Analysis of Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic

Ahmed Al Faresi
Electrical and Computer Engineering
Oregon State University
Corvallis, Oregon 97331–4501
Email: alfaresi@engr.orst.edu

***Abstract*— We present a custom class of primes using modular scaling that facilitate efficient finite filed operations. In addition we introduce an inversion algorithm that utilizes such special modulus. This inversion algorithm is an improvement on the available Euclidean algorithm, incorporating the use of the scaled modulus and proving to be of high performance and efficiency for hardware implementation. Using both the scaled modulus and the inversion algorithm we define a cryptographic processor for Elliptic curves Cryptography (ECC). This processor offers a superior performance in terms of area, power and speed.**

I. Introduction

The applications of Modular arithmetic in cryptography are endless. Many cryptographic schemes like the RSA algorithm [1], Diffe-helman key exchange algorithm [2], the Digital signature schemes [3] and elliptic curve cryptography [4], utilize modular arithmetic. The implementation of certain schemes requires the arithmetic operations modulo the product of two large primes i.e. n=pq. Others schemes like Diffe-Helman and El-Gamal are based on arithmetic of integers modulo a large prime p. In Elliptic Curve Digital Signature Algorithm (ECDSA) the arithmetic operations are modular operations with respect to a large prime modulus $GF(p)$ or polynomial arithmetic modulo a high degree irreducible polynomial defined over the $GF(2^k)$. The key for efficient implementation of ECDSA over $GF = 2^k$ is to choose irreducible polynomials that allow for efficient modular reduction. By doing so we reduce the complexity, power consumption and low speed of the implementation process. To date, the best irreducible polynomial obtained is either a trinomial or an equally-space polynomial (ESP). Unfortunately, there exist only a few irreducible ESPs in the range of interst of most applications, e.g., error-correcting code, computer algebra, and elliptic curve cryptography [5]. Therefore less efficient trinomials or pentanomials are used instead. However such less efficient polynomials result in extra additons and alignment adjustment rendering the implementation to be poor. A solution to such a problem is to use low hamming weight polynomials with a special modulus. To achieve such desired modulai equivalent to the low hamming weight we introduce a modulai of Mersenne form using modulus scaling. This in turn allowed for the development of an inversion algorithm that utilized the modulai to push the margin and create very efficient inversion hardware resulting in a very fast, area-efficient, low power scalable hardware implementation.

II. Mathematical Background

Finite filed defined over the $GF = 2^k$. Arithmetic is fundamental to the implementation of a number of modern cryptographic systems and schemes of certain cryptographic systems. Most arithmetic operations, such as exponentiation, inversion, and division operations, can be carried out using just a modular multiplier [6]. Simple modular multiplication is implemented by first computing the product of the two operands, $c = a.b$, the result is then reduced using the modulus $x = c \pmod p$. Since reduction would mean a division operation, and that is to be avoided, certain modulai have been proposed to alleviate such a problem. Of such modulai is a Mersenne Prime given in the form $2^k c$ with $log_2 c < [k/2]$, where k is an integer for which $0 < |k| < 2^{[m/2]}$ If $c = 1$, then p is a Mersenne Prime and k must necessarily be a prime. If n is positive integer less than p^2, then can be written $n = u.2^{2k} = a.2^k + b$, Where $u = 0$ or 1 and a and b are nonnegative integers less than 2^m then $n = u.c^2 + ac + b \pmod p$. Repeating this substitution a few times will yield n modulo p. This method requires a small number of additions and subtractions rather than the usual division step. Unfortunately Mersenne primes and primes of the from $2^k + 1$ are scarce. For degrees up to 1000 no primes of the form $2^k + 1$ and only two Mersenne primes $2^{521} - 1$ and $2^{607} - 1$ exist.[8]. However for ECDSA such primes are too large and are unsuitable. Therefore an alternative solution is to use primes of the form $2^k - 3$.

III. Modulus Scaling

Modular scaling plays a vital role in cryptographic implementations since it allows for a key bit increase if need be without having to change or reconfigure the cryptographic design the earliest works on modular scalability where introduced by Walter. The basic idea is to obtain a modulus scalable in the higher order bits. This is achieved by scaling a prime modulus to obtain a new one $m = ps$ where p is the original modulus scaled to m. Now for a given integer a reduced by the new modulus m will give a result congruent to a:

$(a \pmod m) \pmod p \equiv a \pmod p$ When a scaled modulus is used, residues will be in the range$[m-1,0] = [s.p-1,0]$. The number is not fully reduced and essentially we are using a redundant representation where an integer is represented using $[log_2 s]$ more bits than necessary. Therefore it will be necessarily that the final result be reduced by p to obtain a fully reduced representation. Now in order to scale a modulus and obtain one of low-hamming wait we need to find a small suitable constant to scale the prime p. Its worth noting that if a random pattern appears in a modulus than a low hamming weight optimization will not be possible. Two heuristics are presented that form a basis for efficient on the fly scaling[8]:

Heuristic 1 if the base B representation of an integer contains a series of repeating digits, scaling the integer with the largest possible digits, produces a string of repeating zero digits in the scaled and recoded integer.

Assume that base B representation contains a repeating value D. Then we use the scaling factor $s = B-1$ to compute m. When a string of repeating D-digits is multiplied with the scaling factor, and written in base B we get:
$(DDDD....DDD)_B.(B-1) = (DDDD...DDD0)_B - (DDDD....DDD)_B = (D000...000\bar{D})_B$.

The bar over the least significant digit denotes a negative valued digit.

Heuristic 2 Given a modulus containing repeating D-digits in base B representation, if B-1 is divisible by the repeating digit, then the modulus can be efficiently scaled by the factor $\frac{B-1}{D}$.

As earlier the heuristic is verfied by multiplying a string of repeating digits with the scaling factor and then be recoding. [savas].
$(DDDD....DDD)_B.\frac{B-1}{D} = ((B-1)(B-1)(B-1)...(B-1)_B = (1000...0\bar{1})_B$.

IV. Inversion Algorithm

Elliptic curve cryptography relies on efficient algorithms for finite filed arithmetic. For instance, the elliptic curve digital signature algorithm requires efficient addition, multiplication and inversion in the finite fields of sizes larger than 2^{160}. This poses a significant problem in embedded systems where computational power is quite limited and public-key operations are unacceptably slow[7]. An efficient way to calculate multiplicative inverses is to use binary extended Euclidean based Algorithms. One such efficient inversion algorithm is the Montgomery. However this algorithm uses only Montgomery arithmetics and renders to be unsuitable for a special modulai. There is however an Algorithm proposed for Mersenne primes of the form 2^q-1.[8]

Algorithm A-modified for divison with scaled modulus
Input: $a \in [1, p-1]$,p,and$q$where$p$ is prime and $p = 2^q - 1$
Output: $b \in [1, p-1]$ where $b = a^{-1} \pmod p$

1: $(b, c, u, v) := (1, 0, a, p)$;
2: Find e such that $2^e \parallel u$
3: $u := u/2^e$; //shift of trailing zeros
4: $b := \mp(2^{q-e}b) \pmod m$; //circular left shift
5: if $u = 1 returnb$;
6: $(b, c, u, v) := (b + c, b, u + v, u)$;
7: go to step 2

The above algorithm has been modified . This simple modification saves one multiplication in elliptic curve operations. The Algorithm A modified is shown below:

Algorithm A-modified for divison with scaled modulus
Input: $a \in [1, m-1]$,$d \in [1, m-1]$,m,and$q$where$m = 2^q \pm 1$
Output: $b \in [1, m-1]$ where $b = d/a \pmod m$

1: $a := a. \cdot s \pmod m$
2: $(b, c, u, v) := (d, 0, a, m)$;
3: Find e such that $2^e \parallel u$
4: $u := u/2^e$; //shift of trailing zeros
5: $b := \mp(2^{q-e}b) \pmod m$; //circular left shift
6: if $u = sreturnb$;
7: $(b, c, u, v) := (b + c, b, u + v, u)$;
8: go to step 3

Inversion algorithms efficiency is measured by the number of iterations k. To show that Algorithm A is efficient in terms of iteration number, we compared its distribution for k against that of a Montgomery inversion algorithm. We computed the inverse of 1000 randomly chosen integers modulo $m = 2^{167} + 1$ using Algorithm A. Since $p = m/3$ is a 166-bit prime we repeated the same experiment with the Montgomery inversion algorithm using p and we depicted the two distributions in Figure 1. Besides having much easier operations in each iteration one can easily observe the average number of iterations of Algorithm A is slightly lower than the number of iterations of the Montgomery inversion algorithm.

V. Elliptic Curve Architecture

In developing the arithmetic architecture we primarily focused on finding the minimal circuit to implement the Algorithm A efficiently. Since the architecture is build around the idea of maximizing hardware sharing among operations, the multiplication, squaring and addition operations are all achieved by the same arithmetic core. The simplicity of Algorithm A and scaled arithmetic allows us to accomplish all operations using only a few small state machines. The arithmetic unit is depicted in Figure 2.

What follows is an outline for the implementation of basic arithmetic operations as follows:

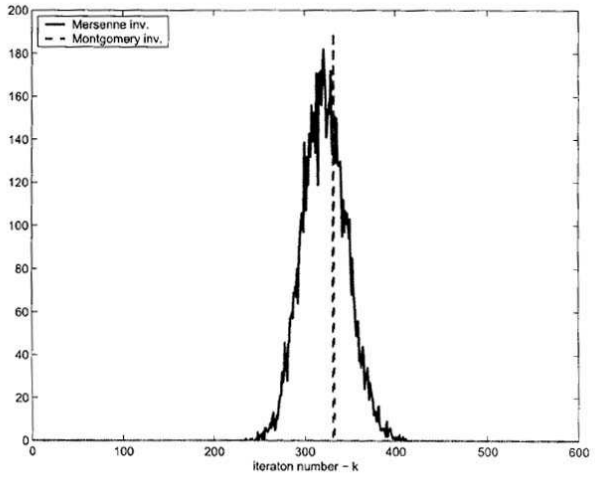

Fig. 1. Distribution of k in Algorithm A and the Montgomery inversion algorithm

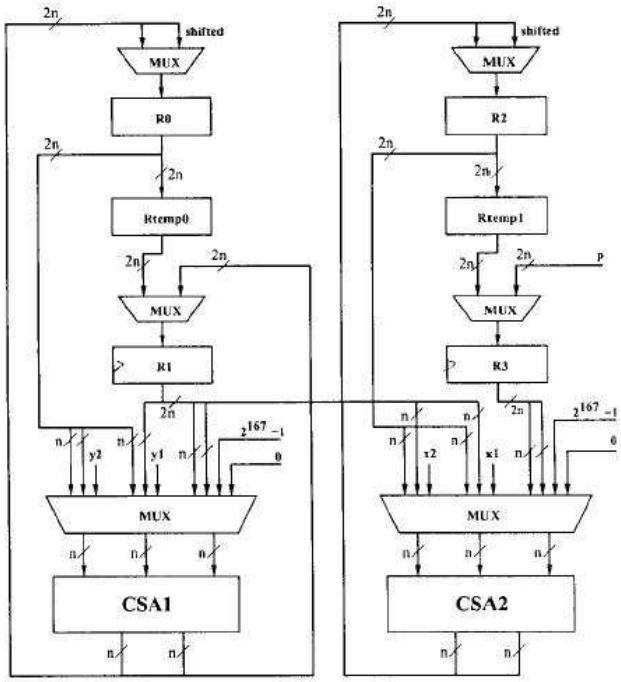

Fig. 2. Block diagram of the arithmetic unit

- *Modulo Reduction*: Since the hardware works for $m = 2^{167} + 1$, 168 bit registers would be sufficient. However, we used an extra bit to detect when the number becomes greater than m. If one of the left-most bits of the number (carry or sum) is one, the number is reduced modulo m.
  $2^{168} = 2.(2^{167} + 1) - 2 = 2m - 2 = m - 2 \pmod m$.
  Hence the reduction is achieved by subtracting 2^{168} (or simply deleting this bit) and adding $m - 2 = (11...11111)_2$ (167 bits) to the number. If both of the leftmost bits are 1 then :
  $2.(2^{168}) = 4.(2^{167}+1)-4 = 4m-4 = m-4 \pmod m$.

  Therefore $m - 4 = (111...11101)_2$ (167 bits) has to be added to the number and both of the leftmost bits are deleted.

- *Subtraction*: Suppose k is a 168 bit number which we want to subtract from another number modulo m. The bitwise complement of k is found as
  $k\prime = (2^{168} - 1)k = 2.(2^{167} + 1) - 3 - k = -3 - k \pmod m$. Thus $-k = k\prime + 3 \pmod m$. This means to subtract k from a number we simply add the bitwise complement of k and 3 to the number. It is worth noting that our numbers are kept in a carry save represntaion , there are two 168-bit numbers represnting k.

- *Multiplication*: We serialize our multiplication algorithm by processing one bit of one operand and all bits of the second operand in each iteration. The standard multiplication algorithm had to be modified to make it compatible with the carry save representation. Due to the redundant representation, the value of the leftmost bit of the multiplier is not known. Hence the left to right multiplication algorithm many not be used directly. We prefer to use the right to left multiplication algorithm. There are 3 registers used for the multiplication: R0 (multiplicand), R1 (product) and R2 (multiplier). The multiplication algorithm has three steps:

  1. Initialization: the control circuit does this step. The multiplicand is loaded to R0, the multiplier is loaded to R2 and R1 is reset.
  2. Addition: This step is only done when the rightmost bit of register R2 is 1. The content of register R0 is added to R1.
  3. Shifting: The multiplier has to be processed bit-by-bit starting from right. We do this by shifting register R2 to the right in each iteration of the multiplication. Since the register R2 is connected to the comparator, the algorithm terminates after this step if the number becomes 0 else the algorithm continues with Step 2. Note that no counters are used in the design. This eliminates potential increases in the critical path delay. The multiplicand needs to be doubled in each iteration as well. This is achieved by shifting register R0 to the left. This operation is performed in parallel with shifting R2, so no extra clock cycles are needed. However shifting to the left can cause overflow. Therefore, the result needs to be reduced modulo m if the leftmost bit of the register R0 is 1.
- Inversion: To realize the inversion operation there are four registers used to hold b,c,u and v, two temporary registers are used for the addition of two numbers in carry-save

architecture. Two carry-save adders, multiplexers and comparator architecture are also utilized. The inversion algorithm shown in Algorithm A has 5 steps, which are depicted in Figure 3.

1: Intialize all registers $(b, c, u, v) \leftarrow (1, 0, a, m)$
2: Shift off all trailing zeros and rotate b $u \leftarrow u >> e \; b \leftarrow b >> e \pmod m$
3: Check terminate condition if $u = s$ return b
4: Update variables $(b, c, u, v) \leftarrow (b + c, b, u + v, u);$
go back to step 2

Figure 3: Hardware algorithm for inversion.

VI. Results

The presented architecture was developed into Verilog modules and synthesized using Synopsys tools Design compiler and Power Compiler. The resulting architecture was synthesized for three operating frequencies. The implementation results are shown in table 1.

Table 1: Implementation Results.

OP. Freq (MHz)	Area (gates)	Power (mW)	Avg. Delay (msec)
20	30,333	0.99	31.9
100	30,443	4.34	6.3
200	34,390	9.89	3.1

As depicted in the table the area varies around 30 K gates. The circuit achieves its intended purpose by consuming only 0.99mW at 20 Mhz. In this mode the point multiplication takes about 31.9 msec. Although this is not very fast, this operating mode might be useful for interactive applications with strict power requirements. The design operates serially in one operand leading to a lower critical paths and much smaller area in the design. Which translates to lower power at a balanced frequency not so high and not so low[8].

VII. Conclusion

We demonstrated that scaled arithmetic, which is based on the idea of transforming a class of primes into special forms that enable efficient arithmetic, could be used in elliptic curve cryptography. Implementation results show that the use of scaled modulai in elliptic curve cryptography offers an efficient performance in terms of power. The use of the special modulai allowed for a superb inversion implementation. This itself eliminated the need for projective coordinates that required prohibitively a large amount of extra storage. The fact that the same data path (i.e. arithmetic core) is used for the filed operations leads to a very small chip area. Elliptical curve cryptography offers a lot of promise in terms of security and power requirement then the any other present cryptosystems. More research is needed in this field for more better and effective implementation[8]

References

[1] W. Diffe and M. E. Hellman., "New directions in cryptography." *IEEE Transactions on information Theory*, pp. 22:644–654, November 1976.

[2] N. I. for Standards and Technology, "Digital signature standard (dss)." *Federal Register*, p. 56:169, Aug 1991.

[3] N. Koblitz, "Elliptic curve cryptosystems." *Mathematics of Computation*, pp. 48(177):203–209, Jan 1987.

[4] A. J. Menezes., "Elliptic curve public key cryptosystems," *Kluwer Academic Publishers, Boston, MA*, 1993.

[5] F. R. Henriquez and C.K.koc., "Parallel multipliers based on special irreducible pentanomials," *IEEE Transactions on Computers*, Aug 2002.

[6] H.-S. Kim., "Efficient systolic architecture for modular multiplication over gf(2)," *PARA'04 State-of-the-Art in Scientific Computing*, June 2004.

[7] S. Baktir, "Efficient algorithms for finite fields, with applications in elliptic curve cryptography," Master of Science, Worcester Polytechnic Institute, 2003.

[8] B. Sunar and E. Savas, "Low-power elliptic curve cryptography using scaled modular arithmetic." IEEE, March 2 2004.