GENEFER

A PROGRAM FOR FINDING
LARGE PROBABLE GENERALIZED FERMAT PRIMES

Mathematical representation and algorithms

Yves Gallot

November 11, 2016

GENEFER is free source code, under the MIT license.

Chapter 1

Introduction

> Si je puis une fois tenir la raison fondamentale que 3, 5, 17, etc. sont nombres premiers, il me semble que je trouverai de très belles choses en cette matière,
>
> *Fermat à Mersenne*
> *25 décembre 1640*

Pierre de Fermat [8] conjectured that every number of the form

$$F_n = 2^{2^n} + 1,$$

where n is a non-negative integer, is prime.

Today these positive integers are named Fermat numbers. The first five Fermat numbers are prime, but Leonhard Euler proved in 1732 that 641 divides F_5. Now we know that F_n is composite for $5 \leq n \leq 32$.

F_{33} is a number over two billion decimal digits and the largest known prime $2^{74207281} - 1$ is a number with $22,338,618$ digits. If we search for prime numbers, Fermat numbers should be generalized.

Theorem. *If $b > 1$ and $b^N + 1$ is prime, then b is even and $N = 2^n$.*

Proof. If b is odd then $b^N + 1$ is even; and if N has an odd factor k and $N = kl$, then $b^N + 1$ is divisible by $b^l + 1$. □

In 1986, Harvey Dubner [6] have searched for primes of the form $b^{2^n} + 1$ and found the 4457-digit prime $150^{2^{11}} + 1$. GENEFER [7, 2, 1] has been extensively used for searching for large generalized Fermat primes. Thanks to the distributed computing project PrimeGrid's Generalized Fermat Prime Search, Masashi Kumagai discovered the 2,976,633-digit prime $475856^{2^{19}} + 1$ in 2012.

Chapter 2

Primality Test

Let n be a positive integer and b be a positive even number. Put $N = 2^n$ and $g = b^N + 1$.

GENEFER 'quick' test checks if g is a Fermat probable prime to base 2:

$$2^{g-1} \equiv 1 \pmod{g}.$$

For $n = 1$ some are probable prime but composite (called pseudoprimes); it happens for $b = 216, 948, 1560, \ldots$ [14].

If b is a power of two, g is a Fermat number and is a Fermat probable prime to base 2.

But if $n \geq 2$ and b is not a power of two (g is of the form $a^4 + 1$), all known probable primes are prime. Then this simple test is well-founded.

The following program is a basic PARI-GP [11] implementation of GENEFER test:

```
check(b, N)=
{
  printf("%d^%d+1 is ", b, N);
  g = b^N + 1;
  r = lift(Mod(2, g)^(g - 1));
  if (r == 1,
    printf("a probable prime.");
  ,
    printf("composite. (RES=");
    res = r \ b^(N - 8);
    for (k = 1, 8,
      printf("%02x", (res % b) % 256);
      res \= b;
    );
    printf(")");
  );
  printf("\n");
}
```

A residue is computed to be able to verify that a primality test was performed without error (double-checking). This program and GENEFER residues are identical.

The primality of generalized Fermat numbers can also be proved with GENEFER.
The following theorem generates the proof:

Theorem (Lucas 1876)**.** *Let g be a positive integer. If there is a positive integer a such that*

$$\begin{aligned} a^{g-1} &\equiv 1 \pmod{g} \\ a^{\frac{g-1}{q}} &\not\equiv 1 \pmod{g} \end{aligned}$$

for all prime factors q of $g-1$ then g is prime.

a is a primitive root and is a Pratt primality certificate [13].

In practice we choose a such that the Jacobi symbol $\left(\frac{a}{g}\right) = -1$ by the quadratic reciprocity law.

Chapter 3

Transforms

3.1 Introduction

Let b, n be some positive integers, $N = 2^n$, GENEFER computes

$$2^{b^N} \mod b^N + 1$$

using a left-to-right binary modular exponentiation [4, Algorithm 1.2.3]. The advantage of this algorithm is that each step is a squaring and a duplication if $\mathrm{bit}(b^N, k) = 1$.

The main program is a single loop

$$x \to a_k\, x^2 \mod b^N + 1,$$

where $a_k = \mathrm{bit}(b^N, k) + 1$ and k scans the binary representation of b^N from left to right. The number of steps is the number of bits of b^N.

3.2 Fast modular squaring

$x^2 \mod b^N + 1$ can be computed as a squaring followed by a reduction modulo $b^N + 1$. If base-b representation is used the reduction is fast because of the relation $a^{N+k} \mod a^N + 1 = -a^k$. With a Fast Fourier Transform-based multiplication, the complexity is $O(N \log N \log\log N)$.

In [5], the authors introduced the concept of Discrete Weighted Transforms and proved that multiplication modulo Fermat numbers may be effected without zero-padding, in the form of halved run length. In [7], the method is extended to generalized Fermat numbers. This algorithm was used in the first versions of GENEFER and is about twice as fast as a FFT-based squaring followed by a modular reduction.

The current version of GENEFER is based on a new algorithm: a direct transform that calculates the product of two polynomials modulo $x^N + 1$. The FFT-based multiplication is the product of

two polynomials modulo $x^N - 1$. The DWT is computed via premultiplication of the j^{th} input digit by g^j, with g a primitive N-th root of -1, then calculating the usual FFT-based convolution of the weighted input and scaling the outputs by the inverse weights. The G-Transform is faster because input and output are not weighted.

	FFT	DWT	G-Transform
Length	$2N$	N	N
Weighted	no	yes	no
Complexity ($b \leq b_{max}$)	$\alpha\, 2N \log(2N)$	$\alpha N \log N + N$	$\alpha N \log N$

Table 3.1: Comparison of transforms

3.3 Convolution modulo $x^{2^n} + 1$

We can follow different paths that lead to the same algorithm. Three approaches are detailed in the following paragraphs. A first one is based on elementary number theory: the Chinese Remainder Theorem over monic polynomial of degree one rings is sufficient to prove the result. After this, we show that the transform is a z-Transform: this approach is more complex because the knowledge of the theory of z-Transforms is necessary. The algorithm was found using this way. Finally, we show that the reduction modulo $x^N + 1$ removes half of a FFT-based squaring. The use of a DIF/IDIT pair for convolutions gives DWT and a DIT/IDIF pair gives G-Transforms.

Let n be a positive integer, $N = 2^n$ and $P(x)$ be a polynomial over $\mathbb{Z}$ of degree smaller than N. We search for

$$S(x) \equiv P(x)^2 \pmod{x^N + 1},$$

where $\deg(S) < N$.

3.3.1 Polynomial multiplication

We have

$$x^N + 1 = \prod_{\zeta_k} (x - \zeta_k),$$

where $\zeta_0, \zeta_1, \ldots, \zeta_{N-1}$ are the primitive $2N^{\text{th}}$ roots of unity.

Note that $x^N + 1$ is the $2N^{\text{th}}$ cyclotomic polynomial $\Phi_{2N}(x)$.
The transform is computed over a field $\mathbb{F}$. If $\mathbb{F}$ is the quotient ring $\mathbb{Z}/p\,\mathbb{Z}$ and if ζ is a primitive $2N^{\text{th}}$ root then $\zeta_k = \zeta^{2k+1}$. If $\mathbb{F}$ is the complex numbers $\mathbb{C}$, we can choose $\zeta_k = e^{2\pi i \frac{2k+1}{2N}}$.

Now we apply the Chinese Remainder Theorem to the system of congruences

$$\begin{aligned} P(x) &\equiv r_0 \pmod{x - \zeta_0} \\ &\vdots \\ P(x) &\equiv r_{N-1} \pmod{x - \zeta_{N-1}} \end{aligned}$$

where $\{r_0, r_1, \ldots, r_{N-1}\} \in \mathbb{F}^N$.

But the remainder of the division by $x - \zeta_k$ of a polynomial $P(x)$ is $P(\zeta_k)$ (Lagrange interpolation).

Then the algorithm is

1. Evaluate $P(x)$ at N points: $\zeta, \zeta^3, \ldots, \zeta^{2N-1}$.
2. Form the N products $s_k = r_k^2$.
3. Solve the interpolation problem for the coefficients s_k to find $S(x)$.

Step 1 is named the transform and step 3 is the inverse transform. The interpolation problem is a mathematical problem but the computational procedure is to compute step 1 in reversed order. In our case, if step 1 is solved then 3 is trivial.

Let $P(x) = \sum_{j=0}^{N-1} a_j\, x^j$, the system becomes

$$r_k = \sum_{j=0}^{N-1} a_j\, \zeta^{j(2k+1)}. \tag{3.1}$$

Therefore $r_k = \sum_{j=0}^{N/2-1} \left(a_j + a_{j+N/2}\, \zeta^{(N/2)(2k+1)}\right) \zeta^{j(2k+1)}$.

Put $b_j = a_j + a_{j+N/2}\, \zeta^{N/2}$ and $c_j = a_j - a_{j+N/2}\, \zeta^{N/2}$.

We have $r_{2k} = \sum_{j=0}^{N/2-1} b_j\, \zeta^{j(4k+1)}$ and $r_{2k+1} = \sum_{j=0}^{N/2-1} c_j\, \zeta^{j(4k+3)}$.

We expressed a N-point transform in terms of two $N/2$-point transforms. Since $N = 2^n$, we can apply this method recursively until we get to the trivial 1-point transform. Relation 3.1 has a complexity of $O(N^2)$; Algorithm 3.1 is a fast transform and has a complexity of $O(N \log N)$.

Algorithm 3.1 (Recursive G-Transform)**.**

$a[\,] \leftarrow$ GTRANSFORM($a[\,]$, $N/2$, 0)

function GTRANSFORM($a[\,]$, m, k)
  if $m < 1$ **then return**
  $w \leftarrow \zeta^{m\cdot(2k+1)}$
  for $0 \le j < m$ **do**
    $b[j] \leftarrow a[j] + a[j+m] \cdot w$
    $c[j] \leftarrow a[j] - a[j+m] \cdot w$
  GTRANSFORM(b, $m/2$, k)
  GTRANSFORM(c, $m/2$, $(N/2)/m + k$)
  for $0 \le j < m$ **do**
    $a[2j+0] \leftarrow b[j]$
    $a[2j+1] \leftarrow c[j]$

Now we can notice that the output doesn't have to be ordered because the N products $s_k = r_k^2$ are independent. The inverse transform will execute the reciprocal operations in reversed order and thus take jumbled input and generate normal order output.
Finally recursion is replaced by iteration and we get a more efficient 'in-place' transform:

Algorithm 3.2 (In-Place G-Transform)**.**

function GTRANSFORM($a[\,], N$)
  $m \leftarrow N/2$
  while $m \geq 1$ **do**
    for $0 \leq j < (N/2)/m$ **do**
      $w \leftarrow \zeta^{m\cdot(2\ \mathrm{bitRev}(j,(N/2)/m)+1)}$
      for $0 \leq i < m$ **do**
        $k \leftarrow 2mj + i$
        $u \leftarrow a[k]$, $v \leftarrow a[k+m] \cdot w$
        $a[k] \leftarrow u + v$, $a[k+m] \leftarrow u - v$
    $m \leftarrow m/2$

bitRev(j, s) is the 'bit-reversal' permutation of a sequence of s items (s is a power of two). For fixed N, $w(j, m)$ is constant and stored in a pre-calculated table.

A 'butterfly' is

$$\begin{aligned} \widehat{a}_k &= a_k + a_{k+m}\, w \\ \widehat{a}_{k+m} &= a_k - a_{k+m}\, w \end{aligned}$$

then

$$\begin{aligned} a_k &= 2^{-1}(\widehat{a}_k + \widehat{a}_{k+m}) \\ a_{k+m} &= 2^{-1}(\widehat{a}_k - \widehat{a}_{k+m})\, w^{-1}. \end{aligned}$$

Instead of multiplying by 2^{-1} at each step, we can multiply the final result by $2^{-n} = N^{-1}$.
The inverse transform is given by:

Algorithm 3.3 (Inverse G-Transform)**.**

function GINVERSETRANSFORM($a[\,], N$)
  $m \leftarrow 1$
  while $m \leq N/2$ **do**
    for $0 \leq j < (N/2)/m$ **do**
      $\overline{w} \leftarrow \zeta^{-m\cdot(2\ \mathrm{bitRev}(j,(N/2)/m)+1)}$
      for $0 \leq i < m$ **do**
        $k \leftarrow 2mj + i$
        $u \leftarrow a[k]$, $v \leftarrow a[k+m]$
        $a[k] \leftarrow u + v$, $a[k+m] \leftarrow (u - v) \cdot \overline{w}$
    $m \leftarrow 2m$
  for $0 \leq k < N$ **do**
    $a[k] \leftarrow a[k] \cdot N^{-1}$

3.3.2 z-Transform

In [3] DFT z-Transform filters are studied and a FFT algorithm is obtained: its flow graph (see [3, Fig. 6]) is unusual (for number theorists but not in signal processing engineering). The butterfly is a DIT but the input array is not in bit-reversed order. The twiddle factors and the output array are bit-reversed.

This algorithm splits the filter $z^{-2n}+1$ into $z^{-n}+1$ and $z^{-n}-1$ (see [3, Fig. 4 and 5]). A filter removes some roots and modular arithmetic keeps them: then $z^{-n}+1 \iff x^n-1$ and $z^{-n}-1 \iff x^n+1$. If we transpose to modular arithmetic the polynomial $x^{2n}-1$ is expressed in terms of x^n-1 and x^n+1.

Let $\Phi_n(x)$ be the n^{th} cyclotomic polynomial, we have

$$x^n-1=\prod_{d|n}\Phi_d(x).$$

With this approach, we can split convolutions modulo x^n-1 (FFT) into basic convolutions (transforms) modulo $\Phi_d(x)$. The G-Transform is this transform modulo $\Phi_{2N}(x)=x^N+1$.

3.3.3 FFT and DWT

The 'Classic' radix-2 Decimation In Time algorithm is:

Algorithm 3.4 (Radix-2 DIT FFT, Cooley-Tukey)**.**

function FFT($a[\,]$, N)
  BITREV($a[\,]$, N)
  $m \leftarrow 1$
  while $m \le N/2$ **do**
    for $0 \le j < m$ **do**
      $w \leftarrow \zeta^{j(N/2)/m}$
      for $0 \le i < (N/2)/m$ **do**
        $k \leftarrow 2mi+j$
        $u \leftarrow a[k]$, $v \leftarrow a[k+m]\cdot w$
        $a[k] \leftarrow u+v$, $a[k+m] \leftarrow u-v$
    $m \leftarrow 2m$

Put $m' = \text{bitRev}(m,N) = (N/2)/m$, $j' = \text{bitRev}(j,m)$, $i' = \text{bitRev}(i,(N/2)/m)$ and $k' = \text{bitRev}(k,N)$. We have:

Algorithm 3.5 (Radix-2 DIT FFT)**.**

function FFT($a[\,]$, N)
  $m' \leftarrow N/2$
  while $m' \ge 1$ **do**
    for $0 \le j' < (N/2)/m'$ **do**
      $w \leftarrow \zeta^{m'\cdot\text{bitRev}(j',(N/2)/m')}$
      for $0 \le i' < m'$ **do**
        $k' \leftarrow 2m'j'+i'$
        $u \leftarrow a[k']$, $v \leftarrow a[k'+m']\cdot w$
        $a[k'] \leftarrow u+v$, $a[k'+m'] \leftarrow u-v$
    $m' \leftarrow m'/2$
  BITREV($a[\,]$, N)

This algorithm is Bruun complex FFT [3, chap. 5]. It can be observed that the roots of x^N+1 are the odd roots of $x^{2N}-1$ and algorithm 3.2 follows.

The last stage of a $2N$-point inverse FFT (DIT or DIF) is

$$\begin{aligned} a'_k &= a_k + a_{k+N} \\ a'_{k+N} &= a_k - a_{k+N} \end{aligned}$$

because $\zeta^0 = 1$. But $x^{N+k} \mod x^N + 1 = -x^k$ then the reduction step is

$$r_k = a'_k - a'_{k+N}$$

and finally

$$r_k = 2\, a_{k+N}.$$

Only the elements $a_N, a_{N+1}, \ldots, a_{2N-1}$ should be computed.

Both DWT and G-Transform can be viewed as some 'sub-FTT' diagrams.

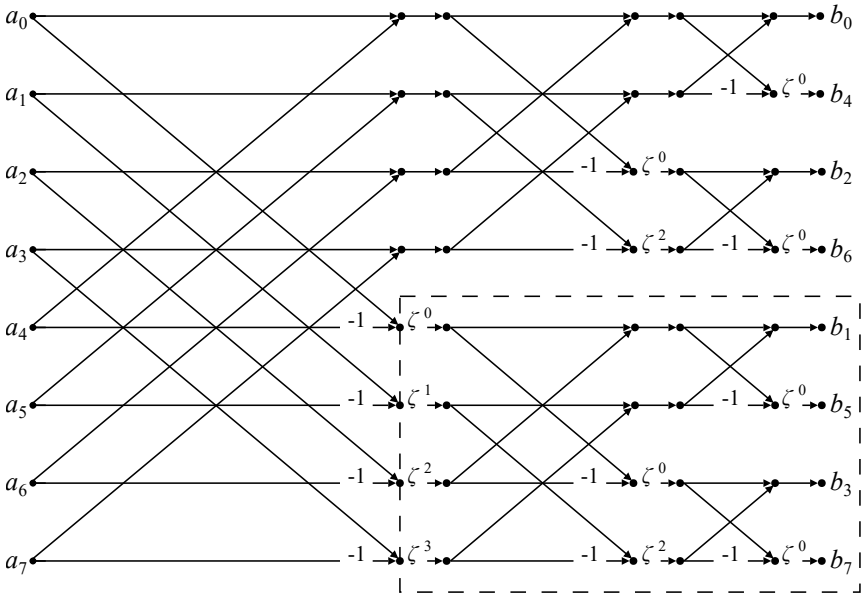

Figure 3.2: 8-point DIF FFT and DWT flow diagram.

With a DIF/IDIT FFT pair and a 'zero padded' input ($a_k = 0$ for $k \in \{N, N+1, \ldots, 2N-1\}$), the upper half of the transform is a DWT (see figure 3.2: $a_4 = a_5 = a_6 = a_7 = 0$ and $\{b_0, b_2, b_4, b_6\}$ are not computed. The weight vector is $(\zeta^0, \zeta^1, \zeta^2, \zeta^3)$ and the transform is a 4-point DIF FFT with $\zeta' = \zeta^2$.).

But with a DIT/IDIF FFT pair (and ordered input values), the upper half of the transform is a G-Transform (see figure 3.3). But N-point G-Transform can't be derived from N-point FFT.

The 2^n-point DWT algorithm requires $(n+2)\,2^{n-1}$ multiplications and $n\,2^n$ additions/subtractions. The G-Transform count is $n\,2^{n-1}$ multiplications and $n\,2^n$ additions/subtractions.

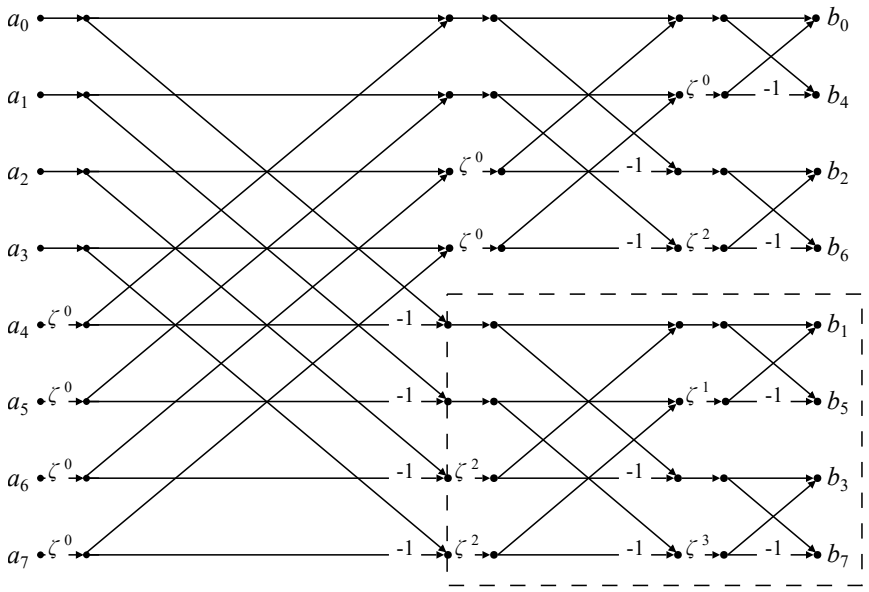

Figure 3.3: 8-point DIT FFT and G-Transform flow diagram.

3.4 Implementation

3.4.1 Complex Transform

The transform is computed over the complex numbers $\mathbb{C}$ and we choose $\zeta = e^{\frac{2\pi i}{2N}}$.

$\zeta^{N/2} = i$ then the first stage of the N-point G-Transform is

$$\begin{aligned} a'_k &= a_k + a_{k+N/2}\, i \\ a'_{k+N/2} &= a_k - a_{k+N/2}\, i. \end{aligned}$$

a_i are real then $a'_{k+N/2} = \overline{a'_k}$. The transform is a linear mapping and $\overline{z^2} = \overline{z}^2$. If b_i is the input of the last stage of the inverse G-Transform, we have $b_{k+N/2} = \overline{b_k}$ and the output is

$$\begin{aligned} b'_k &= b_k + \overline{b_k} &&= 2\ \mathrm{Re}(b_k) \\ b'_{k+N/2} &= (b_k - \overline{b_k})(-i) &&= 2\ \mathrm{Im}(b_k). \end{aligned}$$

Computation can be performed with a $N/2$-point complex G-Transform and the Gaussian integers $a'_k = a_k + a_{k+N/2}\, i$. This is similar to [5, Eq. 5.11] and a 'right-angle' convolution is computed.

In practice 64-bit or 80-bit floating-point numbers, radix-4 transforms and a balanced representation in the range $[-b, b]$ (see [5, Sec. 3]) are employed. Note that the computation of a balanced representation in $[-b/2, b/2]$ is slower and doesn't significantly improve accuracy. Round-off errors are computed and a test is reported as being incorrect if one error exceeds 0.45.

3.4.2 Number Theoretic Transform

Let $p = 2^{64} - 2^{32} + 1$, the transform is computed over the quotient ring $\mathbb{Z}/p\,\mathbb{Z}$.

$2^{96} \equiv -1 \pmod{p}$ then reduction can be calculated efficiently with

$$a\,2^{96} + b\,2^{64} + c \equiv b\,2^{32} + c - a - b \pmod{p}.$$

Graphics Processing Unit cores are 32-bit processors. It is possible to replace the 64-bit prime p by a number of smaller 32-bit primes $\{p_1, p_2, \ldots, p_t\}$ and to find the least absolute residue modulo $P = p_1 \cdot p_2 \cdots p_t$ by the Chinese Remainder Theorem [12].

$p_1 = 125 \cdot 2^{25} + 1$ and $p_2 = 243 \cdot 2^{24} + 1$ are a possible choice. Because p_1 and p_2 are fixed, the division is performed as a multiplication by a precomputed approximation of the reciprocal of the divisor, followed by two adjustment steps [10, Algorithm 4].

But we can use the fact that $w(j,m)$ are invariant (see Alg. 3.2) and compute $x \cdot w \mod p_i$ with Shoup's modular multiplication algorithm [15] [9, Algorithm 2]. Now primes must be some 31-bit integers and Barrett reduction is faster than [10, Algorithm 4] if p_i are being chosen such that a single conditional subtraction is necessary.

For GENEFER we use the two primes $p_1 = 127 \cdot 2^{24} + 1$, $p_2 = 15 \cdot 2^{27} + 1$ or the three primes p_1, p_2, $p_3 = 51 \cdot 2^{25} + 1$.

Bibliography

[1] I. Bethune and Y. Gallot, *Genefer: Programs for finding large probable generalized Fermat primes*, Journal of Open Research Software, **3(1)** (2015), e10, DOI: http://dx.doi.org/10.5334/jors.ca.

[2] I. Bethune and M. Goetz, *Extending the generalized Fermat prime number Search beyond one million digits using GPUs*, Parallel Processing and Applied Mathematics, **8384** (2014), 106–113, DOI: http://dx.doi.org/10.1007/978-3-642-55224-3_11.

[3] G. Bruun, *z-transform DFT filters and FFT's*, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol **26**, (1978), 56–63, DOI: http://dx.doi.org/10.1109/TASSP.1978.1163036.

[4] H. Cohen, *A course in computational algebraic number theory*, Graduate Texts in Mathematics, vol. **138**, Springer-Verlag, New York, 1993.

[5] R. Crandall and B. Fagin, *Discrete Weighted Transforms and Large-Integer Arithmetic*, Math. Comp. **62** (1994), 305–324, DOI: http://dx.doi.org/10.1090/S0025-5718-1994-1185244-1.

[6] H. Dubner, *Generalized Fermat primes*, J. Recreational Math. **18** (1985-86), 279–280, http://fishmech.net/My%20Documents/Harvey/Papers/Generalized%20Fermat%20Primes.doc.

[7] H. Dubner and Y. Gallot, *Distribution of generalized Fermat prime numbers*, Math. Comp. **71** (2002), 825–832, DOI: http://dx.doi.org/10.1090/S0025-5718-01-01350-3.

[8] Pierre de Fermat, *Lettre à Marin Mersenne*, http://www.archive.org/stream/oeuvresdefermat942ferm#page/212/mode/2up.

[9] D. Harvey, *Faster arithmetic for number-theoretic transforms*, Journal of Symbolic Computation, Volume **60**, (2014), 113–119, DOI: http://dx.doi.org/10.1016/j.jsc.2013.09.002

[10] N. Moller and T. Granlund, *Improved Division by Invariant Integers*, IEEE Transactions on Computers, vol **60**, (2011), 165–175. DOI: http://dx.doi.org/10.1109/TC.2010.143.

[11] The PARI group, *PARI/GP Development*, 2003–2016, http://pari.math.u-bordeaux.fr/gp.html.

[12] J. M. Pollard, *The fast Fourier transform in a finite field*, Math. Comp. **25** (1971), 365–374, DOI: http://dx.doi.org/10.1090/S0025-5718-1971-0301966-0.

[13] V. R.. Pratt, *Every Prime Has a Succinct Certificate*, SIAM J. Comput., **4(3)** (1975), 214–220, DOI: http://dx.doi.org/10.1137/0204018.

[14] The On-Line Encyclopedia of Integer Sequences, **A135590**, http://oeis.org/A135590.

[15] V. Shoup, *NTL: A Library for doing Number Theory*, 1996–2016, http://www.shoup.net/ntl.