On Primitivity Tests for Polynomials

Andreas Rieke
University of Hagen
58084 Hagen, Germany
andreas.rieke@fernuni-hagen.de

Ahmad-Reza Sadeghi
University of Saarland
66041 Saarbrücken, Germany
sadeghi@cs.uni-sb.de

Werner Poguntke
Märkische Fachhochschule
58095 Hagen, Germany
poguntke@mfh-iserlohn.de

Abstract — **This paper considers algorithms for primitivity tests of polynomials over finite fields. The algorithms are optimized for the case that the polynomials tested turn out to be primitive.**

As the main contribution, we suggest a careful selection among many known algorithms for exponentiation. The results provide a primitivity test whose implementation confirmed that it is significantly faster than other known algorithms.

I. Introduction

Primitive polynomials over finite fields are an important class of polynomials which find many applications in coding theory and cryptography. In this context, algorithms for testing given polynomials on primitivity are of interest.

We use the prime factor decomposition

$$q^m - 1 = \prod_{j=1}^{s} p_j^{r_j}$$

and the method given in [1, p. 82] to test polynomials of degree m over a finite field F_q where q is a power of a prime.

Test algorithms may be optimized for the worst and the average case. Since our test algorithm consists of checking several necessary conditions whose conjunction is also sufficient, the worst case is given when the polynomial tested is primitive.

Since factorization of large integers is a time consuming task, we use tables with precomputed prime factor decompositions for all q and m of interest.

Many algorithms have been developed for efficient and fast exponentiation; amongst these, we have selected the window method [2] and the Lim/Lee method [3] for our test algorithms because these are very basic and efficient algorithms.

Since a test on primitivity is usually processed by a unit that has enough memory to store intermediate results (i. e. we do not consider chipcards, but computers with several Mbytes of memory), the performance of the algorithms is assessed in terms of the number of operations.

II. Algorithms for a Fast Primitivity Test

In the following, we introduce algorithms that carry out the test for a given polynomial $t(x)$. We restrict our attention to the case that $q^m - 1$ is not prime, since otherwise a test on irreducibility would be sufficient.

The most important aim in the first algorithm A is to reduce the number of operations by avoiding to process one operation more than once. Instead of processing all exponentiations completely, we use the method shown in Table 1. We only have to process the exponentiations marked with $\boxed{\wedge}$, since those marked with $\wedge$ are already processed elsewhere. Using primes sorted in ascending order ($p_j < p_{j+1}$), we reach the minimum number of operations.

j	$x^{(q^m-1)/p_j}$ with $p_0 = 1$
0	$x \wedge \prod_{k=1}^{s} p_k^{r_k-1} \wedge p_s \wedge p_{s-1} \wedge \cdots \wedge p_i \wedge \cdots \wedge p_2 \boxed{\wedge} p_1$
1	$x \wedge \prod_{k=1}^{s} p_k^{r_k-1} \wedge p_s \wedge p_{s-1} \wedge \cdots \wedge p_i \wedge \cdots \boxed{\wedge} p_2$
2	$x \wedge \prod_{k=1}^{s} p_k^{r_k-1} \wedge p_s \wedge p_{s-1} \wedge \cdots \wedge p_i \wedge \cdots \quad \boxed{\wedge} p_1$
$\vdots$	
i	$x \wedge \prod_{k=1}^{s} p_k^{r_k-1} \wedge p_s \wedge p_{s-1} \wedge \cdots \quad \boxed{\wedge} \cdots \boxed{\wedge} p_2 \boxed{\wedge} p_1$
$\vdots$	
s-1	$x \wedge \prod_{k=1}^{s} p_k^{r_k-1} \boxed{\wedge} p_s \quad \boxed{\wedge} \cdots \boxed{\wedge} p_i \boxed{\wedge} \cdots \boxed{\wedge} p_2 \boxed{\wedge} p_1$
s	$x \boxed{\wedge} \prod_{k=1}^{s} p_k^{r_k-1} \quad \boxed{\wedge} p_{s-1} \boxed{\wedge} \cdots \boxed{\wedge} p_i \boxed{\wedge} \cdots \boxed{\wedge} p_2 \boxed{\wedge} p_1$

Tab. 1: Primitivity test for exponentiation with algorithm A

The main objective of the second algorithm B is to find many exponentiations which use the same precomputed results.

We construct 6 variants out of the two algorithms for a primitivity test using the two exponentiation algorithms and analyze which one is best suited to test a given polynomial. Since none of these variants turns out to have satisfactory performance in all cases, we select a specific variant that is expected to perform well for a given problem depending only on the number of different prime factors s of $q^m - 1$.

III. Numerical Results

We compare the performance of the proposed variants in GF(2). Due to missing factorizations of numbers $2^m - 1$ for $m > 512$, we restrict ourselves to degrees $2 \leq m \leq 512$. For each degree, we perform a primitivity test for each of the proposed variants. The results are given depending on the number of different factors s of $2^m - 1$.

We use primitive polynomials for the test in order to get worst case results.

Using the proposed algorithm, our implementation is up to 15 times faster than other test algorithms and implementations.

Further information about primitive polynomials including an online-test with the proposed algorithm can be obtained at `http://www-kommsys.fernuni-hagen.de/~rieke/primitiv/`.

References

[1] Rudolf Lidl and Harald Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, revised edition, 1994.

[2] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

[3] Chae Hoon Lim and Pil Joong Lee, "More flexible exponentiation with precomputation", in *Advances in Cryptology – CRYPTO '94*, pp. 95–107. Springer, Berlin, 1994.