

ECPP in PARI/GP

Jared Asuncion

18 January 2018

Problem

Given an integer N , prove that it is prime.

Proposition

Let $N > 6$ be an integer. If there exists:

- an integer m
- a prime q
- an elliptic curve E over $\mathbb{Z}/N\mathbb{Z}$
- and a point P on E

such that

- $m = qs$ for some $s \in \mathbb{Z}$
- $q > (N^{1/4} + 1)^2$
- $mP = \infty$
- $sP \neq \infty$

then N is prime.

Manual verification

?

Manual verification

```
? N = 97;
```

```
?
```

Manual verification

? $N = 97;$

? $m = 93;$

?

Manual verification

```
? N = 97;
```

```
? m = 93;
```

```
? q = 31;
```

```
?
```

Manual verification

```
? N = 97;
```

```
? m = 93;
```

```
? q = 31;
```

```
? E = ellinit( [Mod(69, N), Mod(2, N)] );
```

```
?
```


Manual verification

```
? N = 97;
```

```
? m = 93;
```

```
? q = 31;
```

```
? E = ellinit( [Mod(69, N), Mod(2, N)] );
```

```
? P = [12, 91];
```

```
?
```

Manual verification

```
? N = 97;
```

```
? m = 93;
```

```
? q = 31;
```

```
? E = ellinit( [Mod(69, N), Mod(2, N)] );
```

```
? P = [12, 91];
```

```
? s = m/q;
```

```
?
```

Manual verification

```
? N = 97;
```

```
? m = 93;
```

```
? q = 31;
```

```
? E = ellinit( [Mod(69, N), Mod(2, N)] );
```

```
? P = [12, 91];
```

```
? s = m/q;
```

```
? (N^(1/4) + 1)^2
```

```
% = 17.125435787226096882657755669491649229
```

```
?
```

Manual verification

```
? N = 97;  
? m = 93;  
? q = 31;  
? E = ellinit( [Mod(69, N), Mod(2, N)] );  
? P = [12, 91];  
? s = m/q;  
? (N^(1/4) + 1)^2  
% = 17.125435787226096882657755669491649229  
? ellmul(E,P,m)  
% = [0]  
?
```

Manual verification

```
? N = 97;  
? m = 93;  
? q = 31;  
? E = ellinit( [Mod(69, N), Mod(2, N)] );  
? P = [12, 91];  
? s = m/q;  
? (N^(1/4) + 1)^2  
% = 17.125435787226096882657755669491649229  
? ellmul(E,P,m)  
% = [0]  
? ellmul(E,P,s)  
% = [Mod(23, 97), Mod(46, 97)]
```

Manual verification

```
? N = 97;
? m = 93;
? q = 31;
? E = ellinit( [Mod(69, N), Mod(2, N)] );
? P = [12, 91];
? s = m/q;
? (N^(1/4) + 1)^2
% = 17.125435787226096882657755669491649229
? ellmul(E,P,m)
% = [0]
? ellmul(E,P,s)
% = [Mod(23, 97), Mod(46, 97)]
```

If $q = 31$ is prime, then $N = 97$ is prime.

First examples of ecpp

?

First examples of ecpp

```
? N = 97;
```

```
?
```


First examples of ecpp

```
? N = 97;
```

```
? ecpp(N)
```

```
% = 97
```

```
?
```

First examples of ecpp

```
? N = 97;
```

```
? ecpp(N)
```

```
% = 97
```

```
? N = 51;
```

```
?
```

First examples of ecpp

```
? N = 97;
```

```
? ecpp(N)
```

```
% = 97
```

```
? N = 51;
```

```
? ecpp(N)
```

```
% = 0
```

```
?
```

First examples of ecpp

```
? N = 97;
```

```
? ecpp(N)
```

```
% = 97
```

```
? N = 51;
```

```
? ecpp(N)
```

```
% = 0
```

```
? N = ecpp(2^100000);
```

```
?
```

First examples of ecpp

```
? N = 97;
```

```
? ecpp(N)
```

```
% = 97
```

```
? N = 51;
```

```
? ecpp(N)
```

```
% = 0
```

```
? N = ecpp(2^100000);
```

```
? ecpp(N)
```

```
% = 0
```

Bigger example for ecpp

?

Bigger example for ecpp

```
? N = randomprime(10^50)
```

```
% = 6218021483076269348132291041647045291708771276517
```

```
?
```

Bigger example for ecpp

```
? N = randomprime(10^50)
```

```
% = 6218021483076269348132291041647045291708771276517
```

```
? cert = ecpp(N)
```


Bigger example for ecpp

```
? N = randomprime(10^50)
```

```
% = 6218021483076269348132291041647045291708771276517
```

```
? cert = ecpp(N)
```

```
% = [[6218021483076269348132291041647045291708771276517, -11  
68858896165124200384481, 104197826361, 0, [27221942753529546  
79804716336431441756799877037507, 17595898176504920460947713  
58785510025002448074670]], [59675155425349644430979496356501  
173759, -5741511905690628272, 112816, 5967515542534964443097  
9496356449505884, [7674594461480654864891586987992110768, 25  
367332704693906308475248433180992766]], [5289600360352223482  
19410440559777, -36557113351409951, 2416487659, 0, [50133951  
3959449988194168803017069, 104622117438367884727793381682328  
]], [218896229022795263849731, 836507980571, 27916833, 0, [4  
2728418640565069383966, 72784625979380726131541]]]
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
?
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ??ecpp  
?
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ??ecpp  
? #cert  
% = 4  
?
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ??ecpp  
? #cert  
% = 4  
? N = cert[1][1]  
% = 6218021483076269348132291041647045291708771276517  
?
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ??ecpp  
? #cert  
% = 4  
? N = cert[1][1]  
% = 6218021483076269348132291041647045291708771276517  
? t = cert[1][2]  
% = -1168858896165124200384481  
?
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ??ecpp  
? #cert  
% = 4  
? N = cert[1][1]  
% = 6218021483076269348132291041647045291708771276517  
? t = cert[1][2]  
% = -1168858896165124200384481  
? s = cert[1][3]  
% = 104197826361  
?
```

Interpreting the certificate

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ??ecpp  
? #cert  
% = 4  
? N = cert[1][1]  
% = 6218021483076269348132291041647045291708771276517  
? t = cert[1][2]  
% = -1168858896165124200384481  
? s = cert[1][3]  
% = 104197826361  
? a4 = cert[1][4]  
% = 0  
?
```


Interpreting the certificate

```

? N = 6218021483076269348132291041647045291708771276517;
? cert = ecpp(N);
? ??ecpp
? #cert
% = 4
? N = cert[1][1]
% = 6218021483076269348132291041647045291708771276517
? t = cert[1][2]
% = -1168858896165124200384481
? s = cert[1][3]
% = 104197826361
? a4 = cert[1][4]
% = 0
? P = cert[1][5]
% = [2722194275352954679804716336431441756799877037507, 1759
589817650492046094771358785510025002448074670]

```

Using ecppexport

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
?
```

Using ecppexport

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? expo = ecppexport(cert)  
% = "\n[1]\n N = 62180214830762693481322910416470452917 [+++]  
?
```

Using ecppexport

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? expo = ecppexport(cert)  
% = "\n[1]\n N = 62180214830762693481322910416470452917 [+++]  
? print(expo)  
[1]  
N = 6218021483076269348132291041647045291708771276517  
t = -1168858896165124200384481  
s = 104197826361  
a4 = 0  
D = -3  
m = 6218021483076269348132292210505941456832971660999  
q = 59675155425349644430979496356501173759  
E = [0, 4726763364707743671876647238916186952220236477595]  
P = [2936327345386591725898087517171112698109138902400 [+++]
```

Exporting to MAGMA/PRIMO

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
?
```

Exporting to MAGMA/PRIMO

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? magma = ecpexport(cert,1)  
% = "[* [* 62180214830762693481322910416470452917087712 [+++]  
?
```

Exporting to MAGMA/PRIMO

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? magma = ecpexport(cert,1)  
% = "[* [* 62180214830762693481322910416470452917087712 [+++]  
? primo = ecpexport(cert,2)  
% = "[PRIMO - Primality Certificate]\nFormat=4\nTestCou [+++]  
?
```

Exporting to MAGMA/PRIMO

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? magma = ecppexport(cert,1)  
% = "[* [* 62180214830762693481322910416470452917087712 [+++]  
? primo = ecppexport(cert,2)  
% = "[PRIMO - Primality Certificate]\nFormat=4\nTestCou [+++]  
? write("primo.out", primo);
```


Is the certificate valid?

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
?
```

Is the certificate valid?

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ecppisvalid(cert)  
% = 1  
?
```

Is the certificate valid?

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ecppisvalid(cert)  
% = 1  
? cert[4][5] = [118915338768549494077476, 216300436757184809  
16130];  
?
```

Is the certificate valid?

```
? N = 6218021483076269348132291041647045291708771276517;
```

```
? cert = ecpp(N);
```

```
? ecppisvalid(cert)
```

```
% = 1
```

```
? cert[4][5] = [118915338768549494077476, 216300436757184809  
16130];
```

```
? ecppisvalid(cert)
```

```
% = 0
```

```
?
```

Is the certificate valid?

```
? N = 6218021483076269348132291041647045291708771276517;  
? cert = ecpp(N);  
? ecppisvalid(cert)  
% = 1  
? cert[4][5] = [118915338768549494077476, 216300436757184809  
16130];  
? ecppisvalid(cert)  
% = 0  
? ecppisvalid(vector(100, i, i))  
% = 0
```

Assume that we have a list \mathcal{L} of discriminants we want to try.

PHASE I: Find m and q .

For each discriminant D under consideration:

- **Write $m = N + 1 - U$**

where (U, V) is an integer solution to $U^2 + |D|V^2 = 4N$.^a

- involves taking a square root of D modulo N
- involves doing a half-gcd

- **Try to write m as $m = qs$**

where q and s are as in the theorem.

- involves finding *small* factors of m
- good idea to check pseudoprimality of q

^a m can take at most six other forms as well

Assume that we have the corresponding (D, m, q) from Phase 1.

PHASE II: Find E and P .

- Find a root j modulo N of the Hilbert class polynomial of $\mathbb{Q}(\sqrt{D})$.
 - involves computing a class polynomial (of degree h)
 - involves finding a root of this polynomial modulo N
- Find an elliptic curve E modulo N with j -invariant j and a point P satisfying the conditions in the theorem.
 - finding the twist involves finding a quadratic non-residue

Finding a solution to $U^2 + |D|V^2 = 4N$.

Globally:

- Keep the prime factorization of each D on the list \mathcal{L} .
- Only consider D which are B -smooth for some B .
- Only consider D which are associated to low class numbers.

For each N :

- Test for necessary conditions for the existence of a solution.
- Find \sqrt{D} modulo N by finding the square roots of primes dividing D .
- Remember the square roots of the primes for future use.
- Find (and remember) a quadratic nonresidue g .

Removing small factors

- Precompute the product \mathcal{P} of all primes less than S .
- Divide m by its gcd with \mathcal{P} . Do this repeatedly until they are coprime.
- There is a clever way to treat multiple m at once.

Finding a root j of the Hilbert class polynomial.

- Remember and reuse previous computations.
- Prioritize discriminants D which give polynomials of lower degree (during the first phase).
- Use class polynomials obtained from other invariants.

Finding an elliptic curve E and a point P .

- Use the quadratic nonresidue g to find twists of elliptic curves.

What to do when the list \mathcal{L} runs out?

- If we are at the top of the recursion tree, then expand the list to include higher class numbers.
- Otherwise, mark that instance as FAIL and resume the computations one level up.

Average runtime for a sample size of 100 primes, single thread.

	1000 bits	2000 bits	3000 bits	4000 bits
Magma 2.11-13	≥ 60.00	-	-	-
ecpp-dj-1.04	2.43	57.4	383	1731
Pari 2.10 isprime	3.02	33.9	175	751
Primo 4.21	3.90	28.0	124	438
Pari 2.10 ecpp	3.04	24.7	113	442

Runtime for each phase, single thread:

		1000 bits	2000 bits	3000 bits	4000 bits
Primo 4.21	I	3.06	20.4	91.1	340
	II	0.84	7.5	32.6	98
Pari ecpp	*	0.31	2.3	4.7	26
	I	1.03	12.0	63.4	260
	II	1.71	10.4	44.7	157

(*) is the average time spent for precomputations

Average runtime for each step of Pari ecpp:

	1000 bits	2000 bits	3000 bits	4000 bits
$\sqrt{D} \pmod{N}$	0.12	4.52	32.2	168
factoring smalls	0.76	4.10	11.8	23
pseudoprimality	0.15	3.33	19.4	68
class polynomial	1.14	3.02	5.0	7
root mod N	0.22	4.94	31.1	128
find E and P	0.35	2.42	8.3	21

- Phase I operations are done more frequently as N becomes higher. It is harder to express $m = qs$ as $m \approx N$ becomes bigger.
- There is a big spike at 4000 bits for computing the roots. This is because the degree can go higher than 40.

New libpari commands




- `Fp_sqrt_i`
find the square root of x given a quadratic nonresidue mod p
- `cornacchia2_sqrt`
finds the solution to $U^2 + |D|V^2 = 4N$ given $\sqrt{D} \pmod{N}$
- `disc_best_modinv`
find the invariant to use with `polclass` such that the height of the output is low
- `FpX_oneroot_split`
finds one root of a totally split polynomial over $F_p[X]$
- removing small factors by batch

Further

- Use intermediate fields of H_K/K to obtain lower degree polynomials and find their roots instead.
- Parallel computation.

Thank you for listening!

References

-  A. Atkin and F. Morain, *Elliptic curves and primality proving*, Mathematics of Computation **61** (1993), no. 203, 29–68.
-  J. Franke, T. Kleinjung, F. Morain, and T. Wirth, *Proving the primality of very large numbers with fastecpp*, pp. 194–207, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
-  F. Morain, *Implementing the asymptotically fast version of the elliptic curve primality proving algorithm*.