Available online at www.sciencedirect.com

SCIENCE DIRECT

Information Processing Letters 94 (2005) 11–14

Information Processing Letters

www.elsevier.com/locate/ipl

Subtraction-free Almost Montgomery Inverse algorithm

Róbert Lórencz [*], Josef Hlaváč

Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, Karlovo nám. 13, 121 35 Praha 2, Czech Republic

Received 6 September 2004; received in revised form 14 December 2004

Available online 18 January 2005

Communicated by A.A. Bertossi

Abstract

A new modification of Kaliski's algorithm for computing the Almost Montgomery Inverse, optimized for implementation in hardware, is presented. The algorithm and the corresponding portion of the hardware architecture overcome certain drawbacks of the currently used methods. In particular, the "less than/greater than" tests that represent subtractions are avoided.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Almost Montgomery Inverse algorithm; Cryptography; Finite field arithmetic

1. Introduction

Multiplicative inverse in a Galois field GF(p) is a rather complex operation that is useful in various cryptographic algorithms, such as the computation of parameters in the decipherment operation of the RSA algorithm [12], certain digital signature systems [11], in computing point operations on elliptic curves defined over GF(p) [7,10], or in accelerating modular exponentiation using a so-called addition–subtraction chain [1,9].

Using the classical definition (e.g., in [8]), the multiplicative inverse of an integer $a \in [1, p-1]$ modulo a prime p is an integer $x \in [1, p-1]$ such that $ax \equiv 1 \pmod{p}$, often written as $x = a^{-1} \bmod p$.

Papers [6,13] introduce the Montgomery Modular Inverse (MMI). The MMI of an integer $a \in [1, p-1]$ is

$$MMI(a) \equiv a^{-1}2^n \pmod{p},$$

where p is prime and $n = \lceil \log_2 p \rceil$. The papers [6,13] also detail an algorithm for computing the MMI (Kaliski's algorithm). As an intermediate result, the so-called Almost Montgomery Inverse (AMI) is calculated:

$$AMI(a) \equiv a^{-1}2^k \pmod{p},$$

where $a \in [1, p-1]$, p is a prime, and $k \in [n, 2n]$ is the number of iterations performed. Computing the MMI of an input operand either in integer or Mont-

[*] Corresponding author.
E-mail addresses: lorencz@fel.cvut.cz (R. Lórencz), hlavacj2@fel.cvut.cz (J. Hlaváč).

0020-0190/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ipl.2004.12.012

gomery domain is then a matter of dividing or multiplying by 2, respectively, an appropriate number of times:

$$MMI(a) \equiv AMI(a) \cdot 2^{n-k} \equiv a^{-1}2^n \pmod p,$$
$$MMI(a2^n) \cdot 2^n \equiv AMI(a2^n) \cdot 2^{2n-k} \pmod p$$
$$\equiv a^{-1}2^n \pmod p.$$

2. Proposed modification

In the following algorithms, “$==$” represents equality comparison. Symbols “$=$” and “$\leftarrow$” both represent assignment to a variable; however, in hardware, “$=$” means that a value is present at the output of a combinatorial circuit, while “$\leftarrow$” indicates that a value is written into a register upon the next rising edge of the clock signal.

The AMI phase of the current algorithms [2–6,13] can be rewritten as Algorithm 1. In this version, the algorithm exits as soon as $u == v$, and the inverse is recovered from s instead of $-r$. Compared to the algorithm in [6,13], variables u, r, s are not updated in the final iteration, returned k is smaller by 1, and s is always less than p. Operations in lines 9–10 of the algorithm in [13] are thus unnecessary and there is no need for an extra bit in r.

Algorithm 1. AMI WITH SUBTRACTIONS

Input: $a \in [1, p-1]$ and $p > 2$ is prime
Output: $o \in [1, p-1]$ and k, where
$o = a^{-1}2^k \bmod p$ and $n-1 \leqslant k < 2n$

```
1. $u \leftarrow p, v \leftarrow a, r \leftarrow 0, s \leftarrow 1, k \leftarrow 0$
2. while(1)
3.    if ($u_{LSB} == 0$) then
4.       $u \leftarrow u/2, s \leftarrow 2s$
5.    else if ($v_{LSB} == 0$)
6.       $v \leftarrow v/2, r \leftarrow 2r$
7.    else
8.       $x = u - v, y = r + s$
9.       if ($x == 0$) then return $o \leftarrow s$ and $k$
10.      if (CARRY($x$) $== 1$) then
11.         $u \leftarrow x/2, r \leftarrow y, s \leftarrow 2s$
12.      else
13.         $v \leftarrow (x = (v - u))/2, s \leftarrow y, r \leftarrow 2r$
14.   $k \leftarrow k + 1$
```

The subtraction $u - v$ and the CARRY test in lines 8–10 represent the $u > v$ test in [6,13]. If $u > v$, the result is divided by 2 and stored in u (line 11). Otherwise, $v - u$ is calculated, the result divided by 2 and stored in v (line 13). (If $u == v$, the algorithm exits.)

In hardware, this sequence can be implemented in two ways. The first option is to use two subtractors that perform the $u - v$ and $v - u$ operations in parallel and take only the positive result. This approach is used, e.g., in [2,3,5]. The second option is to use one subtractor and sometimes an extra clock cycle. In one clock cycle, $u - v$ is computed, and if the result is positive, operations in line 11 are performed. If the result is negative, an extra clock cycle is used to perform the operations in line 13. This method is used, e.g., in [4].

We propose a modification of Kaliski’s algorithm that avoids such tests. The proposed SUBTRACTION-FREE AMI algorithm is shown.

Algorithm 2. SUBTRACTION-FREE AMI

Input: $a \in [1, p-1]$ and $p > 2$ is prime
Output: $o \in [1, p-1]$ and k, where
$o = a^{-1}2^k \bmod p$ and $n-1 \leqslant k < 2n$

```
1. $u \leftarrow (-p), v \leftarrow a, r \leftarrow 0, s \leftarrow 1, k \leftarrow 0$
2. while(1)
3.    if ($u_{LSB} == 0$) then
4.       $u \leftarrow u/2, s \leftarrow 2s$
5.    else if ($v_{LSB} == 0$)
6.       $v \leftarrow v/2, r \leftarrow 2r$
7.    else
8.       $x = u + v, y = r + s$
9.       if ($x == 0$) then return $o \leftarrow s$ and $k$
10.      if (CARRY($x$) $== 0$) then
11.         $u \leftarrow x/2, r \leftarrow y, s \leftarrow 2s$
12.      else
13.         $v \leftarrow x/2, s \leftarrow y, r \leftarrow 2r$
14.   $k \leftarrow k + 1$
```

In hardware, variables x and y are outputs of adders. Other variables represent registers. The contents of u are negative during the entire computation, represented in two’s complement code. The LSB subscript denotes the least significant bit. CARRY(x) is the carry output of the adder that computes $x = u + v$; if $x < 0$, i.e., $|u| > v$, then CARRY(x) $== 0$ and 1 otherwise.

3. Analysis and discussion

It can be shown that in every iteration of the AMI calculation, the same values appear in v, r, s in Algorithm 2 as in Algorithm 1, while opposite values appear in u. In the following text, the subscripts A1 and A2 added to variable names denote the variables of Algorithms 1 and 2, respectively.

Theorem 1. *The number of iterations is equal for Algorithms* 1 *and* 2, *and in each iteration it holds that* $u_{A2} = -u_{A1}$ *and* $v_{A1} = v_{A2}$.

Proof. Let us assume that $u_{A2} < 0$ and $v_{A2} > 0$ during the entire execution of Algorithm 2. Then, for x_{A2} in line 8 it holds that: $x_{A2} = u_{A2} + v_{A2} = -|u_{A2}| + v_{A2}$. If CARRY $= 0$ in line 10, then $|u_{A2}| > v_{A2}$ and the operations in line 11 are executed; otherwise, operations in line 13 execute.

Let us assume that $u_{A1} > 0$ and $v_{A1} > 0$ during the entire execution of Algorithm 1. Then, the operations performed in lines 8–13 of Algorithm 1 and lines 8–13 of Algorithm 2 produce the same results, as long as $u_{A2} = -u_{A1}$.

The assumption that u_{A1} and v_{A1} are always positive holds because $u_{A1} \leftarrow p$ and $v_{A1} \leftarrow a$ after initialization, and the steps 8–13 of Algorithm 1 can store only positive values into u and v. The assumption that $u_{A2} < 0$ and $v_{A2} > 0$ during the entire execution of Algorithm 2 holds, too. After initialization, $u_{A2} \leftarrow -p$ and $v_{A2} \leftarrow a$. During the execution, only negative values are stored into u_{A2} in line 11 (CARRY $= 0$), and only positive values are stored into v_{A2}. □

Corollary 2. *From Theorem* 1 *it follows that both algorithms compute the same result. It further follows that* $u_{A2} < 0$ *during the entire calculation. In hardware,* u_{A2} *can be represented in the two's complement code*; *as a result, the addition* $u_{A2} + v_{A2}$ *effectively replaces the subtractions in line* 8 *and line* 13 *of Algorithm* 1.

From Theorem 1 it further follows that $-p \leqslant u < 0$ and $0 \leqslant v, r, s < p$ during the entire execution of Algorithm 2. Thus, it is not necessary to store the respective sign bits (they never change) and n-bit registers and data buses are sufficient. A n-bit adder is also sufficient; when adding positive v and negative u, the carry from the nth order is equal to the carry from the $(n-1)$th order and its complement represents the sign of the result. One only needs to take into account the proper (implied) sign when shifting register contents to the right and set the MSB of the result accordingly.

Compared to existing architectures that perform every iteration in one clock cycle [2,3,5], two subtractors can be replaced with a single adder. Our approach thus reduces chip area.

Compared to architectures that need an extra clock cycle whenever $u < v$ (or vice versa) [4], our approach completes the calculation in fewer clock cycles. According to the binary gcd algorithm analysis in [8], which applies to Algorithms 1 and 2 as well, line 13 is executed on average in $k/4$ iterations. Assuming one clock cycle per iteration of either algorithm and an extra cycle per line 13 of Algorithm 1 executed, the average speedup of our approach is 1.25. For long words, when additions and subtractions are performed in multiple clock cycles, the average speedup can reach 1.5.

Unlike Algorithm 1, Algorithm 2 needs the negative value of p. It can be pre-calculated once for use in AMI as well as in other operations that require reductions modulo p. For example, $-p$ is actually more useful than p even for subsequent AMI to MMI conversion, as it can eliminate the need for a multifunction arithmetic unit (adder/subtractor). The AMI only needs to be shifted a certain number of times, subtracting p (that is, adding $-p$) whenever necessary to keep the result within $[1, p-1]$.

4. Conclusion

We have presented a significant modification of the algorithm for computing the Almost Montgomery Inverse. The modification is especially suited for hardware implementations because it avoids the drawbacks presented by the $u > v$ test, which is used in current algorithms.

The proposed algorithm reduces area complexity by one adder/subtractor, needed in Algorithm 1 if its time complexity is to be comparable to Algorithm 2. If the number of adders is the same, Algorithm 2 computes AMI in fewer clock cycles than the original Algorithm 1.

References

[1] Ö. Eğecioğlu, Ç.K. Koç, Exponentiation using canonical recoding, Theoret. Comput. Sci. 129 (2) (1994) 407–717.

[2] A. Gutub, New hardware algorithms and designs for Montgomery modular inverse computation in Galois fields GF(p) and GF(2^n), PhD Thesis, Department of Electrical and Computer Engineering, Oregon State University, 2002.

[3] A. Gutub, A. Tenca, Ç. Koç, Scalable VLSI architecture for GF(p) Montgomery modular inverse computation, in: Proc. IEEE Computer Society Annual Symposium on VLSI, 2002.

[4] J. Hlaváč, R. Lórencz, Ordinary modular inverse using AMI—hardware implementation, in: Proc. IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems—DDECS'03, Poznań, Poland, 2003, pp. 309–310.

[5] J. Hlaváč, R. Lórencz, Improved hardware architecture for computing the modular inverse using AMI, in: Proc. Internat. Conf. on Computer, Communication and Control Technologies CCCT 2003 and ISAS 2003, Orlando, FL, USA, vol. 3, 2003, pp. 255–260.

[6] B.S. Kaliski Jr., The Montgomery inverse and its applications, IEEE Trans. Comput. 44 (8) (1995) 1064–1065.

[7] N. Koblitz, Elliptic curve cryptosystem, Math. Comp. 48 (177) (1987) 203–209.

[8] D.E. Knuth, Seminumerical Algorithms, third ed., The Art of Computer Programming, vol. 2, Addison–Wesley, Reading, MA, 1998.

[9] Ç.K. Koç, High-radix and bit recoding techniques for modular exponentiation, Internat. J. Comput. Math. 40 (1991) 139–156.

[10] A.J. Menezes, Elliptic Curve Public Key Cryptosystem, Kluwer Academic Publishers, Boston, MA, 1993.

[11] National Institute of Standards and Technology (NIST), Digital Signature Standard, FIPS Publication 186-2, 2000.

[12] J.-J. Quisquarter, C. Couvreur, Fast decipherment algorithm for RSA public-key cryptosystem, Electron. Lett. 18 (21) (1982) 905–907.

[13] E. Savaş, Ç.K. Koç, The Montgomery modular inverse—revisited, IEEE Trans. Comput. 49 (7) (2000) 763–766.