The Efficient Calculation of a Combined Fermat PRP and Lucas PRP Test

Paul Underwood

November 24, 2014

Abstract

By an elementary observation about the computation of the difference of squares for large integers, a deterministic probable prime (PRP) test is given with a running time almost equivalent to that of a Lucas PRP test but with the combined strength of a Fermat PRP test and a Lucas PRP test.

1 Introduction

Much has been written about Fermat PRP tests [1, 2, 3], Lucas PRP tests [4, 5], Frobenius PRP tests [6, 7, 8, 9, 10, 11, 12] and combinations of these [13, 14, 15]. These tests provide a probabilistic answer to the question: "Is this integer prime?" Although an affirmative answer is not 100% certain, it is answered fast and reliable enough for "industrial" use [16]. For speed, these PRP tests are usually preceded by factoring methods such as sieving and trial division.

The speed of the PRP tests depends on how quickly multiplication and modular reduction can be computed during exponentiation. Techniques such as Karatsuba's algorithm [17, section 9.5.1], Toom-Cook multiplication, Fourier Transform algorithms [17, section 9.5.2] and Montgomery exponentiation [17, section 9.2.1] play their roles for different integer sizes. The sizes of the bases used are also critical.

Oliver Atkin introduced the concept of a "selfridge" [18], approximately equal to the running time of a Fermat PRP test. Hence a Lucas PRP test is 2 selfridges. The Baillie-PSW test costs 1+2 selfridges, the use of which is very efficient when processing a candidate prime list; There is no known Baillie-PSW pseudoprime but Greene and Chen give a way to construct some examples [19]. However, if the 2 selfridges test presented in this paper

is preceded with a Fermat 2-PRP test it also becomes 1+2 selfridges, but with the strength of a 1+1+2 selfridges one.

2 Calculation $\pmod{n, x^2 - ax + 1}$

At first sight, taking a modularly reduced power of x would appear to be more efficient than taking a modularly reduced power of something more complicated, but this turns out to be false for the case presented here.

For integer a, the equation $x^2 - ax + 1 = 0$ has discriminant $\Delta = a^2 - 4$ and solution

$$x = \frac{a \pm \sqrt{\Delta}}{2}$$

There is a method to reduce any power of x to one of at most degree 1 polynomial in x since, for integer $n > 1$, we can recursively use

$$x^n \equiv (ax - 1)x^{n-2}$$

For an odd prime p, the Jacobi symbol $\left(\frac{\Delta}{p}\right)$ equals the Legendre symbol $\left(\frac{\Delta}{p}\right)$. So if Jacobi symbol $\left(\frac{\Delta}{p}\right) = -1$ then Δ will not be square modulo p, and by the Frobenius automorphism, $x^p \equiv a - x \pmod{p, x^2 - ax + 1}$ so that

$$x^p + x \equiv a \pmod{p, x^2 - ax + 1}$$

In general, for a prime number, p, and for integers S and T:

$$(Sx + T)^p = S^p x^p + (\sum_{i=1}^{p-1} \binom{p}{i} (Sx)^{p-i} T^i) + T^p$$

and since the indicated binomial coefficients are divisible by p and since $S^p \equiv S \pmod{p}$ and $T^p \equiv T \pmod{p}$ we have

$$(Sx + T)^p \equiv Sx^p + T \pmod{p}$$

Multiplying by $Sx + T$ gives

$$\begin{array}{rcll} (Sx + T)^{p+1} & \equiv & (Sx + T)(Sx^p + T) & \pmod{p} \\ & \equiv & S^2 x^{p+1} + STx^p + STx + T^2 & \pmod{p} \\ & \equiv & S^2 x^{p+1} + ST(x^p + x) + T^2 & \pmod{p} \\ & \equiv & S^2 + aST + T^2 & \pmod{p, x^2 - ax + 1} \quad (*) \end{array}$$

In practice, left to right binary exponentiating of $Sx + T$ to the $(n+1)^{th}$ power can be accomplished with intermediate values s and t as follows.

Firstly, obtain the binary representation of $n+1$. Secondly, assign $s = S$ and $t = T$. Thirdly, loop over bits of $n+1$, left to right, starting at the 2^{nd} bit, squaring the intermediate sum, $sx + t$, at each stage and if the corresponding bit is 1 multiply the resulting squared intermediate sum by the base $Sx + T$.

Squaring the intermediate sum is achieved with appropriate modular reductions:

$$\begin{aligned}
(sx+t)^2 &= s^2x^2 + 2stx + t^2 \\
&\equiv s^2(ax-1) + 2stx + t^2 && (\text{mod } n, x^2 - ax + 1) \\
&\equiv (as^2 + 2st)x - s^2 + t^2 && (\text{mod } n, x^2 - ax + 1) \\
&\equiv s(as+2t)x + (t-s)(t+s) && (\text{mod } n, x^2 - ax + 1)
\end{aligned}$$

If the bit is 1 in the loop then the following must be calculated:

$$\begin{aligned}
(sx+t)(Sx+T) &= sSx^2 + (sT+tS)x + tT \\
&\equiv sS(ax-1) + (sT+tS)x + tT && (\text{mod } n, x^2 - ax + 1) \\
&\equiv (asS + sT + tS)x + tT - sS && (\text{mod } n, x^2 - ax + 1)
\end{aligned}$$

If a, S and T are small then the squaring part is dominated by 2 major multiplications and 2 modular reductions: s by $as + 2t$ modulo n, and $t - s$ by $t + s$ modulo n; the "if" part is relatively faster. This makes an algorithm that is a little over 2 selfridges. In pseudocode the process can be summarized as

```
function general(n,a,S,T) {
  BIN=binary(n+1);
  LEN=length(BIN);
  aSpT=a*S+T;
  s=S;
  t=T;
  for (index=2;index<=LEN;index++) {
    temp=(s*(a*s+2*t))%n;
    t=((t-s)*(t+s))%n;
    s=temp;
    if (BIN[index]==1) {
      temp=s*aSpT+t*S;
      t=t*T-s*S;
      s=temp;
    }
  }
  return( (s==0) && (t==(S*S+a*S*T+T*T)%n) );
}
```

If $S = 1$ and $T = 0$ the following program for computing the binary Lucas chain [17, algorithm 3.6.7] is quicker, being 2 selfridges

```
function lucas(n,a) {
  BIN=binary(n);
  LEN=length(BIN);
  va=2;
  vb=a;
  for (index=1;index<=LEN;index++) {
    if (BIN[index]==1) {
      va=(va*vb-a)%n;
      vb=(vb*vb-2)%n;
    } else {
      vb=(va*vb-a)%n;
      va=(va*va-2)%n;
    }
  }
  return( (va==a) && (vb==2) );
}
```

3 Equivalence of Tests

The main test $(*)$ for odd n, with Jacobi symbol $\left(\frac{\Delta}{n}\right) = -1$, is

$$(Sx + T)^{n+1} \equiv S^2 + aST + T^2 \pmod{n, x^2 - ax + 1}$$

This is equivalent to

$$x^{n+1} \equiv S^2 + aST + T^2 \pmod{n, x^2 - (aS + 2T)x + S^2 + aST + T^2}$$

Let

$$P = aS + 2T$$
$$Q = S^2 + aST + T^2$$

If $\gcd(PQ, n) = 1$, so that the inverse of Q modulo n exists, the main test implies the Fermat PRP test:

$$Q^{n-1} \equiv 1 \pmod{n}$$

and, following [17, Section 3.6.3], implies the Lucas PRP test:

$$x^{n+1} \equiv 1 \pmod{n, x^2 - (\frac{P^2}{Q} - 2)x + 1}.$$

The number 21 with $a = 6$, $S = 1$ and $T = 8$, and so $P = 22 \equiv 1 \pmod{21}$ and $Q = 113 \equiv 8 \pmod{21}$, is an example composite that passes the Fermat PRP test but not the Lucas PRP test. For a vice versa example: composite 27 with $a = 6$, $S = 1$ and $T = 7$, so that $P = 20$ and $Q = 92 \equiv 11 \pmod{27}$, passes the Lucas PRP test but not the Fermat PRP test.

4 The Main Algorithm for $S = 1$ and $T = 2$

A test is now presented that is a little over 2 selfridges. In comparison to the binary Lucas chain algorithm for a binary representation with an average number of ones and zeroes, the presented test requires an extra 7 operations per loop iteration of multiple precision word additions or multiplications of multiple precision words by small numbers. Branching the code to handle the cases where $a = 0$ and $a = 1$ will reduce the operation count to 5 and 6 respectively. Perhaps the biggest difference in running times for the various PRP tests is that a Fermat PRP is dominated by a modularly reduced squaring per loop iteration; the Lucas PRP test requires a modularly reduced squaring and a modularly reduced multiplication in its loop iteration; and the test presented in this section requires 2 modularly reduced multiplications per loop iteration. For an example of this difference, if Fourier Transform arithmetic is used, only 1 forward transform is required for a squaring operation, whereas 2 are required for multiplication.

For a candidate odd prime n, a *minimal* integer $a \geq 0$ such that the Jacobi symbol $\left(\frac{a^2-4}{n}\right) = -1$ is sought. Then there is no ambiguity about how the algorithm works, there is no randomness. If, while searching for a minimum a, a value is found such that the Jacobi symbol $\left(\frac{a^2-4}{n}\right) = 0$ then clearly n is not prime, but this is unlikely to occur if sieving or trial division is performed firstly. Another reason for choosing a minimal a is that there is more likelihood that the Jacobi symbol will be 0 for the numerous candidates with small factors. The time taken to test a Jacobi symbol is negligible, but some time can be saved by testing a chosen in order from

$$0, 1, 3, 5, 6, 9, 11, 12, 13, 15, 17, 19, 20, 21, 24, 25, 27, 29, 30, 31, 32 \ldots$$

Clearly, 2 is to be omitted from this list. $a = 4$ is omitted because it is covered by $a = 0$ and $a = 1$. $a = 7$ is omitted since $\left(\frac{3^2-4}{n}\right) = \left(\frac{7^2-4}{n}\right)$, and so on. Also, if a candidate prime is a square number then a Jacobi symbol equal to -1 will not be found. So it is recommended that a squareness test, which is rapid, is computed initially. To ensure the implications of section

3, the following is screened for:

$$\gcd((a+4)(2a+5), n) = 1$$

On finding a Jacobi symbol equal to -1 the following test can be done:

$$(x+2)^{n+1} \equiv 2a+5 \pmod{n, x^2 - ax + 1}$$

The pseudocode for this test is

```
function selfridge2(n,a) {
  BIN=binary(n+1);
  LEN=length(BIN);
  ap2=a+2;
  s=1;
  t=2;
  for (index=2;index<=LEN;index++) {
    temp=(s*(a*s+2*t))%n;
    t=((t-s)*(t+s))%n;
    s=temp;
    if (BIN[index]==1) {
      temp=ap2*s+t;
      t=2*t-s;
      s=temp;
    }
  }
  return( (s==0) && (t==(2*a+5)%n) );
}
```

Using primesieve [20] and the GMP library [21], verification of the algorithm was pre-screened with the implied Fermat PRP test $(2a+5)^{n-1} \equiv 1 \pmod{n}$. For odd $n < 2^{50}$ there were 1,518,678 such pseudoprimes. The maximum a required was 81, for $n = 170557004069761$. However, none that passed pre-screening were a pseudoprime for the full algorithm when run under Pari/GP [22].

5 Conclusion

Figure 1 is a plot of pseudoprimes of the algorithm given in section 4 but for freely ranging a and odd $n < 2 \cdot 10^7$. This leaves us with the question: Does a minimum a for a pseudoprime ever come close to the minimum a required by the algorithm?

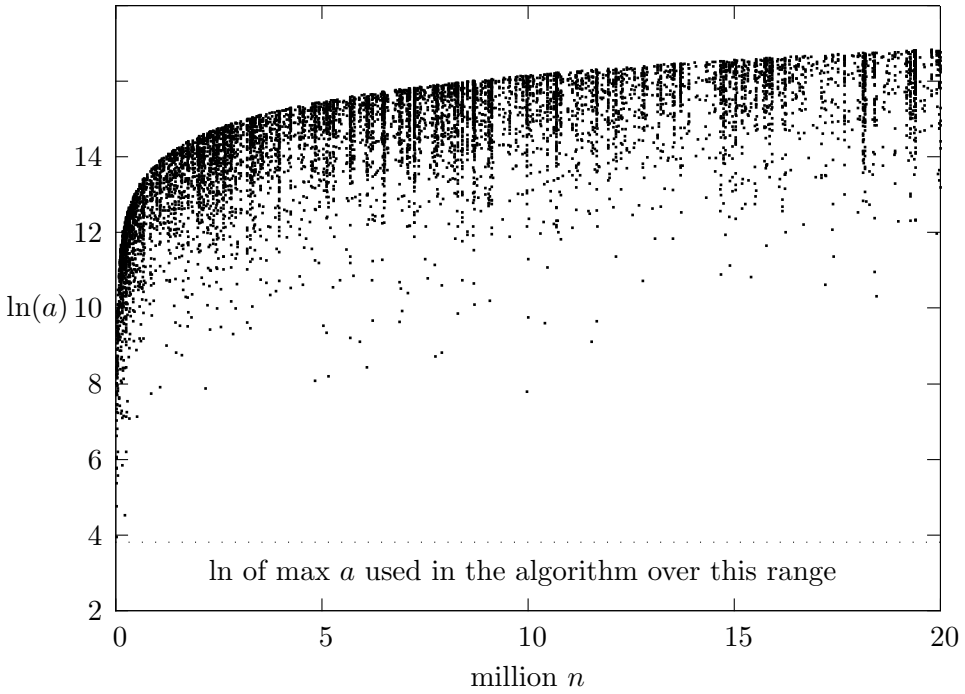

Figure 1: Pseudoprimes for S=1 and T=2

Another question: The Baillie-PSW test uses two independent tests, a strong Fermat 2-PRP test and a specific Lucas PRP test, whereas the test given in section 4 depends on one parameter, a. Can this difference influence reliability?

6 Acknowledgements

I thank Vincent Diepeveen for helping me code C/C++ and for allowing me to use his prime sieving function. Thanks to members of mersenneforum.org and Yahoo! primenumbers groups for their encouraging remarks, in particular Carlos Pinho, Maximilian Hasler and Dana Jacobsen.

References

[1] C. Pomerance, J. L. Selfridge, and S. S. Wagstaff, Jr., "The pseudoprimes to $25 \cdot 10^9$," *Mathematics of Computation*, vol. 35, no. 151, pp. 1003–1026, 1980.

[2] M. O. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory*, vol. 12, no. 1, pp. 128–138, 1980.

[3] S. H. Kim and C. Pomerance, "The probability that a random probable prime is composite," *Mathematics of Computation*, vol. 53, no. 188, pp. 721–741, 1989.

[4] F. Arnault, "The rabin-monier theorem for lucas pseudoprimes," *Mathematics of Computation*, vol. 66, no. 218, pp. 869–881, 1997.

[5] H. C. Williams, *Édouard Lucas and Primality Testing.* Wiley-Interscience, 1998.

[6] J. Grantham, "A frobenius probable prime test with high confidence," *Journal of Number Theory*, vol. 72, pp. 32–47, 1998.

[7] J. Grantham, "Frobenius pseudoprimes," *Mathematics of Computation*, vol. 70, pp. 873–891, 2001.

[8] S. Müller, "A probable prime test with very high confidence for n equiv 1 mod 4," *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, pp. 87–106, 2001.

[9] I. B. Damgård and G. S. Frandsen, "An extended quadratic frobenius primality test with average and worst case error estimates," *Lecture Notes in Computer Science. Fundamentals of Computation Theory (Springer Berlin Heidelberg)*, vol. 2751, pp. 118–131, 2003.

[10] M. Seysen, "A simplified quadratic frobenius primality test." `http://www.dm.unito.it/~cerruti/zyx/Articoli/simplified-quadratic-frobenious.pdf`, 2005.

[11] D. Loebenberger, "A simple derivation for the frobenius pseudoprime test." Cryptology ePrint Archive, Report 2008/124, 2008. `http://eprint.iacr.org/`.

[12] S. Khashin, "Counterexamples for frobenius primality test," *eprint arXiv:1307.7920*, 2013.

[13] R. Baillie and S. S. Wagstaff, Jr., "Lucas pseudoprimes," *Mathematics of Computation*, vol. 35, pp. 1391–1417, October 1980.

[14] C. Pomerance, "Are there counterexamples to the baillie-psw primality test?." `http://www.pseudoprime.com/dopo.pdf`, 1984.

[15] Z. Zhang, "A one-parameter quadratic-base version of the baillie-psw probable prime test," *Mathematics of Computation*, vol. 71, no. 240, pp. 1699–1734, 2002.

[16] C. Caldwell, "probable prime." `http://primes.utm.edu/glossary/xpage/PRP.html`, 1999-2014.

[17] R. Crandall and C. Pomerance, *Prime Numbers, A Computational Perspective, 2nd Ed.* Springer, 2005.

[18] A. O. L. Atkin., "Intelligent primaility test offer," *Computational Perspectives on Number Theory (D. A. Buell and J. T. Teitelbaum, eds.), Proceedings of a Conference in Honor of A. O. L. Atkin, International Press*, pp. 1–11, 1998.

[19] J. R. Greene and Z. Chen, "Want to earn some cash the hard way?." `http://www.d.umn.edu/~jgreene/baillie/Baillie-PSW.html`.

[20] "primesieve." `http://primesieve.org`, 2014.

[21] "The gnu multiple precision arithmetic library." `https://gmplib.org/`, 2014.

[22] "Pari/gp." `http://pari.math.u-bordeaux.fr/`, 2014.

E-mail address: `paulunderwood@mindless.com`