Architectures for Unified Field Inversion with Applications in Elliptic Curve Cryptography

E. Savaş and Ç. K. Koç

Department of Electrical & Computer Engineering
Oregon State University
Corvallis, Oregon 97331
{savas,koc}@ece.orst.edu

ABSTRACT

We present two new inversion algorithms for binary extension and prime fields, which are slightly modified versions of the Montgomery inverse algorithm. An hardware architecture implementing these algorithms is also introduced. In our proposed architecture, the field elements are represented using a multi-word format which allows a scalable and unified architecture to operate in a broad range of precision. This hardware architecture can be used to obtain efficient implementations of elliptic curve cryptography primitives.

1. INTRODUCTION

The basic arithmetic operations (i.e. addition, multiplication, and inversion) in prime and binary extension fields, $GF(p)$ and $GF(2^n)$, have several applications in cryptography, such as decipherment operation of RSA algorithm [1], Diffie-Hellman key exchange algorithm [2], the Government Digital Signature Standard [3] and also elliptic curve cryptography [4, 5]. Recently, speeding up inversion operation in both fields has been gaining some attention since inversion is the most time consuming operation in elliptic curve cryptographic algorithms when affine coordinates are selected [6, 7, 8, 9, 10].

In this paper, we will give and analyze multiplicative inversion algorithms for $GF(p)$ and $GF(2^n)$ which allow very fast and area-efficient, unified and scalable hardware implementations. The algorithms are based on the Montgomery inverse algorithms given in [6].

2. THE MONTGOMERY INVERSION ALGORITHM

The following algorithm performs the Montgomery inversion in $GF(2^n)$. However, the Phase II of the algorithm is omitted since it is not relevant to this paper, and a similar algorithm is in [9].

Algorithm A
Input: $a(x)$ and $p(x)$, where $deg(a(x)) < deg(p(x))$
Output: $r(x)$ and k, where $r = a(x)^{-1}x^k \pmod{p(x)}$ and $deg(a(x)) \leq k \leq deg(p(x)) + deg(a(x)) + 1$

```
1:  $u(x) := p(x)$, $v(x) := a(x)$, $r(x) := 0$, and $s(x) := 1$
2:  $k := 0$
3:  while $(v(x)! = 0)$
4:    if $u(0) = 0$ then $u(x) := u(x)/x$, $s(x) := xs(x)$
5:    else if $v(0) = 0$ then $v(x) := v(x)/x$, $r(x) := xr(x)$
6:    else if $deg(u(x)) > deg(v(x))$ then
        $u(x) := (u(x) + v(x))/x$
        $r(x) := r(x) + s(x)$
        $s(x) := xs(x)$
7:    else $v(x) := (v(x) + u(x))/x$
        $s(x) := s(x) + r(x)$
        $r(x) := xr(x)$
8:    $k := k + 1$
9:  if $deg(r(x)) = deg(p(x))$ then $r(x) := r(x) + p(x)$
10: return $r(x)$ and $k$
```

The following properties are observed:
- If $deg(p(x)) > deg(a(x)) > 0$, then the degrees of intermediate binary polynomials $r(x)$, $s(x)$, $u(x)$, and $v(x)$ in the Montgomery inverse algorithm are always in the range $[0, deg(p(x))]$.
- If $p(x)$ is an irreducible polynomial, and $deg(p(x)) > deg(a(x)) > 0$, then $n+1 < k \leq deg(a(x))+n+1$.
- If $p(x)$ is an irreducible polynomial, and $deg(p(x)) > deg(a(x)) > 0$, then Phase I of Montgomery inverse algorithm for $GF(2^n)$ returns $a(x)^{-1}x^k \pmod{p(x)}$.

Additions and subtractions in the original algorithm are replaced with additions without carry in $GF(2^n)$ version of the algorithm. Since it is possible to perform addition (and subtraction) with carry and addition without carry in a single arithmetic unit, this difference does not cause a change in the control unit of a possible unified hardware implementation. On the other hand, the algorithm for $GF(2^n)$ differentiates from the original algorithm in Step 6, in which the degrees of $u(x)$ and $v(x)$ are compared. In order to have a unified architecture, we propose a slight modification in the original algorithm for $GF(p)$ which is given in the following section.

3. A VARIATION OF MONTGOMERY INVERSION ALGORITHM

We propose to modify Step 6 of the algorithm given in [6] in a such way that instead of comparing u and v, number of bits needed to represent them are compared. The proposed modifications can be seen in Step 6 and Step 7.a of the modified algorithm given below:

Algorithm B
Input: $a \in [1, p-1]$ and p
Output: $r \in [1, p-1]$ and k, where $r = a^{-1}2^k \pmod{p}$ and $n \leq k \leq 2n$

```
1:     $u := p$, $v := a$, $r := 0$, and $s := 1$
2:     $k := 0$
3:     while $(v > 0)$
4:         if $u$ is even then $u := u/2$, $s := 2s$
5:         else if $v$ is even then $v := v/2$, $r := 2r$
6:         else if $bitsize(u) > bitsize(v)$ then
               $u := (u - v)/2$, $r := r + s$, $s := 2s$
7:         else then
               $v := (v - u)/2$, $s := s + r$, $r := 2r$
7.a:           if $v < 0$ then $v := -v$, $s := -s$
8:         $k := k + 1$
9:     if $r < 0$ then
9.a:       if $r \leq -p$ then $r := r + p$
9.b:       return $r := -r$
10:    else
10.a:      if $r \geq p$ then $r := r - p$
10.b:      return $r := p - r$ and $k$
```

When corrections in Step 7.a are executed, the effect is multiplying both sides of the invariant by -1. Therefore, new invariant when $s < 0$ is given as $-p = us + vr$. While u and v remain to be positive integers, s and r might be positive or negative. Therefore, we need to alter the final reduction steps to bring r in the correct range, which is $[0, p)$. The range of s and r are $[-p, p]$ and $[-2p, 2p]$, respectively. As a result we need to use one more bit to represent s and r than in the original algorithm. The advantage of this version of the algorithm will be discussed in the next section.

4. HARDWARE ARCHITECTURE

Scalability of the arithmetic modules is important in cryptographic context since it allows to increase the key length when the need for more security arises without having to modify or redesign the cryptographic unit. The scalability of the inverter unit can easily be achieved by using shifter and adder units which handle only certain number of bits of the operands at a time. One addition (or shift) operation, therefore, in the corresponding field takes more than one clock cycle. The number of bits that the unit operate on is referred as *word* and its length can be determined or adjusted with respect to given area, speed or latency requirements.

The algorithms B and C can be implemented in a unified hardware architecture provided that a dual-field adder/subtractor (DFA/S) that operates in both fields is available. In order for the inverter unit to be scalable, The DFA/S is designed to handle words of finite number of bits at a clock cycle, therefore we call them word DFA/s(WDFA/S).

Except the final correctional steps (steps 9 through 11), the main loops of the Algorithm A and Algorithm B can be implemented in the same hardware unit. The only difference in the main loops of the two algorithms is that the Algorithm B has the extra Step 7.a. However, this extra step neither necessitates a major change in the circuitry nor introduces any extra clock cycle in the computation. Algorithm B replaces integer comparison operation of the original algorithm with just one bitsize comparison. In exchange for that, some of the intermediate variables take negative integer values. For example, the variables v and s may have to change sign in Step 7.a if the subtraction operation in Step 7 produce a negative result. Taking two's complement of these two variables may re-introduce the clock cycles we saved by eliminating integer comparison operation in Step 6 of the original algorithm [6]. On the other hand, When variable v turns out to be a negative number as a result of the subtraction in Step 7, we may keep it as negative in two's complement representation. In the next iteration in the loop, it can easily be seen that Step 5 or Step 6 is executed. Sign change of the variable may be performed at the same time as the subtraction operation in the subsequent Step 6.

On the other hand, the magnitudes of r and s cannot easily be determined. Therefore, we need to devise a method in order to avoid taking two's complement of s in Step 7.a. We propose to maintain one extra bit for each of the variables s and r which holds extra sign information for them. We call this extra sign bit as *correct sign* (CS) of the variable. These variables can be kept as negative (in two's complement representation) or positive, however, their real sign is determined by the value in correct sign bit. If their actual sign is different from the one in the correct sign bit, the sign must be flipped. On the other hand, taking two's complement when this happens is not desirable since we want to avoid the extra clock cycles it introduces. The actual and correct signs of a variable determine the way we execute the addition operation $x := r + s$ in Steps 6 and 7. Assuming that S_x and CS_x are the actual and correct sign of the variable x respectively, this operation is performed as in the following:

Algorithm C
Input: r, s, S_r, S_s, CS_r, and CS_s
Output: $x := r + s$, S_x, and CS_s

```
1:   if $S_r = CS_r$ and $S_s = CS_s$ then
1.a:     $x := s + r$ and $CS_x := S_x$
2:   else if $S_r = CS_r$ and $S_s = \overline{CS}_s$ then
2.a:     $x := r - s$ and $CS_x := S_x$
3:   else if $S_r = \overline{CS}_r$ and $S_s = CS_s$ then
3.a:     $x := s - r$ and $CS_x := S_x$
4:   else $S_r = \overline{CS}_r$ and $S_s = \overline{CS}_s$ then
4.a:     $x := s + r$ and $CS_x := \bar{S}_x$
```

5. COMPLEXITY ANALYSIS OF THE UNIFIED INVERTER

Assuming that we have two WDFA/S in our design, the total computation time of inversion in terms of total clock cycle count can be computed using the formula $T = k\cdot(e+1)$, where k is the iteration index in the main loop of the algorithms, $e = \lceil \frac{n+1}{w} \rceil$ is the number of words and w is the word length.

Based on these experimental values we calculated the estimated execution time in terms of number of clock cycles for inversion operation using word length 32. We summarized the results in Table 1. Table 1 also includes the clock cycle count estimates for the modular multiplication operation for the same precisions, which is assumed to be performed using unified and scalar Montgomery modular multiplication unit proposed in [11] with 7 pipeline stages and 32-bit word size. The ratio of inversion time to multiplication time, which is important in the decision whether affine or projective coordinates are to be employed in elliptic curve cryptography, is also included in the table. It is argued in [12] that for binary extension fields $GF(2^k)$ projective coordinates, which does not entail fast execution of inversion operation, perform better than the affine coordinates when inversion operation is more than 7 times slower than the multiplication operation. Similarly, our calculations show that this ratio is about 9 for prime field $GF(p)$. As can be observed in Table 1 the ratio stays lower than 7 for the precisions of interest to the elliptic curve cryptography.

Table 1: Estimated clock cycles for inversion and the ratio to the multiplication operation.

bitsize	Inversion	Multiplication	Ratio
160	1368	327	4.18
192	1911	398	5.00
224	2544	469	5.42
256	3276	526	6.23

In Figure 1 and Figure 2, hardware realizations of the operations $(u - v)/2$ and $r + s$ are shown, respectively. In Figure 1, the building block A simply separates the least significant bit from the rest of the result bits, which are to be kept in the latch one clock cycle in order to perform shift operation. In the next clock these bits are combined with the least significant bit of the current result, which is placed in the most significant position of the final resulting word, in block B. The block C of the Figure 1, is used to connect the register outputs to the correct input of the adder/subtractor unit. The circuit in Figure 2 performs two operations: $r + s$ and $2r$(or $2s$). The register content, which is to be shifted left by one bit, is available at the output of block D. The block D is also used to connect the register outputs to the correct input of the adder/subtractor unit. Blocks A and B are used to shift a word in each clock cycle. Block C directs the results of the two operation ($r + s$ and $2r$(or $2s$)) to the appropriate registers.

Figure 1: Hardware realization of $(u - v)/2$.

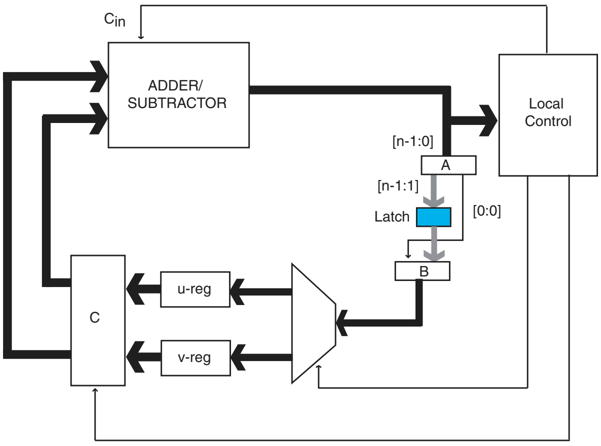
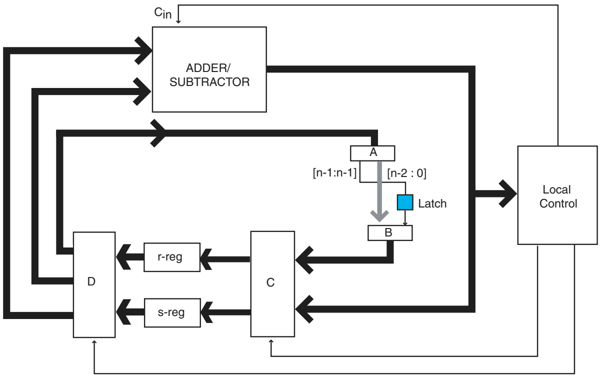


Figure 2: Hardware realization of $r + s$.



6. REFERENCES

[1] J.-J. Quisquater and C. Couvreur, “Fast decipherment algorithm for RSA public-key cryptosystem,” *Electronics Letters*, vol. 18, no. 21, pp. 905–907, Oct. 1982.

[2] W. Diffie and M. E. Hellman, “New directions in cryptography,” *IEEE Transactions on Information Theory*, vol. 22, pp. 644–654, Nov. 1976.

[3] National Institute for Standards and Technology, “Digital signature standard (DSS),” *Federal Register*, vol. 56, pp. 169, Aug. 1991.

[4] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan. 1987.

[5] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, Boston, MA, 1993.

[6] B. S. Kaliski Jr., “The Montgomery inverse and its applications,” *IEEE Transactions on Computers*, vol. 44, no. 8, pp. 1064–1065, Aug. 1995.

[7] R. Schroeppel, H. Orman, S. O’Malley, and O. Spatscheck, “Fast key exchange with elliptic curve systems,” in *Advances in Cryptology — CRYPTO 95*, D. Coppersmith, Editor, Lecture Notes in Computer Science, No. 973, pp. 43–56, Springer, Berlin, Germany, 1995.

[8] T. Kobayashi and H. Morita, “Fast modular inversion algorithm to match any operand unit,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82–A, no. 5, pp. 733–740, May 1999.

[9] E. Savaş and Ç. K. Koç, “The Montgomery modular inverse - revisited,” *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 763–766, July 2000.

[10] M. A. Hasan, “Efficient computation of multiplicative inverses for cryptographic applications,” Technical Report CORR 2001–03, Centre for Applied Cryptographic Research, University of Waterloo, Canada, 2001.

[11] E. Savaş, A. F. Tenca, and Ç. K. Koç, “A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m),” in *Cryptographic Hardware and Embedded Systems - CHES 2000*, Ç. K. Koç and C. Paar, Editors, Lecture Notes in Computer Science No. 1965, pp. 281–296, Springer, Berlin, Germany, 2000.

[12] J. López and R. Dahab, “Fast multiplication on elliptic curves over GF(2^m) without precomputation,” in *Cryptographic Hardware and Embedded Systems*, Ç. K. Koç and C. Paar, Editors, Lecture Notes in Computer Science, No. 1717, pp. 316–325, Springer, Berlin, Germany, 1999.