathcal{S}$ does not contain ± 2 but the next state does. The ± 2 in the next state must have been computed by Algorithm 1 lines 4 or 5 in one of the following ways:

- $\pm 2 = VW - P$. According to the lemma, $(W \pm V)^2 = 4(2 \pm P)$ and $(W \mp V)^2 = 0$. In a field, $V = \pm W = \sqrt{2 \pm P}$.
- $-2 = V^2 - 2$, so $V^2 = 0$. Applying (3), we get $(2W - PV)^2 = -4D$. In a field, $V = 0$ and $W = \sqrt{-D} = \sqrt{(2 + P)(2 - P)}$.
- $-2 = W^2 - 2$, so $W^2 = 0$. This case is the same, but with V and W swapped.

– $+2 = V^2 - 2$ or $W^2 - 2$. This is impossible in a field, since we assumed that $V \neq \pm 2$ and $W \neq \pm 2$.

Over a field $\mathbb{F}_q$, one of $\{V_{(q-1)/2}(P,1), V_{(q+1)/2}(P,1)\}$ must be ± 2. So if we calculate these values using Algorithm 1, we must obtain along the way a square root of either $2+P, 2-P$ or of $-D = (2+P)(2-P)$. If only one of these values is a quadratic residue, we must obtain that square root. This fact is also useful in a primality-testing context: we must either obtain one of these square roots or a proof that the modulus is composite.

3.1 Simplified Kim et al. algorithm

Algorithm 4 `Plucky`: a simplified square root algorithm

Require: $x \in \mathbb{F}_q$ with $q \equiv 1 \pmod 4$.
Ensure: Return $y = \pm\sqrt{x}$ if it exists, or otherwise an error.

if $x = 0$ **then return** $y \leftarrow 0$

Choose a random $a \xleftarrow{R} \mathbb{F}_q \backslash \{0\}$.
if $a^2 + x = 0$ **then** go to step 2
Calculate $(a^2+x)^{-1}$ and $(4a)^{-1}$ in a batch

Set $V \leftarrow 2$, $P \leftarrow 2 \cdot (a^2 - x) \cdot (a^2 + x)^{-1}, W \leftarrow P$
for $i = \lfloor \log_2 q \rfloor$ **down to** 2 **do** ▷ *Algorithm 1 with termination check*
    if bit i of q is 0 **then** $(V, W) \leftarrow (V^2 - 2, V \cdot W - P)$
    else $(V, W) \leftarrow (V \cdot W - P, W^2 - 2)$
    if $V = 0$ **then return** $y \leftarrow W(a^2+x)(4a)^{-1}$ ▷ *Success!*

if $V = \pm 2$ **then** go to step 2 ▷ *Retry*
else raise error: x isn't square!

Our observations lead directly to an improvement in Kim-Koo-Kwon. As in their algorithm, we special-case $\sqrt{0} = 0$, then choose a random $a \neq 0$ and set

$$P \leftarrow 2 \cdot \frac{a^2 - x}{a^2 + x} \qquad \text{so that} \qquad D = -\frac{16a^2 x}{(a^2+x)^2}.$$

If $x \in \mathcal{Q}(\mathbb{F}_q)\backslash\{0\}$ and $q \equiv 1 \pmod 4$ then we have $\delta = \chi(D) = \chi(-D) = 1$. We calculate $V_{(q-1)/4}(P,1)$ using Algorithm 1. If at any step $V = 0$, we obtain

$$W = \sqrt{-D} = \frac{4a\sqrt{x}}{a^2+x} \qquad \text{from which} \qquad \sqrt{x} = \frac{W(a^2+x)}{4a}.$$

We therefore don't need separate cases for when $V = 0$ occurs, and we don't need to track V_{prev} and W_{prev}. Since the denominator $4a$ is known ahead of time, the inversions $(4a)^{-1}$ and $(a^2+x)^{-1}$ can be batched. We also don't need

to assume that x is square: we can detect this based on whether the calculation ends with $V \in \{0, \pm 2\}$. This must be the case if x is a nonzero square, since we must have $V^2 - 2 = V_{(q-1)/2} = \pm 2$. But if x is nonsquare then we instead have $VW - P = V_{(q+1)/2} = \pm 2$, from which it follows that $V = \sqrt{2 \pm P}$ which cannot be 0 or ± 2.

These changes give the faster, simpler algorithm shown in Algorithm 4. Since "Pocklington-Peralta algorithm with Lucas-sequence improvement by Kim, Koo and Kwon" is getting to be a bit of a mouthful, we will shorten this to "`Plucky`". We can see several possible variations on `Plucky`:

- The setup for an iteration of `Plucky` uses inversion. We could therefore set up n iterations in a batch; call this `Plucky-`n.
- When x is a nonzero square, passes with $\chi(a^2 + x) = -1$ are guaranteed to succeed: they cannot find $\sqrt{2 \pm P}$ since these are not in the field. We could therefore only run passes where $\chi(a^2 + x) = -1$; but the result would be essentially a more complex Müller's algorithm. It could still be useful if combined with batching: calculate $\chi(a_1^2 + x)$ during batch setup, and if it is $+1$ we could skip the first iteration and start with a_2.
- Better yet, we could calculate $\chi(a_1^2 x - 4)$ during batch setup, and if it is -1 then we could branch to Müller's algorithm, which saves $\nu(q-1)$ multiplications. Call this `Plucky-M`, or `Plucky-`n`-M` if n passes are batched.

We note that for $x \neq 0$, the calculation

$$a \xleftarrow{R} \mathbb{F}_q \backslash \{0, \pm\sqrt{-x}\}; \qquad P = 2 \cdot \frac{a^2 - x}{a^2 + x}$$

samples uniformly from $P \in \mathbb{F}_q$ such that $\chi(-D) = \chi(4 - P^2) = \chi(x)$. This is because all $P \neq \pm 2$ are of the form $2(z - x)/(z + x)$ for some $z \neq 0$, and then

$$\chi(4 - P^2) = \chi(16zx/(z + x)^2) = \chi(z) \cdot \chi(x)$$

so the constraint is satisfied when $z = a^2$ for some nonzero a. This means that the state of the Lucas ladder, and whether or when it reaches $V = 0$, is independent of x so long as x is a nonzero square.

3.2 Performance estimates

We summarize in Table 1 the performance of `Plucky` variants in comparison to other algorithms, including variants of Müller where the inversion is batched into χ, or where χ is skipped (as in Algorithm 9 in Appendix B). Note that these are asymptotic: for moderate q, variants of Tonelli-Shanks such as [Ber2001; Sar2020; Por2023] may be faster than Sutherland [Sut2011].

We can see that if $k := \nu(q-1)$ is small to moderate (depending on how much precomputed information is available), then Tonelli-Shanks variants are usually fastest. Müller has two advantages over `Plucky`: by checking χ it avoids fruitless Lucas iterations, which matters when k is small, and also it can skip

Algorithm	Expected runtime	Small $k = 3 \ll \ell$	Large $k \approx \ell$
TS, no prec.	$\mathbf{E} + (1 - 2^{1-k}) \cdot (2\mathbf{L} + \mathbf{E}) + \Theta(k^2)\mathbf{S}$	$\frac{7}{4}\mathbf{E} + \frac{3}{2}\mathbf{L}$	$\Theta(k^2)\mathbf{S}$
TS, prec. RoU	$\mathbf{E} + \Theta(k^2)\mathbf{S}$	$\mathbf{E}$	$\Theta(k^2)\mathbf{S}$
Sutherland	$\mathbf{E} + O\left(\frac{k \log k}{\log \log k}\right)\mathbf{S}$	$\mathbf{E}$	$O\left(\frac{k \log k}{\log \log k}\right)\mathbf{S}$
Müller	$2\mathbf{L} + \mathbf{I} + \ell\mathbf{S} + (\ell - k)\mathbf{M}$	$2\mathbf{L} + \mathbf{I} + \ell(\mathbf{S} + \mathbf{M})$	$2\mathbf{L} + \mathbf{I} + \ell\mathbf{S}$
Müller, batch	$2\mathbf{I_L} + \ell\mathbf{S} + (\ell - k)\mathbf{M}$	$2\mathbf{I_L} + \ell(\mathbf{S} + \mathbf{M})$	$2\mathbf{I_L} + \ell\mathbf{S}$
Müller, no χ	$\mathbf{I} + 2(\ell\mathbf{S} + (\ell - k)\mathbf{M})$	$\mathbf{I} + 2\ell(\mathbf{S} + \mathbf{M})$	$\mathbf{I} + 2\ell\mathbf{S}$
Kim et al.	$\mathbf{I} + (\mathbf{I} + \ell\mathbf{S} + \ell\mathbf{M})/(1 - 2^{1-k})$	$\mathbf{I} + \frac{4}{3}(\mathbf{I} + \ell\mathbf{S} + \ell\mathbf{M})$	$2\mathbf{I} + \ell(\mathbf{S} + \mathbf{M})$
`Plucky`	$(\mathbf{I} + \ell\mathbf{S} + \ell\mathbf{M})/(1 - 2^{1-k})$	$\frac{4}{3}(\mathbf{I} + \ell\mathbf{S} + \ell\mathbf{M})$	$\mathbf{I} + \ell(\mathbf{S} + \mathbf{M})$
`Plucky-`n	$\mathbf{I} + (\ell\mathbf{S} + \ell\mathbf{M})/(1 - 2^{1-k})$	$\mathbf{I} + \frac{4}{3}\ell(\mathbf{S} + \mathbf{M})$	$\mathbf{I} + \ell(\mathbf{S} + \mathbf{M})$
`Plucky-M`	$(\mathbf{I_L} + \ell\mathbf{S} + (\ell - k/2)\mathbf{M})/(1 - 2^{-k})$	$\frac{8}{7}(\mathbf{I_L} + \ell\mathbf{S} + \ell\mathbf{M})$	$\mathbf{I_L} + \ell(\mathbf{S} + \frac{1}{2}\mathbf{M})$
`Plucky-`n`-M`	$\mathbf{I_L} + (\ell\mathbf{S} + \ell\mathbf{M})/(1 - 2^{-k}) - \frac{k}{2}\mathbf{M}$	$\mathbf{I_L} + \frac{8}{7}\ell(\mathbf{S} + \mathbf{M})$	$\mathbf{I_L} + \ell(\mathbf{S} + \frac{1}{2}\mathbf{M})$

Table 1. Asymptotic runtimes of various square root algorithms for public modulus q and secret x. Here $\ell = \log_2 q$ and $k = \nu(q-1)$. The symbol $\mathbf{I_L}$ denotes a batch inversion and Legendre symbol, and $\mathbf{E}$ abbreviates an exponentiation $x^{(q-1)/2^k}$ costing $(\ell - k)\mathbf{S} + o(\ell - k)\mathbf{M}$. Only the dominant terms are listed. The rightmost two columns specialize the cost to $k = 3$ and $k \approx \ell$ respectively. If x is random and not secret, then Müller's algorithm can replace $\mathbf{I}$ with a smaller term. When k is moderate, Müller's algorithm behaves as in the small-k case but `Plucky` behaves as in the large-k case, and `Plucky-M` in between.

k multiplications in Lucas calculations, which matters when k is large. When k is moderate, these advantages are less important and `Plucky` should be faster because it need not compute χ at all, which is especially important in implementations where χ is slow (e.g. if it is computed by exponentiation). Furthermore, with large k the runtime of `Plucky`, while not bounded, has a shorter tail than with Müller.

Overall, the performance of `Plucky` is competitive with Müller. We think that `Plucky` is an attractive choice for embedded implementations, where code size is at a premium, side-channel resistance is required, and a fast χ might not be available.

Müller's algorithm when $q \equiv 3 \bmod 4$ Müller's algorithm requires $(q-\delta)/2$ to be odd, which it ensures by restricting to $q \equiv 1 \bmod 4$ and searching for a such that $\delta = \chi(D) = -1$. When $q \equiv 3 \bmod 4$, we instead would need $\delta = +1$. We can unify these cases by searching for $\chi(-D) = -1$. Of course 3-mod-4 fields have faster square root algorithms, but with this algorithm the timing does not depend on whether $q \equiv 3 \bmod 4$. We show the full algorithm in Appendix B. The same can just as easily be done for `Plucky` but there isn't much point: the advantage of `Plucky` over Müller is that it takes fewer iterations for certain q, and if q is secret then this advantage goes away.

4 Application to the BFW primality test

Our techniques also give insight into improving heuristic or probabilistic primality tests. These tests take an integer $N \geq 2$ and return either `prime?` (with a question mark as it might not guarantee primality) or `composite` (which does guarantee compositeness). A *probabilistic* test is randomized, and provably returns `composite` for any composite N with probability $> \frac{1}{2}$. A *heuristic* test can be randomized or deterministic, but there might exist composite N that reliably fool it.

4.1 Fermat and Euler tests

One family of primality tests is based on Fermat's theorem, which says that $a^{N-1} \equiv 1 \pmod{N}$ for all primes N and all integers $a \not\equiv 0 \bmod N$. This relation does not hold for the majority of a modulo composite N, except for special N called *Carmichael numbers*; as a result this *weak Fermat test* is only heuristic. The *strong Fermat test* instead calculates $\alpha := a^{N_o}$, where $N_o := (N-1)/2^{\nu(N-1)}$ is the odd part of $N-1$, and then $\chi_N(a) = a^{(N-1)/2} = \alpha^{2^{\nu(N-1)-1}}$ by repeated squaring. The strong Fermat test requires that $\alpha \equiv \pm 1$ or that one of the squarings gives -1. The test returns `composite` for at least $3/4$ of all nonzero $a \in \mathbb{Z}/N$ when N is composite. The popular Miller-Rabin probabilistic test [Mil1975; Rab1980] runs several strong Fermat tests with independent random bases a.

One can also run an *Euler test* (also called the *Euler-Jacobi test*, or in the case of a random base a *Solovay-Strassen* test [Art1966; Sol1977]), which checks whether $\chi_N(a) \equiv \left(\frac{a}{N}\right)$. This test is stronger when $\left(\frac{a}{N}\right) = -1$ than when it is $+1$ [Wan2023], but it is weaker than the strong Fermat test [Bai1980].

4.2 The Baillie-PSW and Baillie-Fiori-Wagstaff tests

To reduce the chance of being fooled by Carmichael numbers, some systems run other tests, such as the heuristic Lucas test. This test uses an analog of Fermat's theorem for extension fields $\mathbb{F}_{p^2}$, but replacing the exponentiation with a Lucas sequence calculation, which is faster to compute. The most widespread examples are variations on the heuristic Baillie-Pomerance-Selfridge-Wagstaff (Baillie-PSW) test [Bai1980], which combine a strong Fermat test and a strong Lucas test. Likewise FIPS 186-5 [Sta2023] allows the Miller-Rabin test to be combined with a heuristic Lucas test from [Bai1980]. Variations on Baillie-PSW are used in at least Crypto++ 7.0, Java 10, Golang 1.10.3, GNU MP 6.2.0, LibTomMath 1.3.0, Pari/GP 2.17, SageMath 10, and some versions of FLINT, Maple, and Mathematica [Alb2018; Mas2020; Bai2021].

The Baillie-Fiori-Wagstaff (BFW) heuristic primality test [Bai2021] is a strengthened version of Baillie-PSW, shown in Algorithm 6. Here lines 13 through 16 are additions to Baillie-PSW. They do not add much computational expense, because the sequences U_n, V_n and Q^n can be calculated together. Baillie-PSW and

Algorithm 5 Strong Fermat primality test `StrongFermatTest`(N, a) with optional Euler symbol and inverse

Require: Inputs: odd $N \in \mathbb{N}$ where $N > 2$ and base $a \in (\mathbb{Z}/N)\backslash\{0\}$.
Ensure: If the test determines that N is composite, return `composite`.
Otherwise, return (`prime?`, $\chi_N(a)$, $a^{-1} \pmod N$).

Let $N - 1 =: 2^k \cdot q_o$ where q_o is odd
▷ *If we only need* `prime?` *or* `composite`*, the* β *calculations can be skipped.*
Calculate $(\alpha, \beta) \leftarrow (a^{q_o}, a^{q_o - 1}) \pmod N$
if $\alpha \equiv 1$ **then return** (`prime?`, $+1$, β)
for $m = 0$ **to** $k - 2$ **do**
    if $\alpha \equiv -1 \pmod N$ **then return** (`prime?`, $+1$, $-\beta$)
    $(\alpha, \beta) \leftarrow (\alpha^2, \alpha \cdot \beta) \pmod N$
if $\alpha \equiv -1 \pmod N$ **then return** (`prime?`, -1, $-\beta$)
else return `composite`

Algorithm 6 The BFW primality test

Require: Input $N \in \mathbb{N}$ where $N > 2$.
Ensure: Output `prime?` if N is prime.
Otherwise, hopefully return `composite`.

if $N = 2$ **then return** `prime?`
else if N is even or N is square **then return** `composite`
else if `StrongFermatTest`$(N, 2) =$ `composite` **then return** `composite`

for $D \in [5, -7, 9, -11, 13, \ldots]$ **do** ▷ *Find suitable discriminant* D
    Calculate $\delta \leftarrow \left(\frac{D}{N}\right)$
    if $\delta = 0$ **and** ($|D| < N$ **or** $N \nmid D$) **then return** `composite`
    else if $\delta = -1$ **then break** with D

if $D = 5$ **then** set $(P, Q) \leftarrow (5, 5)$
else set $(P, Q) \leftarrow (1, (1 - D)/4)$

Let $N + 1 =: 2^\ell \cdot r_o$, where r_o is odd
if $U_{r_o}(P, Q) \not\equiv 0$ and $V_{r_o \cdot 2^m}(P, Q) \not\equiv 0 \pmod N$ for each $m \in \{0, 1, \ldots, \ell - 1\}$ **then**
    return `composite` ▷ *Strong Lucas test*
else if $V_{N+1}(P, Q) \not\equiv 2Q \pmod N$ **then**
    return `composite` ▷ *V-Lucas test*
else if $Q^{(N+1)/2} \not\equiv Q \cdot \left(\frac{Q}{N}\right) \pmod N$ **then**
    return `composite` ▷ *Euler criterion test*
else return `prime?`

BFW use the *strong Fermat primality test* shown in Algorithm 5 as a subroutine, as well as a *strong Lucas test* on P, Q chosen by Method A* from [Bai1980].

Method A* chooses the discriminant D of the Lucas sequence as the first element of $\{5, -7, 9, -11, \ldots\}$ with $\left(\frac{D}{N}\right) = -1$, aborting if it finds $\left(\frac{D}{N}\right) = 0$ for $N \nmid D$. It then sets $(P, Q) = (1, (1 - D)/4)$ unless $D = 5$, in which case it sets $(P, Q) = (5, 5)$ instead of $(1, -1)$. To avoid taking exponential time if N is square,[3] we should check that N is not square, either before Method A* or during it if passes some predetermined D. This can checked very quickly using quadratic Hensel lifting, as shown in Appendix D.

4.3 Improving Baillie-PSW and BFW

The Baillie-PSW and BFW tests are reasonably fast, but they would be faster if we could take $Q = 1$. Here we describe first how to accelerate or skip some tests in BFW, and then a stronger test which reinstates some of the conditionally-skipped tests. The proof of our construction is straightforward, but is mostly deferred to appendix C. By way of related work, Underwood [Und2017] also notes that Lucas tests on (P, Q) can follow from an Euler test on Q and a Lucas test on $V_n(P^2/Q - 2, 1)$, but this work is sparse on details and does not discuss strong Lucas or strong Fermat tests.

Our first change is minor, justifying some simple changes to Method A*.

Proposition 1 (Method A^p). *Let $N \geq 13$ be odd, and define Method A^p to check, for each $D \in \{5, -7, 9, -11, 13, -15, \ldots\}$ such that either $D = -15$ or $|D|$ is prime, whether $\left(\frac{D}{N}\right) = -1$. If so, it returns D. If instead $\left(\frac{D}{N}\right) = 0$ it declares that N is composite. By quadratic reciprocity we can rewrite $\left(\frac{D}{N}\right) = \left(\frac{N}{|D|}\right)$, which allows us to work with positive numbers.*

When Method A^* succeeds and outputs (D, P, Q), then N is coprime to $2PQD$. If N is divisible by 3, it is possible that Method A^* declares N composite while Method A^p returns D such that $|(1 - D)/4| = 3$. In all other cases, the two routines return the same D.

The case when A* declares N composite and A^p returns D with $|(1-D)/4| = 3$ cannot change the outcome of the entire test, because such N will always fail the base-Q Fermat test. The next two lemmas are the heart of our improvement over BFW:

Proposition 2 (Accelerated strong Lucas test). *Suppose N is coprime to $2PQD$. Factor $N + 1 =: (2m + 1) \cdot 2^\ell$, and let*

$$\hat{V}_n := V_n(P^2/Q - 2, 1).$$

[3] N being square is unusual but not impossible once the base-2 strong Fermat test is passed: the prime factors of N would be Wieferich primes [Fis2015], of which 1093 and 3511 are the only known examples. Note that if we are willing to assume the generalized Riemann hypothesis, we only need to check the primes below $(\log N)^2$ [Lam2016].

Then N passes the strong Lucas test base (P, Q) if and only if either $\hat{V}_{m+1} \equiv \pm\hat{V}_m$, or if some $\hat{V}_{(2m+1)\cdot 2^k} \equiv 0$ where $0 \le k \le \ell - 2$.

Proposition 3 (Euler-Lucas test instead of V-Lucas test). *Suppose $2PQD \perp N$ and that N passes the accelerated strong Lucas test base (P, Q). Then $\hat{V}_{(N+1)/2} \equiv \pm 2$ and*

$$V_{N+1}(P, Q) \equiv Q \cdot Q^{(N-1)/2} \cdot \hat{V}_{(N+1)/2}.$$

So once the strong Lucas test passes, the V-Lucas test $V_{N+1} \stackrel{?}{=} 2Q$ also passes if and only if they pass the Euler-Lucas test

$$Q^{(N-1)/2} \stackrel{?}{\equiv} \frac{1}{2}\hat{V}_{(N+1)/2}.$$

It follows that the strong Lucas and V-Lucas tests base (P, Q) can be replaced by a Fermat test base Q, and the accelerated strong Lucas and Euler-Lucas tests starting at $(P^2/Q - 2, 1)$. The value $\frac{1}{2}\hat{V}_{(N+1)/2}$ is always ± 1: in fact, according to Lemma 2, it is always the Jacobi symbol $\left(\frac{P^2/Q}{N}\right) = \left(\frac{Q}{N}\right)$. This gives the next proposition:

Proposition 4 (BFW's Euler test always passes). *Suppose N is coprime to $2PQD$, and that the strong Lucas test and V-Lucas test pass. Then $Q^{(N-1)/2} \equiv \left(\frac{Q}{N}\right)$, so this test in [Bai2021] is redundant.*

Since this Euler test is redundant we will skip it, but the test is very fast so the performance improvement is negligible.

Proposition 5 (Special case of $D = 5$). *Suppose N is coprime to $2PQD$ and $\left(\frac{5}{N}\right) = -1$. The accelerated strong Lucas test and Euler-Lucas test pass on $(P, Q) = (5, 5)$ if and only if they pass on $(P, Q) = (1, -1)$ and also $5^{(N-1)/2} \equiv -1$. This also means that Method A from [Bai1980] can be used instead of Method A^*, if this extra check is used.*

Proposition 6 (Redundant calculations and sign simplifications). *Suppose that $P = 1$ and $QD \perp N$ and N passes a base-2 strong Fermat test. Let $T := |Q| = \lfloor (|D|+1)/4 \rfloor$ and $s := -\mathrm{sign}(Q)$. Then a Fermat test on Q is equivalent to one on T. Furthermore, an accelerated strong Lucas test on $V_n(Q^{-1}-2, 1)$ is equivalent to one on $V_n(T^{-1} + 2s, 1)$.*

When changing from Q to T in this manner, the Euler-Lucas test picks up a net sign of $(-s)^{(N-1)/2}/(-s)^{(N+1)/2} = -s$. This changes it from

$$Q^{(N-1)/2} \stackrel{?}{\equiv} \frac{1}{2}V_{(N+1)/2}(Q^{-1} - 2, 1)$$

to

$$T^{(N-1)/2} \stackrel{?}{\equiv} -\frac{1}{2}sV_{(N+1)/2}(T^{-1} + 2s, 1).$$

Since this lemma is only sign-adjustment, it is presented without proof. Finally, if $T = 2^k$, then a separate base-T Fermat test is redundant; furthermore, according to Lemma 2, the Euler-Lucas test is also redundant.

Combining these, we obtain the following theorem:

Theorem 1 (Accelerated BFW test). *An odd number $N \geq 13$ passes the BFW test if and only if it passes the following tests:*

- *Perform a base-2 strong Fermat test, extracting $\chi_N(2)$.*
- *Check that N isn't square, tested using e.g. quadratic Hensel lifting.*
- *Use Method A^p to select $|D|$ with $\left(\frac{N}{|D|}\right) = -1$. Set $T := \lfloor(|D|+1)/4\rfloor$, and $s := +1$ if $|D| \equiv 1 \bmod 4$ or $s := -1$ if $|D| \equiv -1 \bmod 4$.*
- *If $D = 5$, perform a base-5 Euler test, checking that $5^{(N-1)/2} \stackrel{?}{\equiv} -1$.*
- *If T is not a power of 2, then perform a base-T Fermat test, extracting also $\chi_N(T) \equiv T^{(N-1)/2}$ and $T^{-1} \equiv T^{N-2}$. If T is a power of 2, then this test would always pass, and T^{-1} is easy to calculate.*
- *Perform the accelerated strong Lucas test on $V_n(T^{-1}+2s, 1)$. If T isn't a power of 2, also check that $V_{(N+1)/2}(T^{-1}+2s, 1) \equiv -2s\chi_N(T)$.*

If N fails any of these sub-tests then it is composite. The new test is typically faster than a Baillie-PSW or BFW test, because the base-Q Fermat test and the accelerated strong Lucas test together are faster than computing the sequence (U_n, V_n, Q^n). On average per bit of N, the Lucas ladder from [Bai2021] costs $2\mathbf{S} + \mathbf{M} + 2\mathbf{m}$. With our method, using a square-multiply approach, the Fermat and Lucas tests cost $\mathbf{S} + \frac{1}{2}\mathbf{m}$ and $\mathbf{S} + \mathbf{M}$ respectively[4], and when Q is a power of 2 the Fermat test can be skipped. The new algorithm is also easier to fortify against side-channel attack: the regular structure of Algorithm 1 reduces how much information about N leaks through timing, and the Fermat tests can be protected using the Montgomery ladder.

4.4 Strengthening the test

We now propose a stronger, simpler variant of the BFW test, which we call `SuperBFPSW`. This test is shown in Algorithm 7. It is the same as the accelerated BFW test, with the following changes:

- We replace the base-T or base-5 Fermat test with a strong Fermat test.
- Passing the strong Fermat test and accelerated strong Lucas test will guarantee that the Euler-Lucas test $V_{(N+1)/2}(T^{-1}+2s, 1) \equiv -2s\chi_N(T)$ passes, so we can skip that test.
- We may as well move the update before the termination check, because we cannot have $V = \pm W$ or $W = 0$ on the first iteration anyway.

[4] The Fermat test requires extra multiplications to extract T^{-1}, but this is only significant if $N-1$ is divisible by many powers of 2. Alternatively (pointed out by Claude AI), since T is small it can be efficiently inverted using Arazi's lemma.

Algorithm 7 `SuperBFPSW` primality test

Require: Inputs $N \in \mathbb{N}$

Ensure: Return `prime?` if N is prime.
Otherwise, hopefully return `composite`.

if $N \in \{2, 3, 5, 7, 11\}$ **then return** `prime?` ▷ *Deal with small cases*
else if N is even **or** $N < 13$ **or** N is square **then return** `composite`
▷ *First strong Fermat-Euler test*
if `StrongFermatTest`$(N, 2) =$ `composite` **then return** `composite`

for $|D| \in [5, 7, 9, 11, 13, \ldots]$ **do** ▷ *Method A^p*
    if $|D| \neq 15$ **and** $|D|$ is composite **then next** D ▷ *Optionally skip these $|D|$*
    Calculate $\delta \leftarrow \left(\frac{N}{|D|}\right)$
    if $\delta = -1$ **then break** with $|D|$
    else if $\delta = 0$ **then return** `composite`

Set $T \leftarrow \left\lfloor \frac{|D|+1}{4} \right\rfloor$ and $s := \begin{cases} +1 \text{ if } |D| \equiv 1 \bmod 4 \\ -1 \text{ if } |D| \equiv 3 \bmod 4 \end{cases}$
if $T = 2^k$ for some k **then**
    ▷ *A base-T Fermat test would be redundant, so use base 5 instead*
    if `StrongFermatTest`$(N, 5) =$ `composite` **then return** `composite`
    Calculate $T^{-1} \bmod N$ using Montgomery reduction
else
    if `StrongFermatTest`$(N, T) =$ `composite` **then return** `composite`
    The strong Fermat test returns T^{-1}.

Let $\ell \leftarrow \nu(N + 1)$, the number of trailing zero-bits of $N + 1$
Initialize $V \leftarrow 2, P \leftarrow T^{-1} + 2s, W \leftarrow P$
for $b = \lfloor \log_2 N \rfloor$ **down to** 2 **do** ▷ *Algorithm 1 with termination check*
    if bit b of N is 0 **then** $(V, W) \leftarrow (V^2 - 2, VW - P)$
    else $(V, W) \leftarrow (VW - P, W^2 - 2)$
    if $b = \ell + 1$ **and** $V = \pm W$ **then return** `prime?`
    else if $b \leq \ell$ **and** $W = 0$ **then return** `prime?`

return `composite`

– Whenever T is a power of 2, we replace the redundant base-T strong Fermat test with a base-5 strong Fermat test. BFW only performs effectively an Euler test for $D = 5$, but not for e.g. $D = -7$ in which case $T = 2$. This change guarantees that exactly two independent strong Fermat-Euler tests are run.

We note that this Lucas test is very similar to `Plucky` with $x = -D$ and $a = 1$.

5 Conclusion and future work

We have shown how to simplify the Kim et al. variant of Pocklington-Peralta, and how to batch it and/or combine it with Müller's algorithm. The new algorithm can be quite simple, and in some cases it may outperform Müller's algorithm.

There is some possibility of future improvement. It would be nice to skip part of the Lucas powering steps, as is possible in Müller's algorithm, without explicitly branching to Müller's algorithm. Another observation is that when `Plucky` fails, it gives a square root of either $a^2 + x$ or $x/(a^2 + x)$, and perhaps this could be useful, either within the square root algorithm itself (e.g. to somehow obtain an algorithm that runs in deterministically polynomial time if the generalized Riemann hypothesis holds) or for hashing to elliptic curves, along the lines of [Kos2024].

We have also shown how to speed up and strengthen Baillie-PSW and Baillie-Fiori-Wagstaff. The same techniques might apply to give a faster or simpler-to-implement variant of other primality tests such as the Simple Quadratic Frobenius Test [Sey2005].

5.1 AI usage disclosure

This version corrects several small mistakes found by Anthropic's Claude AI in September 2026. Claude also suggested the possible performance improvement in Footnote 4.

5.2 Intellectual property disclosure

Some of these techniques may be covered by US and/or international patents.

References

[Adj2013] Gora Adj and Francisco Rodríguez-Henríquez. "Square root computation over even extension fields". In: *IEEE Transactions on Computers* 63.11 (2013), pp. 2829–2841.

[Alb2018] Martin R Albrecht, Jake Massimo, Kenneth G Paterson, and Juraj Somorovsky. "Prime and prejudice: primality testing under adversarial conditions". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 281–298.

[Ara2023] Diego F Aranha, Benjamin Salling Hvass, Bas Spitters, and Mehdi Tibouchi. "Faster constant-time evaluation of the Kronecker symbol with application to elliptic curve hashing". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2023, pp. 3228–3238.

[Art1966] MM Artjuhov. "Certain criteria for primality of numbers connected with the little Fermat theorem". In: *Acta Arith* 12.355-364 (1966), p. 67.

[Atk1992] A Oliver L Atkin. "Probabilistic primality testing". In: *INRIA Res. Rep* (1992), pp. 159–163.

[Bai1980] Robert Baillie and Samuel S Wagstaff. "Lucas pseudoprimes". In: *Mathematics of Computation* 35.152 (1980), pp. 1391–1417.

[Bai2021] Robert Baillie, Andrew Fiori, and Samuel Wagstaff Jr. "Strengthening the Baillie-PSW primality test". In: *Mathematics of Computation* 90.330 (2021), pp. 1931–1955.

[Ber2001] Daniel J Bernstein. "Faster square roots in annoying finite fields". In: *preprint* (2001).

[Ber2012] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *Journal of cryptographic engineering* 2.2 (2012), pp. 77–89.

[Ber2019] Daniel J Bernstein and Bo-Yin Yang. "Fast constant-time gcd computation and modular inversion". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), pp. 340–398.

[Cip1903] Michele Cipolla. "Un metodo per la risolutione della congruenza di secondo grado". In: *Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche, Napoli* 9 (1903), pp. 154–163.

[Fis2015] Daniel Fischer. *Is a number n known, which is not squarefree, but a strong fermat-pseudoprime to base* 2 *and* 3*?* Mathematics Stack Exchange. 2015. eprint: `https://math.stackexchange.com/q/1495451`.

[Ham2014] Mike Hamburg. *Efficient way to determine if a number is Perfect Square*. Mathematics Stack Exchange. `https://math.stackexchange.com/q/878338`. 2014.

[Ham2015] Mike Hamburg. "Decaf: Eliminating cofactors through point compression". In: *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I 35*. Springer. 2015, pp. 705–723.

[Ham2021] Mike Hamburg. "Computing the Jacobi symbol using Bernstein-Yang". In: *Cryptology ePrint Archive* (2021).

[Kim2022] Chang Heon Kim, Namhun Koo, and Soonhak Kwon. "On the Pocklington-Peralta square root algorithm in finite fields". In: *Bulletin of the Korean Mathematical Society* 59.6 (2022), pp. 1523–1537.

[Kos2024] Dmitrii Koshelev. "Hashing to elliptic curves through Cipolla–Lehmer–Müller's square root algorithm". In: *Journal of Cryptology* 37.2 (2024), p. 11.

[Lam2016] Youness Lamzouri, Xiannan Li, and Kannan Soundararajan. *Conditional bounds for the least quadratic non-residue and related problems*. 2016. arXiv: `1309.3595 [math.NT]`.

[Leh1969] Derrick H Lehmer. "Computer technology applied to the theory of numbers". In: *Studies in number theory* (1969).

[Mas2020] Jake Massimo. "An Analysis of Primality Testing and Its Use in Cryptographic Applications". PhD thesis. Royal Holloway, University of London, 2020.

[Mil1975] Gary L Miller. "Riemann's hypothesis and tests for primality". In: *Proceedings of the seventh annual ACM symposium on Theory of computing*. 1975, pp. 234–239.

[Mül2004] Siguna Müller. "On the computation of square roots in finite fields". In: *Designs, Codes and Cryptography* 31 (2004), pp. 301–312.

[Per1986] Rene Peralta. "A simple and fast probabilistic algorithm for computing square roots modulo a prime number (Corresp.)" In: *IEEE transactions on information theory* 32.6 (1986), pp. 846–847.

[Poc1917] Henry C Pocklington. "The direct solution of the quadratic and cubic binomial congruences with prime moduli". In: *Proceedings of the Cambridge Philosophical Society*. Vol. 19. 1917, pp. 57–59.

[Pom1980] Carl Pomerance, John L Selfridge, and Samuel S Wagstaff. "The pseudoprimes to $25 \cdot 10^9$". In: *Mathematics of Computation* 35.151 (1980), pp. 1003–1026.

[Por2023] Thomas Pornin. "Optimized Discrete Logarithm Computation for Faster Square Roots in Finite Fields". In: *Cryptology ePrint Archive* (2023).

[Rab1980] Michael O Rabin. "Probabilistic algorithm for testing primality". In: *Journal of number theory* 12.1 (1980), pp. 128–138.

[Sar2020] Palash Sarkar. "Computing square roots faster than the Tonelli-Shanks / Bernstein algorithm". In: *Cryptology ePrint Archive* (2020).

[Sey2005] Martin Seysen. *A Simplified Quadratic Frobenius Primality Test*. Cryptology ePrint Archive, Paper 2005/462. 2005.

[Sha1973] Daniel Shanks. "Five number-theoretic algorithms". In: *Proceedings of the Second Manitoba Conference on Numerical Mathematics (Winnipeg), 1973*. 1973.

[Sol1977] Robert Solovay and Volker Strassen. "A fast Monte-Carlo test for primality". In: *SIAM journal on Computing* 6.1 (1977), pp. 84–85.

[Sta2023] National Institute of Standards and Technology (NIST). "Digital signature standard (DSS)". In: (2023).

[Sut2011] Andrew Sutherland. "Structure computation and discrete logarithms in finite abelian p-groups". In: *Mathematics of Computation* 80.273 (2011), pp. 477–500.

[Sze2011] Tsz-Wo Sze. "On taking square roots without quadratic nonresidues over finite fields". In: *Mathematics of Computation* 80.275 (2011), pp. 1797–1811.

[Ton1891] Alberto Tonelli. "Bemerkung über die Auflösung quadratischer Congruenzen". In: *Nachrichten von der Königl. Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen* 1891 (1891), pp. 344–346.

[Und2017] Paul Underwood. *Quadratic Frobenius probable prime tests costing two selfridges*. 2017. arXiv: `1706.01265 [math.NT]`.

[Wan2023] Almas Wang. *Gauss-Euler Primality Test*. 2023. arXiv: `2311.07048 [math.NT]`.

A Variants on `Plucky`

It is worth mentioning several simple variants of `Plucky`, which may be preferable depending on implementation constraints:

- We can check the success condition $V \stackrel{?}{=} 0$ only on iterations with $i \leq \nu(q-1)$, where it is more likely to obtain.
- We can check the retry condition $V \stackrel{?}{=} \pm 2$ at iteration $\nu(q-1)$, or any later iteration, instead of at the end of the loop.
- By eliminating common subexpressions, or unrolling the first or last few iterations of the loop, we can save a few multiplications.
- We can re-express the conditional branch in the Lucas ladder in terms of conditional swaps.
- We can use the inverse construction $1 + a^2x$ instead of $a^2 + x$. This explicitly blinds x, which could be useful for side-channel reasons, and is more similar to Müller's algorithm. Likewise we could use $a^2 + b^2x$.
- If an inverse square root is desired, then instead of $(4a)^{-1}$ we would calculate $(4ax)^{-1}$, and we would also reject $1/\sqrt{0}$.
- Instead of explicitly avoiding $a = 0$ or retrying when $a = \sqrt{-x}$, we could take care that these reach the retry case naturally, or (as in Kim et al.) return $\sqrt{x} = a\sqrt{-1}$ if $\sqrt{-1}$ is known.

B When $q \equiv 3 \pmod 4$

`Plucky` and Müller's algorithm assume that $q \equiv 1 \pmod 4$. Of course when $q \equiv 3 \pmod 4$ we can just compute $\sqrt{x} = x^{(q+1)/4}$. But for completeness, could we extend these algorithms to cover $q \equiv 3 \pmod 4$? For `Plucky`, when x is a nonzero square then $-x$ is not square, and the discriminant $P^2 - 4 = -16a^2x/(a^2 + x)^2$ is a quadratic nonresidue. Therefore the order of the sequence divides $q + 1$ rather than $q - 1$. Likewise for Müller's algorithm we would choose $\chi(D) = +1$, so that the order of the sequence divides $(q-1)/2$. Otherwise the same analysis holds: we will get $\sqrt{2+P}$, $\sqrt{2-P}$ or $\sqrt{-D}$, and in the Müller case it must be $\sqrt{2+P}$ because the other options aren't square. We can also avoid changing

the exponent[5] since Algorithm 1 tracks $(V, W) = (V_n(P, 1), V_{n+1}(P, 1))$. The application of this to Müller's algorithm is shown in Algorithm 8.

Algorithm 8 Variant of Müller's algorithm which works for any odd q, and can be run in nearly-secret-invariant time for nonzero x.

Require: Value $x \in \mathbb{F}_q$ with q odd and $q > 3$.
Ensure: Return $y = \pm\sqrt{x}$ if it exists, or otherwise an error.

if $x = 0$ **then return** $y \leftarrow 0$
repeat
  Randomly sample $a \xleftarrow{R} \mathbb{F}_q \backslash \{0\}$
  Compute a^{-1} and $\chi(4 - a^2 x)$ in a batch
until $\chi(4 - a^2 x) = -1$
Use Algorithm 1 to compute $V \leftarrow V_{\lfloor q/4 \rfloor}(a^2 x - 2, 1)$
if $V^2 = a^2 \cdot x$ **then return** $y \leftarrow V \cdot a^{-1}$
else raise error: x isn't square!

If in some implementation calculating χ is expensive, or leaks q through side-channels, then it may be preferable to skip the χ check, as shown in Algorithm 9.[6]

Algorithm 9 Variant of Müller's algorithm which works for any odd q, runs in nearly-secret-invariant time for nonzero x, and skips χ.

Require: Value $x \in \mathbb{F}_q$ with q odd.
Ensure: Return $y = \pm\sqrt{x}$ if it exists, or otherwise an error.

if $x = 0$ **then return** $y \leftarrow 0$
Randomly sample $a \xleftarrow{R} \mathbb{F}_q \backslash \{0\}$
Use Algorithm 1 to compute $V \leftarrow V_{\lfloor (q+1)/4 \rfloor}(a^2 x - 2, 1)$
if $V^2 = a^2 \cdot x$ **then return** $y \leftarrow V \cdot a^{-1}$
else if $V = \pm 2$ **then go to line** 2
else raise error: x isn't square! ▷ *Here* $V \in \{0, \pm\sqrt{2 - P}\}$

A very similar approach works for `Plucky`, but there isn't much point, since `Plucky`'s advantages over Müller are a lower iteration count when $\nu(q - 1)$ is large, and not having to calculate χ. When q is not secret, we might as well just return $x^{(q+1)/4}$ for 3-mod-4 q, and when q is secret we must reduce the success probability to 1/2 to avoid leaking $\nu(q - 1)$ (by only checking the last iteration for $V \stackrel{?}{=} 0$), and we have just shown how to skip χ.

[5] This works in Algorithm 9 as well, with an extra conditional swap to get $V_{\lfloor (q+1)/4 \rfloor} = W$ when $q \equiv 3 \bmod 4$.

[6] We could also run this algorithm in cases where q isn't secret, but we just want to skip χ. In that case, we could also try to extract the square root from $V_{(q \pm 1)/2^{\nu(q \pm 1)}}$, but this succeeds with probability only 2^{-k}. And if k is small and q isn't secret then we would rather use Tonelli-Shanks anyway.

C Proof of the improved BFW test

Here we prove our propositions about `Plucky` and `SuperBFPSW`.

Lemma 1 (How V sequences reach 2). *Let R be a commutative ring with unit. Take any $P \in R$ and $n \in \mathbb{N}$, and let $(V, W) := (V_n(P,1), V_{n+1}(P,1))$. We have $V_{2n+1}(P,1) = VW - P$. If $V_{2n+1}(P,1) = \pm 2$, then*

$$(W \mp V)^2 = 0 \qquad \text{and} \qquad (W \pm V)^2 = 4 \cdot (2 \pm P)$$

where all the $\pm$ terms have the same sign as the ± 2 in the assumption. If R is a field, or if $R = \mathbb{Z}/N\mathbb{Z}$ where N is square-free, it follows that $V = \pm W$ and $V^2 = W^2 = 2 \pm P$.

Conversely, if $V = \pm W$ and if $P \mp 2$ is invertible in R, then $V^2 = 2 \pm P$ and $V_{2n+1}(P,1) = \pm 2$.

Proof. If $V_{2n+1}(P,1) \equiv VW - P = \pm 2$ then $VW = P \pm 2$. Setting $Q = 1$ and combining with (3), we calculate:

$$\begin{aligned} W^2 - PVW + V^2 &= 4 - P^2 \\ W^2 \pm 2VW + V^2 &= 4 - P^2 + (P \pm 2)VW \\ &= 4 - P^2 + (P \pm 2)(P \pm 2) \\ (W \pm V)^2 &= 8 \pm 4P \end{aligned}$$

Likewise:

$$(W \mp V)^2 = W^2 \mp 2VW + V^2 = 4 - P^2 + (P \mp 2)(P \pm 2) = 0$$

For the converse case where $V = \pm W$ and $2 \mp P$ is invertible, we proceed similarly:

$$\begin{aligned} W^2 - PVW + V^2 &= 4 - P^2 \\ V^2 \mp PV^2 + V^2 &= 4 - P^2 \\ V^2(2 \mp P) &= 4 - P^2 \\ V^2 &= 2 \pm P \end{aligned}$$

Thus $VW = \pm V^2 = P \pm 2$, so $V_{2n+1}(P,1) = \pm 2$, completing the proof.

We note that the conclusions $V = \pm W$ and $V^2 = W^2 = 2 \pm P$ do not hold mod non-square-free N. For example, with $P = 4 \in \mathbb{Z}/25$, the V_n sequence is $[\![2, 4, 14, 2, \ldots]\!]$, where $(4-14)^2 \equiv 0$ but $4 \not\equiv 14$, and likewise $V^2 = 16, W^2 = 21$ but $2 + P = 6$.

Proposition 1 (Method $\mathbf{A}^p$). *Let $N \geq 13$ be odd, and define Method A^p to check, for each $D \in \{5, -7, 9, -11, 13, -15, \ldots\}$ such that either $D = -15$ or $|D|$ is prime, whether $\left(\frac{D}{N}\right) = -1$. If so, it returns D. If instead $\left(\frac{D}{N}\right) = 0$ it declares*

that N is composite. By quadratic reciprocity we can rewrite $\left(\frac{D}{N}\right) = \left(\frac{N}{|D|}\right)$, which allows us to work with positive numbers.

When Method A^* succeeds and outputs (D, P, Q), then N is coprime to $2PQD$. If N is divisible by 3, it is possible that Method A^* declares N composite while Method A^p returns D such that $|(1-D)/4| = 3$. In all other cases, the two routines return the same D.

Proof. To prove that it is acceptable to reject unconditionally when $\left(\frac{D}{N}\right) = 0$, it suffices to show that if $N \geq 13$ is prime, then there is a QNR on the list $L := \{5, -7, 9, \ldots, \pm(N-2)\}$ which is $\pm$ prime or -15. This will have Jacobi symbol -1 and will be returned. This is true by calculation for $N = 13$ (where $D = 5$ is a QNR), so we can bump the bound to $N \geq 17$. Then it also suffices if -3 is a QNR: once $N \geq 17$, we have 5 and $-15 \in L$, so if -3 is a QNR then one of these is as well. Let $L' := \{-3\} \cup L$.

We decompose by cases: if $N \equiv 7 \bmod 8$, then $N - 2 \in L'$, and is a QNR. If $N \equiv 3 \bmod 8$ then 2 is a QNR; it follows that either -3 or $-(N+3)/2$ is a QNR in L', since their product $(3N+9)/2 \equiv 9/2$ is a QNR. If $N \equiv 1 \bmod 4$, then $\left(\frac{-D}{N}\right) = \left(\frac{D}{N}\right)$, and also $2 \equiv -(N-2) \in L'$. So it suffices if any $D \in \{2, 3, 5, 7, 9, 11, \ldots, N-2\}$ is a QNR. But they cannot all be QRs, because then all elements in $\{1, \ldots, N-1\}$ (which are products of these numbers) would be QRs, which is impossible.

To confine D to $\pm$prime or -15, consider that each composite $D \in L'$ is the product of two smaller values $(a, b) \in L'$, and the Jacobi symbol is multiplicative. So if $\left(\frac{D}{N}\right) = -1$ then either $\left(\frac{a}{N}\right) = -1$ or $\left(\frac{b}{N}\right) = -1$. This means that we will not reach D since a or b would be returned first, unless one of them is -3. The multiples of -3 in L' are $\{-3, 9, -15, 21, \ldots\}$. The values -3 and 9 cannot be returned, and if $\left(\frac{-3}{N}\right) = -1$ then Method A^p will terminate at $D = -15$ at the latest.

We note that when $N \geq 13$, then changing from a conditional rejection to an unconditional rejection does not change the outcome of Method A*: we have just shown that prime N will not be rejected, and if composite N is affected by the change, it would be rejected anyway. This is because the first time D is tested where $\gcd(D, N) > 1$, we have either $|D|$ prime or $D = 9$. If $|D|$ is prime and N is composite then $N \nmid D$, so it is rejected immediately. If $D = 9$ then either $N \nmid D$, or $N = 3$ (which is prime and doesn't reach $D = 9$ anyway), or $N = 9$ which is rejected for being square. However, testing only primes and -15 can change the outcome: if $3|N$ then skipping $D = 9$ could lead to -11 or 13 being returned instead of a rejection. In this case, $Q = \pm 3$, so as long as we test that Q is coprime to N (which we do, via the base-Q strong Fermat test), the results will be the same as with Method A$*$.

Method A* always returns (D, P, Q) which are coprime to N. This is obvious for D, since $\left(\frac{D}{N}\right) = -1$. For P, we have $P = 1$ unless $D = P = Q = 5 \perp N$. For Q, factor $|Q|$ into powers of 2, 3 and other primes. 2 is coprime to our odd N, 3 cannot appear until $D = 9$ has been tested, and any higher prime p cannot appear until $\pm p$ has been tested.

Proposition 2 (Accelerated strong Lucas test). *Suppose N is coprime to $2PQD$. Factor $N+1 =: (2m+1)\cdot 2^\ell$, and let*

$$\hat{V}_n := V_n(P^2/Q - 2, 1).$$

Then N passes the strong Lucas test base (P,Q) if and only if either $\hat{V}_{m+1} \equiv \pm\hat{V}_m$, or if some $\hat{V}_{(2m+1)\cdot 2^k} \equiv 0$ where $0 \le k \le \ell - 2$.

Proof. Set $\hat{P} :\equiv P^2/Q - 2$ and discriminant $\hat{D} :\equiv P^2D/Q^2$. Lucas sequences satisfy the identities

$$V_{mn}(P,Q) = V_n(V_m(P), Q^m)$$
$$V_n(\lambda P, \lambda^2 Q) = \lambda^n \cdot V_n(P,Q)$$

from which it follows that

$$V_{2n}(P,Q) = Q^n \cdot \hat{V}_n.$$

We therefore have

$$DU_{2n+1}(P,Q) = V_{2n+2}(P,Q) - QV_{2n}(P,Q) = Q^{n+1}(\hat{V}_{n+1} - \hat{V}_n)$$
$$PV_{2n+1}(P,Q) = V_{2n+2}(P,Q) + QV_{2n}(P,Q) = Q^{n+1}(\hat{V}_{n+1} + \hat{V}_n)$$

Since P, Q and D are invertible, this simplifies:

$$\begin{aligned} U_{2n+1}(P,Q) \equiv 0 \quad &\Leftrightarrow \quad \hat{V}_{n+1} \equiv +\hat{V}_n \\ V_{2n+1}(P,Q) \equiv 0 \quad &\Leftrightarrow \quad \hat{V}_{n+1} \equiv -\hat{V}_n \\ V_{2n}(P,Q) \equiv 0 \quad &\Leftrightarrow \quad \hat{V}_n \equiv 0 \end{aligned}$$

Replacing the three conditions on the left with the three on the right gives us the claimed test.

Before analyzing the V-Lucas test, we will pause for a lemma about symbol estimation.

Lemma 2 (Symbol estimators). *Let $N \ge 3$ be an odd integer, and let x be in $\mathbb{Z}/N$. Define the Jacobi symbol and two estimators mod N:*

- *Let $\left(\frac{x}{N}\right)$ be the Jacobi symbol of x on N.*
- *Let $\chi_{\mathrm{Eul},N}(x) :\equiv x^{(N-1)/2}$ if the latter is ± 1, and otherwise 0.*
- *Define $\chi_{\mathrm{Luc},N}(x)$ as follows. Let $\delta := \left(\frac{x(x-4)}{N}\right)$. If $\delta \neq 0$, then set $\chi_{\mathrm{Luc},N}(x) :\equiv \frac{1}{2}V_{(N-\delta)/2}(x-2,1)$ if this is ± 1. In the special case $x \equiv 4$, define $\chi_{\mathrm{Luc},N}(x) := +1$. In all other cases define $\chi_{\mathrm{Luc},N}(x) := 0$.*

We prove that these estimators are accurate in several cases:

- *When N is prime, $\left(\frac{x}{N}\right) = \chi_{\mathrm{Eul},N}(x) = \chi_{\mathrm{Luc},N}(x)$.*

- *When $N \equiv 3$ (mod 4) and $\chi_{\mathrm{Eul},N}(x) \neq 0$ then $\left(\frac{x}{N}\right) = \chi_{\mathrm{Eul},N}(x)$.*
- *When there is any $y \in \mathbb{Z}/N$ such that $\chi_{\mathrm{Eul},N}(y) = -1$, then for any x with $\chi_{\mathrm{Eul},N}(x) \neq 0$ we also have $\left(\frac{x}{N}\right) = \chi_{\mathrm{Eul},N}(x)$.*
- *When $N \equiv -\delta$ (mod 4) and $\chi_{\mathrm{Luc},N}(x) \neq 0$ then $\left(\frac{x}{N}\right) = \chi_{\mathrm{Luc},N}(x)$.*
- *When $\chi_{\mathrm{Eul},N}(x) = -1$ or $\chi_{\mathrm{Luc},N}(x) = -1$, then $\left(\frac{x}{N}\right) = -1$.*
- *When N passes a base-x strong Fermat test, $\chi_{\mathrm{Eul},N}(x) = \left(\frac{x}{N}\right)$ (this case is shown in [Pom1980, Theorem 3]).*
- *When for some $1 \leq k < \nu(N-\delta)$ we have $V_{(N-\delta)/2^k}(x-2,1) = -2$, or if $\delta = -1$ and N passes a base-$x-2$ accelerated strong Lucas test, then $\left(\frac{x}{N}\right) = \chi_{\mathrm{Luc},N}(x)$.*

Proof. First, we will deal with the case that N is prime. Euler proved that for prime N, the Euler and Legendre (and thus Jacobi) symbols are the same. For the Lucas part, the cases where $x = 4$ or $\delta = 0$ are straightforward. Otherwise we know that if $P + 2 = x$ is square mod N, then $V_n(P,1)$ mod N has period dividing $(N-\delta)/2$, so that $V_{(N-\delta)/2} = +2$; and otherwise its period divides $N-\delta$ but not $(N-\delta)/2$, in which case the V sequence still satisfies $V^2_{(N-\delta)/2} - 2 = 2$, so $V_{(N-\delta)/2} = -2$ as desired.

For the case when $N \equiv 3$ (mod 4), we know that if $\chi_{\mathrm{Eul},N}(x) = \pm 1$ then x is coprime to N. Since the Jacobi symbol is multiplicative, we know that

$$\chi_{\mathrm{Eul},N}(x) := x^{(N-1)/2} = \left(x^{(N-3)/4}\right)^2 \cdot x$$

must have the same Jacobi symbol as x. For $N \equiv 3$ (mod 4), the Jacobi symbols of each $y \in \{-1, 0, 1\}$ are equal to y, so $\chi_{\mathrm{Eul},N}(x)$ having the same Jacobi symbol as x means that it equals $\left(\frac{x}{N}\right)$, as claimed.

For the case when $N \equiv -\delta$ (mod 4), let $P = x - 2$. We know that if $\chi_{\mathrm{Luc},N}(x) \neq 0$ then either $x = 4$ (in which case the lemma is trivial) or D is invertible, which implies that x is invertible. We observe that the sequence index $(N-\delta)/2$ is odd, so that $V_{(N-\delta)/2} = \pm 2 = VW - P$ for earlier sequence elements

$$(V, W) = \left(V_{(N-\delta-2)/4}(P,1),\ V_{(N-\delta+2)/4}(P,1)\right).$$

Applying Lemma 1 we have $(W \pm V)^2 = 4(2 \pm P)$. If $\chi_{\mathrm{Luc},N}(x) = +1$ case this is $4x$, meaning that x is square and invertible so its Jacobi symbol is also $+1$. Likewise if $\chi_{\mathrm{Luc},N}(x) = -1$ then $4(2 \pm P) = -4D/x$; we also know that $\left(\frac{-D}{N}\right) = \left(\frac{-1}{N}\right)\left(\frac{D}{N}\right) = -1$, so that x has Jacobi symbol -1.

For the case when $\chi_{\mathrm{Eul},N}(x) = -1$, recall the notation $\nu(m)$ for a positive integer m, meaning the largest integer such that $2^{\nu(m)}$ divides m. We factor $N =: \prod p_i^{e_i}$ and consider that $x^{(N-1)/2} = -1$ must hold modulo each p_i. If $\nu(p_i - 1) < \nu(N-1)$ this is impossible; if $\nu(p_i - 1) = \nu(N-1)$ this means $\left(\frac{x}{p_i}\right) = -1$; and if $\nu(p_i - 1) > \nu(N-1)$ this means $\left(\frac{x}{p_i}\right) = +1$. Modulo

$2^{\nu(N-1)+1}$ we have

$$\begin{aligned} 2^{\nu(N-1)}+1 \equiv N &= \prod((p_i-1)+1)^{e_i} \\ &\equiv \prod_{i:\nu(p_i-1)=\nu(N-1)} (2^{\nu(N-1)}+1)^{e_i} \cdot \prod_{i:\nu(p_i-1)>\nu(N-1)} (0+1)^{e_i} \\ &\equiv 1 + 2^{\nu(N-1)} \cdot \sum_{i:\nu(p_i-1)=\nu(N-1)} e_i \end{aligned}$$

so this sum must be odd. Since these p_i are exactly the prime-power factors where $\left(\frac{x}{p_i}\right) = -1$, it follows that

$$\left(\frac{x}{N}\right) = \prod_i \left(\frac{x}{p_i}\right)^{e_i} = \prod_{i:\nu(p_i-1)=\nu(N-1)} (-1)^{e_i} \cdot \prod_{i:\nu(p_i-1)>\nu(N-1)} (+1)^{e_i} = -1.$$

The case where some $\chi_{\mathrm{Eul},N}(y) = -1$ follows, because $\chi_{\mathrm{Eul},N}(\cdot)$ is multiplicative when it's nonzero, and the Jacobi symbol is also multiplicative. More explicitly, we have just shown that $\left(\frac{y}{N}\right) = -1$. For x where $\chi_{\mathrm{Eul},N}(x) = -1$, we've also just shown that $\left(\frac{x}{N}\right) = -1$. For x where $\chi_{\mathrm{Eul},N}(x) = +1$, we have

$$\chi_{\mathrm{Eul},N}(xy) = -1 \;\text{ so }\; \left(\frac{xy}{N}\right) = -1$$

so that $\left(\frac{x}{N}\right) = +1$.

Finally, suppose that $V_{(N-\delta)/2^k}(x-2,1) = -2$, where $1 \le k < \nu(N-\delta)$ so that $(N-\delta)/2^k$ is still even. Since the sequence has reached ± 2 it will stay there, so certainly $\chi_{\mathrm{Luc},N}(x) = \pm 1$. Let $\epsilon := 2^{\nu(N-\delta)-k+1} \ge 4$. We know if $V_m(P,1) = -2$ in some field, and the sequence has period λ, then $\nu(\lambda) = \nu(m)+1$: this is because certainly $V_{2m}(P,1) = +2$ so that $\lambda | 2m$, but we cannot have $\lambda | m$ or else $V_m(P,1)$ would be $+2$ instead of -2.

Now let us analyze the period λ_i of $V(P,1)$ modulo each prime p_i dividing N; of course $\mathbb{Z}/p_i$ is a field. Since $V_{(N-\delta)/2^k}(P,1) = -2 \bmod N$ and thus mod p_i, we know that

$$\nu(\lambda_i) = \nu((N-\delta)/2^k) + 1 = \nu(N-\delta) - k + 1.$$

We also know that $\lambda_i | p_i - \left(\frac{D}{p_i}\right)$, and thus $\nu(\lambda_i) \le \nu\left(p_i - \left(\frac{D}{p_i}\right)\right)$, and that equality holds iff $\left(\frac{x}{p_i}\right) = -1$. It follows that

$$p_i \equiv \begin{cases} \epsilon \pm 1 \text{ if } \left(\frac{x}{p_i}\right) = -1 \\ \pm 1 \text{ otherwise} \end{cases} \mod 2\epsilon.$$

We can now calculate modulo 2ϵ:

$$\begin{aligned} N &= \prod_{\left(\frac{x}{p_i}\right)=-1} p_i^{e_i} \cdot \prod_{\left(\frac{x}{p_i}\right)=+1} {p_i}^{e_i} \\ &\equiv \prod_{\left(\frac{x}{p_i}\right)=-1} (\epsilon \;\pm_i\; 1)^{e_i} \cdot \prod_{\left(\frac{x}{p_i}\right)=+1} (\pm_i\, 1)^{e_i} \end{aligned}$$

Here the notation $\pm_i$ indicates that the sign may be different for each i. Since $\epsilon \geq 4$, the multiplicative group $\{1, -1, \epsilon + 1, \epsilon - 1\}$ mod 2ϵ is isomorphic to $Z_2 \times Z_2$, so the sum of e_i in the left product is odd iff $N \equiv \epsilon \pm 1 \bmod 2\epsilon$; this happens exactly if $k = 1$, which is also exactly the case when $\chi_{\mathrm{Luc},N}(x) = -1$. And indeed, the sum of e_i in the left product is odd iff $\left(\frac{x}{N}\right) = \prod \left(\frac{x}{p_i}\right)^{e_i} = -1$.

The case where $\chi_{\mathrm{Luc},N}(x) = -1$ follows, because in that case either $N \equiv -\delta$ in which case we have already shown that the symbols match, or $N \equiv +\delta$ in which case $(N - \delta)/2$ is even.

The case where $\delta = -1$ and N passes a base-$x - 2$ accelerated strong Lucas test also follows. To pass the strong Lucas test, either $V_{(N-\delta)/2^k}(x - 2, 1) = 0$ with $k \geq 2$, or $V_{m+1}(x-2, 1) = V_m(x-2, 1)$ where $N - \delta = 2^k \cdot (2m+1)$. In the first case we have $V_{(N-\delta)/2^{k-1}}(x - 2) = -2$, in which case we have just shown that the symbols match.

In the second case, according to Lemma 1 we have $V_m(x-2)^2 = 2 \pm (x-2)$ with the same sign. In the + case, clearly x is square and we also have $V_{2m+1} = +2$, so the sequence stays at $+2$ and the symbols match. In the $-$ case, instead $V_{2m+1} = -2$. Furthermore $4 - x = -D/x$ is square, so $\left(\frac{x}{N}\right) = -\left(\frac{-1}{N}\right)$. If $N \equiv 3 \pmod 4$, then $N - \delta$ is a multiple of 4, so there is at least one doubling step and the sequence reaches $+2$, and indeed $\left(\frac{x}{N}\right) = +1$. On the other hand if $N \equiv 1 \pmod 4$, then $(N - \delta)/2 = 2m + 1$, so both symbols are -1.

This completes the proof.

We are finally ready to analyze the strong and V-Lucas tests.

Proposition 3 (Euler-Lucas test instead of V-Lucas test). *Suppose $2PQD \perp N$ and that N passes the accelerated strong Lucas test base (P, Q). Then $\hat{V}_{(N+1)/2} \equiv \pm 2$ and*

$$V_{N+1}(P, Q) \equiv Q \cdot Q^{(N-1)/2} \cdot \hat{V}_{(N+1)/2}.$$

So once the strong Lucas test passes, the V-Lucas test $V_{N+1} \stackrel{?}{=} 2Q$ also passes if and only if they pass the Euler-Lucas test

$$Q^{(N-1)/2} \stackrel{?}{\equiv} \frac{1}{2}\hat{V}_{(N+1)/2}.$$

Proof. The first equality is true because for all positive integers n,

$$\begin{aligned} V_{2n}(P, Q) &= V_n(V_2(P, Q), Q^2) \\ &= V_n(P^2 - 2Q, Q^2) \\ &= Q^n \cdot V_n(P^2/Q - 2, 1) \\ &=: Q^n \cdot \hat{V}_n \end{aligned}$$

Plugging in $n = (N + 1)/2$ gives the claimed equality.

Now the accelerated strong Lucas test passes in one of two ways. Either $\hat{V}_{m+1} \equiv \pm\hat{V}_m$ where $2m + 1$ is the odd part of $N + 1$; then by Lemma 1 we have $\hat{V}_{2m+1} \equiv \pm 2$, and $\hat{V}_{(2m+1)\cdot 2^k} \equiv +2$. Or, $\hat{V}_{(2m+1)\cdot 2^k} \equiv 0$, in which case

$\hat{V}_{(2m+1)\cdot 2^{k+1}} \equiv -2$ and $\hat{V}_{(2m+1)\cdot 2^{k'}} \equiv +2$ for $k' \geq k+2$. Since $(N+1)/2 = (2m+1)\cdot 2^{k'}$ where $k' \geq k+1$, in either case $\hat{V}_{(N+1)/2} \equiv \pm 2$.

The V-Lucas test requires that $V_{N+1}(P,Q) = 2Q$. Dividing through by $2Q$ (which we assumed is coprime to N) gives $Q^{(N-1)/2} \cdot \frac{1}{2}\hat{V}_{(N+1)/2} \equiv 1$. Since $\frac{1}{2}\hat{V}_{(N+1)/2} \equiv \pm 1$, it is its own inverse, so that $Q^{(N-1)/2} \equiv \frac{1}{2}\hat{V}_{(N+1)/2}$.

Proposition 5 (Special case of $D = 5$). *Suppose N is coprime to $2PQD$ and $\left(\frac{5}{N}\right) = -1$. The accelerated strong Lucas test and Euler-Lucas test pass on $(P,Q) = (5,5)$ if and only if they pass on $(P,Q) = (1,-1)$ and also $5^{(N-1)/2} \equiv -1$. This also means that Method A from [Bai1980] can be used instead of Method A*, if this extra check is used.*

Proof. Changing (P,Q) from $(5,5)$ back to $(1,-1)$ changes $P^2/Q - 2$ from 3 to -3, which changes the sign of odd terms of $V_n(P^2/Q-2,1)$. This doesn't affect the simplified accelerated strong Lucas test, which only checks V against 0 and ± 2. If this test passes, it guarantees that $V_{(N+1)/2}(P^2/Q-2,1) = \pm 2$. Now, the simplified V-Lucas test on $(5,5)$ tests:

$$5^{(N-1)/2} \stackrel{?}{=} \frac{1}{2}V_{(N+1)/2}(3,1)$$

and on $(1,-1)$ it tests:

$$(-1)^{(N-1)/2} \stackrel{?}{=} \frac{1}{2}V_{(N+1)/2}(-3,1) = (-1)^{(N+1)/2} \cdot \frac{1}{2}V_{(N+1)/2}(3,1)$$

which is equivalent to

$$\frac{1}{2}V_{(N+1)/2}(3,1) = -1.$$

Thus if $5^{(N-1)/2} = -1$ the tests are equivalent, proving the "if" case. For the "only if", if the strong Lucas test and V-Lucas test pass on $(5,5)$ then

$$5^{(N-1)/2} = \frac{1}{2}V_{(N+1)/2}(3,1) = \pm 1,$$

so it remains only to show that this value is indeed -1 and not $+1$. We have already assumed that $\left(\frac{5}{N}\right) = -1$, which is enough: according to Lemma 2, if $N \equiv 3 \bmod 4$ then $5^{(N-1)/2}$ (being in $\{-1,1\}$) must match $\left(\frac{5}{N}\right)$, and likewise if $N \equiv 1 \equiv -\left(\frac{D}{N}\right) \bmod 4$ then $\frac{1}{2}V_{(N+1)/2}(3,1)$ (being in $\{-1,1\}$) must match $\left(\frac{5}{N}\right)$. So if these are equal and are ± 1 then they must be -1 as claimed.

D Quadratic Hensel lifting for integer square roots

For Baillie-PSW's Method A*, we may need to determine whether an odd positive integer N is square, which can be done easily by attempting to calculate the square root. Square roots can be efficiently calculated with the Newton-Raphson method, but Hensel lifting is easier. Linear Hensel lifting solves one bit at a time,

but on typical hardware it is faster to use quadratic lifting, which doubles the number of correct bits in every iteration. Quadratic Hensel lifting uses the same recurrence as Newton-Raphson, but starting with the least significant bits instead of the most significant bits. A simple implementation for square roots is shown in Algorithm 10, costing asymptotically $O(1) \cdot \mathbf{M}$. This algorithm has been used for small numbers before [Ham2014] but to our knowledge has not been thoroughly described for large ones. This method calculates each iteration mod some power-of-2 value R to reduce multiplication cost. Those calculations (but not the one on line 7) can be done modulo any larger power of 2 instead, for simplicity or to align with the hardware's capabilities; for example, we can fix $R = 2^{L+1}$, though this increases the runtime to $O(\log \log N) \cdot \mathbf{M}$.

Algorithm 10 Quadratic Hensel lifting for integer square roots

Require: An odd positive input $N \in \mathbb{N}$.
Ensure: Return $\sqrt{N}$ if N is square, or else `not square`.

1: **if** $N \not\equiv 1 \bmod 8$ **then return** `not square`
2: Calculate $L \leftarrow \left\lceil \frac{1}{2} \log_2 N \right\rceil$ ▷ *Thus* $2^L \geq \sqrt{N}$
3: Set $\alpha \leftarrow 1$
4: **for** i **from** 1 **to** $\lceil \log_2 \max(1, L) \rceil$ **do** ▷ *Max is to avoid taking* $\log_2 0$
5: Set $R \leftarrow 2^{\min(2^i, L)+1}$
6: Update $\alpha \leftarrow \alpha \cdot \frac{(3-\alpha^2 N) \bmod 2R}{2} \bmod R$ ▷ *Gives* $\alpha^2 N \equiv 1 \bmod 2R$
7: Let $\beta \leftarrow \min\left(\alpha N \bmod 2^{L+1}, -\alpha N \bmod 2^{L+1}\right)$ ▷ *Extract* $0 < \beta < 2^L$.
8: **if** $\beta^2 = N$ **then return** $\sqrt{N} = \beta$
9: **else return** `not square`

To prove this algorithm correct, it suffices to show that if N is an odd square, then on line 8 we will have $\beta = \sqrt{N}$. First note that for any $a, b, N, R \in \mathbb{Z}$ with R even, if $a \equiv b \bmod R$, then $a^2 N \equiv b^2 N \bmod 2R$, since their difference is $N(a+b)(a-b)$, and $R|(a-b)$, and $2|(a+b)$. We next show an invariant: when entering iteration i of the loop, we have $\alpha^2 N \equiv 1 \bmod 2^{2^{i-1}+2}$, or in other words, there is an integer k such that

$$\alpha^2 N = 1 + 2^{2^{i-1}} \cdot 4k.$$

We show this by induction, with base case $i = 1$ satisfied by $N \equiv 1 \bmod 8$. Finishing the iteration with a new α_{next}, we want to show that $\alpha_{\text{next}}^2 N \equiv 1 \bmod$

$2R$. Let $S := 2^{2^{i-1}}$. We have:

$$\begin{aligned}
\alpha_{\text{next}}^2 \cdot N &= N \cdot \left(\frac{3 - \alpha^2 N}{2} \cdot \alpha \bmod \mathrm{R} \right)^2 \\
&\overset{*}{\equiv} \alpha^2 N \cdot \left(\frac{3 - \alpha^2 N}{2} \right)^2 \bmod 2\mathrm{R} \\
&\overset{**}{=} (1 + 4Sk) \cdot (1 - 2Sk)^2 \\
&= 1 - 12S^2k^2 + 16S^3k^3 \\
&\equiv 1 \bmod 4S^2.
\end{aligned}$$

Here $*$ is true by the note, and $**$ follows from the induction hypothesis. Now $2R|4S^2$ (and they are equal in every iteration except possibly the last), so the congruence also holds mod $2R$.

Exiting the loop we likewise have $\alpha^2 N \equiv 1 \bmod 2^{L+2}$, so also $\beta^2 \equiv (\alpha N)^2 \equiv N \bmod 2^{L+2}$. An odd number N has at most four modular square roots mod 2^m, of the form $\pm x$ and $2^{m-1} \pm x$, and of these line 7 sets β to the unique answer in the range $0 < \beta < 2^L$. Since also $0 < \sqrt{N} < 2^L$, we must have $\beta = \sqrt{N}$ if the latter is an integer.

Of course, the same method can calculate the square roots of even numbers by factoring out powers of 4 first.