Improved Montgomery modular inverse algorithm

Article *in* Electronics Letters · October 2004

DOI: 10.1049/el:20045610 · Source: IEEE Xplore

3 authors:

Corinne McIvor
University of the West of Scotland
11 PUBLICATIONS **364** CITATIONS
SEE PROFILE

Maire Mcloone
Queen's University Belfast
91 PUBLICATIONS **1,930** CITATIONS
SEE PROFILE

J.V. Mccanny
Queen's University Belfast
209 PUBLICATIONS **3,024** CITATIONS
SEE PROFILE

Improved Montgomery modular inverse algorithm

C. McIvor, M. McLoone and J.V. McCanny

A new, single and unified Montgomery modular inverse algorithm, which performs both classical and Montgomery modular inversion, is proposed. This reduces the number of Montgomery multiplication operations required by 33% when compared with previous algorithms reported in the literature. The use of this in practice has been investigated by implementation of the improved unified algorithm and the previous algorithms on FPGA devices. The unified algorithm implementation shows a significant speed-up and a reduction in silicon area usage.

Introduction: Montgomery modular inversion is a common operation in many public-key cryptosystems, including elliptic curve cryptosystems defined over GF(p). Savas and Koç [1] proposed Montgomery modular inversion algorithms based on work originally undertaken by Kaliski [2]. These algorithms were originally derived for and suited to software implementations on general-purpose microprocessors. In this Letter we present a single, unified version of these algorithms, which is suitable for both hardware and software implementations. These are demonstrated using FPGA implementations and comparisons made with comparable implementations based on Kaliski's, and Savas and Koç's algorithms.

Montgomery modular inversion algorithms:

Kaliski method: The Montgomery modular inverse of an integer $a \in [1, p-1]$, where p is a k-bit prime number, is given by

$$MonInv(a) = a^{-1}2^k (mod\ p) \tag{1}$$

The algorithm for computing this is broken into two phases. Phase 1 computes the 'almost Montgomery inverse' given by

$$Phase1(a) = a^{-1}2^z (mod\ p) \tag{2}$$

where z is an integer and $k \le z \le 2k$. Phase 2 completes the operation by taking *Phase* 1(a) and z as inputs and returning *MonInv*(a). However, it is often desirable to compute either the classical modular inverse of an integer, given by

$$ModInv(a) = a^{-1} (mod\ p) \tag{3}$$

or the Montgomery modular inverse of an integer already in the Montgomery domain. These can be achieved using the following two algorithms based on Kaliski's method [2]:

Algorithm 1: Kaliski ModInv
Input: a, p
Output: a^{-1} (mod p)
1. $r = Phase1(a) = a^{-1}\ 2^z\ (mod\ p)$;
2. for i = 1 to z loop
if r is even then $r = r/2$;
else $r = (r+p)/2$;
3. Return $r = a^{-1}\ (mod\ p)$;

Algorithm 2: Kaliski MonInv
Input: $a2^k$ (mod p), p, k, 2^{2k} (mod p)
Output: $a^{-1}\ 2^k$ (mod p)
1. $r = Phase1(a2^k) = a^{-1}\ 2^{-k}\ 2^z\ (mod\ p)$;
2. for i = 1 to z − k loop
if r is even then $r = r/2$;
else $r = (r+p)/2$;
3. $r = MonPro(r,\ 2^{2k}) = a^{-1}\ 2^k\ (mod\ p)$;
4. Return $r = a^{-1}\ 2^k\ (mod\ p)$;

The *MonPro* function at step 3 of algorithm 2 refers to the Montgomery modular multiplication [3] of two k-bit integers A, $B \in [0, p-1]$ given by

$$MonPro(A, B) = A\ B\ 2^{-k} (mod\ p) \tag{4}$$

Savas and Koç method: Savas and Koç [1] suggested that Montgomery multiplication can be used to replace the iterative loops in algorithms 1 and 2 and derived the following algorithms based on this. Here m is an integer multiple of the word size, w, of the computer system used and $m \ge k$. In this case, the output z from phase 1 is an integer satisfying $k \le z \le k+m$. Also, $R^2 = 2^{2m}\ (mod\ p)$ and the inputs to the MonPro function are assumed to be m-bit integers.

Algorithm 3: Savas/Koç ModInv
Input: a, p, m
Output: a^{-1} (mod p)
1. $r = Phase\ 1(a) = a^{-1}\ 2^z\ (mod\ p)$;
2. if $z > m$ then
$r = MonPro(r,\ 1) = a^{-1}\ 2^{z-m}\ (mod\ p)$;
$z = z - m < m$;
3. $r = MonPro(r,\ 2^{m-z}) = a^{-1}\ (mod\ p)$;
4. Return $r = a^{-1}\ (mod\ p)$;

Algorithm 4: Savas/Koç MonInv
Input: $a2^m$ (mod p), p, m, R^2
Output: $a^{-1}\ 2^m$ (mod p)
1. $r = Phase\ 1(a2^m) = a^{-1}\ 2^{-m}\ 2^z\ (mod\ p)$;
2. if $k \le z \le m$ then
$r = MonPro(r,\ R^2) = a^{-1}\ 2^z\ (mod\ p)$;
$z = z + m > m$;
3. $r = MonPro(r,\ R^2) = a^{-1}\ 2^z\ (mod\ p)$;
4. $r = MonPro(r,\ 2^{2m-z}) = a^{-1}\ 2^m\ (mod\ p)$;
5. Return $r = a^{-1}\ 2^m\ (mod\ p)$;

Algorithm 3 requires one or two Montgomery multiplications and algorithm 4 requires two or three multiplications. As discussed above, these algorithms were originally developed for software implementation. Clearly if these are used as the basis of a hardware system then two separate circuit architectures would be required.

Improved unified algorithm: In this Section we introduce a single, more efficient algorithm for computing both the classical modular inverse and the Montgomery modular inverse of an integer already in the Montgomery domain. Here, the output z from phase 1 is again the integer satisfying $k \le z \le 2k$, $R^2 = 2^{2k}\ (mod\ p)$ and the inputs to the MonPro function are k-bit integers. We have written the algorithm twice in order to illustrate its functionality.

Algorithm 5: Unified inverse algorithm
Input: **a**, p, k, **1**
Output: a^{-1} (mod p)
1. $r = Phase\ 1(a) = a^{-1}\ 2^z\ (modp)$;
2. if $z = k$ then
$r = MonPro(r,\ 1) = a^{-1}\ (mod\ p)$;
else
$r = MonPro(r,\ 2^{2k-z}) = a^{-1}\ 2^k\ (mod\ p)$;
$r = MonPro(r,\ 1) = a^{-1}\ (mod\ p)$;
3. Return $r = a^{-1}\ (mod\ p)$;
Input: **$a2^k$ (mod p)**, p, k, **R^2**
Output: $a^{-1}\ 2^k$ (mod p)
1. $r = Phase\ 1\ (a2^k) = a^{-1}\ 2^{-k}\ 2^z\ (mod\ p)$;
2. if $z = k$ then
$r = MonPro(r,\ R^2) = a^{-1}\ 2^k\ (mod\ p)$;
else
$r = MonPro(r,\ 2^{2k-z}) = a^{-1}\ (mod\ p)$;
$r = MonPro(r,\ R^2) = a^{-1}\ 2^k\ (mod\ p)$;
3. Return $r = a^{-1}\ 2^k\ (mod\ p)$;

Notice the two pairs of inputs highlighted in bold type. If one wishes to calculate the classical modular inverse then the pair (a, 1) is input to the algorithm, whereas the Montgomery modular inverse of an integer already in the Montgomery domain is calculated by inputting the pair ($a2^k$ (mod p), R^2) to the algorithm. Moreover, the unified inverse algorithm is more efficient than algorithm 4 in computing the Montgomery modular inverse of an integer already in the Montgomery domain. Algorithm 4 requires a maximum of three multiplications to calculate this whereas the new unified algorithm requires at most only two and thus at least a 33% saving in Montgomery multiplication operations is achieved. This algorithm is ideal for hardware (and indeed software) implementations, as a single circuit architecture can serve to compute both types of inverse.

FPGA hardware implementation: Algorithms 1 to 5 have been implemented on a Xilinx Virtex2 Pro XC2VP125 FPGA for demonstration and comparative purposes, using a 256-bit operand length.

The MonPro operations were performed using algorithm 6 [4] given below, where n is the k-bit modulus, $r = 2^k$, $rr^{-1} - nn\prime = 1$ and $r^{-1}r = 1 \ (mod\ n)$:

Algorithm 6: MonPro(A, B)
1. $t = A\ B$;
2. $u = (t + (t\ n'\ mod\ r)\ n)/r$;
3. if $u \geq n$ *then return* $u - n$ *else return* u;

All the additions and subtractions needed in algorithms 1 to 6 have been carried out using the fast carry look-ahead logic located on the Virtex2 Pro FPGAs. The ordinary multiplications required in algorithm 6 are performed using the 18×18-bit multipliers embedded within these FPGAs. Table 1 provides performance results for these implementations. These results indicate that using the unified inversion algorithm results in a speed up of 27.1 and 37.2%, respectively, when compared with algorithms 1 and 2. An overall reduction of 23.1% in silicon area usage is also achieved in this case, as the unified inversion algorithm can serve to calculate both the classical inverse and the Montgomery modular inverse of an integer already in the Montgomery domain (Table 1). There is not a significant speed-up by using the unified algorithm instead of algorithms 3 and 4. However, the reduction in silicon area usage in this case is 99.8%, for the reasons described earlier. Similar speed-ups and savings in source code/silicon area are attainable if the algorithms are implemented in software or alternative hardware media, such as modern ASIC devices. The unified inversion algorithm affords this because of its inherently less complicated structure.

Table 1: Performance results for 256-bit inversion architectures

Algorithm	Clock speed (MHz)	Clock cycles	Inversion time (μs)	CLB slices	Function (inverse type)	Speed-up by using algorithm 5	Area saved by using algorithm 5
1	57.75	1029	17.82	3,193	Classical	21.7%	23.1%
2	40.18	807	20.08	15,029	Montgomery	37.2%	
3	40.46	585	14.46	14,723	Classical	−1.2%	99.8%
4	40.68	619	15.22	14,844	Montgomery	4.0%	
5	40.04	586	14.64	14,800	Classical and Montgomery	N/A	N/A

Conclusions: We have described a novel, unified algorithm for classical and Montgomery modular inversion. This compares with two parallel algorithms [1] previously required (algorithms 3 and 4) and results in a saving of 33% in the number of Montgomery multiplication operations needed. This has been supported by FPGA implementations, which allow a direct comparison between the unified algorithm and Kaliski's (algorithms 1 and 2), and Savas and Koç's (algorithms 3 and 4) algorithms. A speed-up of 27.1 and 37.2% is achieved by using the unified algorithm instead of algorithms 1 and 2, respectively. An overall reduction of 23.1% in silicon area usage is also achieved in this case. A reduction in silicon area usage of 99.8% is attained by using the unified algorithm instead of algorithms 3 and 4. The algorithm described can also be used to provide similar speed-ups and savings in source code/silicon area in both software and ASIC implementations.

31 May 2004
Electronics Letters online no: 20045610
doi: 10.1049/el:20045610

C. McIvor, M. McLoone and J.V. McCanny (*The Institute of Electronics, Communications and Information Technology, The Queen's University of Belfast, Riddell Hall, 185 Stranmillis Road, Belfast BT9 5EE, Northern Ireland*)

E-mail: c.mcivor@ee.qub.ac.uk

References

1 Savas, E., and Koç, C.K.: 'The Montgomery modular inverse – revisited', *IEEE Trans. Comput.*, 2000, **49**, (7), pp. 763–766
2 Kaliski, B.S.: 'The Montgomery inverse and its applications', *IEEE Trans. Comput.*, 1995, **44**, (8), pp. 1064–1065
3 Montgomery, P.L.: 'Modular multiplication without trial division', *Math. Comput.*, 1985, **44**, pp. 519–521
4 Koç, C.K., Acar, T., and Kaliski, B.S.: 'Analyzing and comparing Montgomery multiplication algorithms', *IEEE Micro*, 1996, **16**, (3), pp. 26–33