Information Processing Letters 132 (2018) 33–38

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Fast modular reduction and squaring in $GF(2^m)$

L. Boppre Niehues [a,*], R. Custódio [a], D. Panario [b]

[a] Department of Informatics and Statistics, Federal University of Santa Catarina, Brazil
[b] School of Mathematics and Statistics, Carleton University, Canada

ARTICLE INFO

Article history:
Received 26 November 2016
Received in revised form 13 October 2017
Accepted 11 December 2017
Available online 12 December 2017
Communicated by L. Viganò

Keywords:
Finite field
Number theory
Polynomial
Squaring
Cryptography

ABSTRACT

We present an efficient bit-parallel algorithm for squaring in $GF(2^m)$ using polynomial basis. This algorithm achieves competitive efficiency while being aimed at any choice of low-weight irreducible polynomial. For a large class of irreducible polynomials it is more efficient than the previously best general squarer. In contrast, other efficient squarers often require a change of basis or are suitable for only a small number of irreducible polynomials. Additionally, we present a simple algorithm for modular reduction with equivalent cost to the state of the art for general irreducible polynomials. This fast reduction is used in our squaring method.

1. Introduction

Arithmetic in the finite field $GF(2^m)$ of 2^m elements (also denoted $\mathbb{F}_{2^m}$) is fundamental for many important cryptosystems such as ECC (Elliptic Curve Cryptography). Such arithmetic is usually implemented by choosing an irreducible polynomial $f \in \mathbb{F}_2[x]$, $\deg(f) = m$, performing operations and reducing modulo this polynomial. It is common to have algorithms that implement arithmetic operations using a particular class of irreducible polynomials. This is because these algorithms are efficiently implemented considering such class of irreducibles.

Arithmetic operations in $GF(2^m)$ usually consist of addition (which is equivalent to subtraction in characteristic 2), multiplication (of which squaring is a special case) and inverses. All these operations are used in ECC, making any optimizations reflect directly on the speed of elliptic curve arithmetic, raising the importance of choosing an irreducible polynomial and associated algorithms. As examples, the classes of pentanomials $x^m + x^{n+2} + x^{n+1} + x^n + 1$, $x^{4s} + x^{3s} + x^{2s} + x^s + 1$, $x^m + x^{m-r} + x^s + x^r + 1$ and $x^m + x^{n+1} + x^n + x + 1$ are used in fast multipliers [1], each with their own fast multiplication and reduction algorithms.

However, the algorithms designed for irreducible polynomials with specific exponents may contain internal structures with an unclear impact on security of applications that use these algorithms. No attacks have been demonstrated so far, but the security community has seen evidence of standards containing back doors [2] and cryptography failing due to fixed parameters [3]. In light of these events there has been discussions for less magic parameters and more randomness in the structures used.

A related problem is that many classes of irreducible polynomials contain too few elements. In ECC, for example, many classes of polynomials often have no irreducible polynomials for a desired degree. Furthermore, choosing a class to speed up a specific operation may lead to less efficient algorithms for other operations used in the same application.

We introduce a general algorithm for $GF(2^m)$ modular reduction and an efficient squarer suitable for *any* low-

* Corresponding author.
E-mail addresses: lucasboppre@inf.ufsc.br (L.B. Niehues), ricardo.custodio@ufsc.br (R. Custódio), daniel@math.carleton.ca (D. Panario).

https://doi.org/10.1016/j.ipl.2017.12.002
0020-0190/

weight irreducible polynomial f. These algorithms operate on elements represented in polynomial basis, where the coefficients are stored simply as an array of bits, and operations are performed bitwise in a logic circuit. Such algorithms are usually measured by the number of bit-level XOR operations performed and their circuit delay (we note that only XOR operations are required in these algorithms). Previous works on this operation have been limited to certain classes of irreducible polynomials to achieve competitive efficiency, fixing the weight (number of non-zero elements) and the relationship among exponents. Although our algorithms hold for any low-weight polynomial, to achieve better circuit delay and be more comparable to other proposals, we focus on the pentanomial case $x^m + x^a + x^b + x^c + 1$, $m > a > b > c > 0$, where $a \leq \lceil m/2 \rceil$.

Our squarer has different costs depending on the irreducible polynomial used. Applications requiring utmost efficiency should choose an irreducible polynomial to minimize the global cost of operations. We observe there are polynomials that minimize the number of XOR operations and delay with our squarer; we show that low-weight polynomials with this characteristic are abundant (see Table 3 at Section 4). Additionally, our squarer can be used for higher weight polynomials, with some performance penalty.

The structure of this paper is as follows. In Section 2, we present a general algorithm to perform modular reduction. This algorithm is generic and can be used with any irreducible polynomial. In Section 3, we modify the algorithm previously proposed to make it a squarer. This strategy allowed us to propose a squarer algorithm of low complexity. In Section 4, we compare our squarer with previous methods for this operation. Section 5 shows how the reduction and squaring algorithm can be generalized for p-th power computation in characteristic p. We give final conclusions in Section 6.

2. Modular reduction

Let $GF(2^m)$ be a finite field generated by an irreducible polynomial $f(x) = x^m + r(x)$, where $\deg(r) < m$. Some polynomial basis operations performed on this field may require a reduction modulo f. This is a classical operation; see for example [4, Chapter 2.3.5].

Let $C(x) = \sum_{i=0}^{d} c_i x^i$, $d \geq m$, be the result of an operation before the reduction mod f is executed. A simple method for reduction consists of iteratively computing $C(x) \leftarrow C(x) + c_i x^{i-m} f(x)$ for $i = d, d-1, \ldots, m$, where each step i reduces the coefficient c_i.

To reduce the number of operations we replace $c_i x^{i-m} f(x)$ with $c_i x^{i-m} r(x)$ in the equation above, and then compute the final result mod x^m, which is a simple truncation. This avoids one XOR per iteration because r has one less coefficient than f. We consider the truncation to have negligible cost. Algorithm 1 is a pseudocode representation of these operations. The input consists of $d+1$ signals (the input coefficients), which are modified using XOR operations and returned as m signals. We describe the details of this algorithm since it is a step in the construction of the main contribution of this paper: our squarer method.

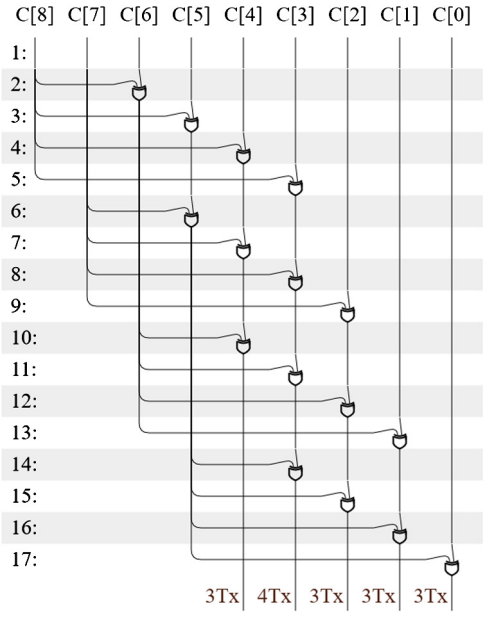

Fig. 1. Example of circuit generated by 1 by fixing $f(x) = x^5 + x^3 + x^2 + x + 1$.

Algorithm 1 General modular reduction for $GF(2^m)$.

Input: $C = [c_0, c_1, c_2, ..., c_d]$, $d \geq m$, $f(x) = x^m + r(x)$
Output: $C \bmod f$
1: **for** $i = d, d-1, \ldots, m$ **do**
2: **for each** exponent e **of** r **do**
3: $C[i-m+e] \leftarrow C[i-m+e] \oplus C[i]$
4: **end for**
5: **end for**
6: **return** $C[0]$, $C[1]$, $C[2]$, $\ldots$, $C[m-1]$

By fixing the irreducible polynomial in Algorithm 1, the instructions can be mapped naturally to a circuit of XOR gates. Fig. 1 shows an example of such circuit, for $f(x) = x^5 + x^3 + x^2 + x + 1$. The input coefficients are given at the top, with each numbered row representing a step of the algorithm. Note that all steps are in the form $C[i] \leftarrow C[i] \oplus C[j]$ for some i, j. At the bottom of each column is the wire containing the result signal, along with the total circuit delay for that wire, which measures the number of XOR operations in its critical path (T_x). The total delay of the circuit is therefore the largest of these individual delays.

Algorithm 1 uses $(d-m+1)w_r$ XOR operations, where w_r is the weight of the polynomial r. If C is a product of two polynomials of degree at most $m-1$, then the degree of C is at most $d \leq 2m-2$. It is common, in practical implementations, to fix $d = 2m-2$. In this case, the number of XOR operations to perform a modular reduction for trinomials and pentanomials are $2m-2$ and $4m-4$, respectively. The trinomial algorithm can be trivially modified to the equally spaced case, $x^m + x^{m/2} + 1$, for m even, where many operations cancel themselves out and result in $1.5m - 1$ XOR operations [5].