

A highly efficient modular Multiplication Algorithm for Finite Field Arithmetic in $\mathbb{GF}(P)$

Rainer Blümel^b, Ralf Laue^a, and Sorin A. Huss^a

^aIntegrated Circuits and Systems Laboratory, Department of Computer Science
Technische Universität Darmstadt, Germany
{laue|huss}@iss.tu-darmstadt.de

^bcv cryptovision GmbH, Gelsenkirchen, Germany
rainer.bluemel@cryptovision.com

Abstract

The performance of today's public key cryptosystems depends mainly on the efficiency of the underlying finite field arithmetic, especially the modular multiplication. In this work we propose a new modular multiplication algorithm for $\mathbb{GF}(P)$ which has a complexity of only $n^2 + 7n$. To our knowledge this is superior to the complexity values of any other modular multiplication algorithm for $\mathbb{GF}(P)$.

1 Introduction

Most common public key algorithms – particularly RSA[10] and the emerging elliptic curve cryptography (ECC)[8] – depend on modular arithmetic, where the most critical operation is modular multiplication. Therefore, improving the efficiency of modular multiplication will significantly enhance the efficiency of many cryptographic algorithms.

As the bit length of numbers typically used in cryptographic applications is counted in hundreds of bits, operations between these multi-precision numbers have to be mapped to operations between single digits. Of these digit operations the most expensive one is the digit multiplication. Thus, we will take the number of digit multiplications as complexity metric.

Modular multiplication is the multiplication in a finite field like $\mathbb{GF}(P)$ or $\mathbb{GF}(2^m)$, where the operations are executed modulo a prime number or a prime polynomial, respectively. In this work we present a new modular multiplication algorithm[†] optimized for $\mathbb{GF}(P)$ with a complexity of $n^2 + 7n$. For comparison the widely used Montgomery multiplication has a complexity of $2n^2 + 2n$. The rest of this paper is structured as follows: The next section points out relations between the proposed algorithm and other modular multiplication algorithms. Section 3 is the main part of this work, where the algorithm is described in detail. Section 4 explains a prototype implementation with its chosen parameters and constraints. Section 5 presents the results and a comparison with a corresponding implementation using the Montgomery multiplication. Section 6 finally concludes this work with some closing remarks.

This work has been partly supported by the research program SicAri of BMBF (German Federal Ministry of Education and Research).

[†]for which cryptovision GmbH has already filed a patent application

2 Related Work

This new algorithm combines ideas from different modular multiplication algorithms in a new way: It relies on an idea derived from the Karatsuba algorithm[4] to gain an accelerated multiplication algorithm applicable to general finite fields. This was used before in the MSK-algorithm from [2], which is only applicable to $\mathbb{GF}(2^m)$. Contrary to the Karatsuba scheme the proposed algorithm uses no recursion. Although this leads to a higher computational effort, it provides both a better flexibility and less overhead than the original Karatsuba algorithm and still retains a part of its computational advantage compared to standard multiplication techniques.

Furthermore, the new algorithm does not execute multiplication and reduction phase successively, but interleaves both phases similar to the Montgomery multiplication [9]. Thus, by choosing an adequate operation order the number of necessary memory accesses is decreased.

For the reduction we adapt ideas similar to the Barrett reduction[7]: The division needed in the reduction phase of our algorithm is substituted by a multiplication with the reciprocal – here denoted *Look Ahead*. But in contrast to the Barrett reduction our algorithm executes a partial reduction in each step resulting in several but shorter multiplications. Computing the reduction multiplication – needed in addition to the division – with the accelerated multiplication scheme permits a further improvement for the proposed algorithm.

Compared to the Montgomery multiplication our algorithm exhibits less computational effort by using the accelerated multiplication scheme for both phases. But because of the *Look Ahead*-step it has a more complex flow. Note that this results in our algorithm not being superior to Montgomery, if the multiplication is not the most expansive digit operation. Similar to the Montgomery multiplication it can in theory be used for both $\mathbb{GF}(2^m)$ and $\mathbb{GF}(P)$. However, for $\mathbb{GF}(2^m)$ one should prefer the MSK-algorithm, because it is hard to integrate the special prime polynomials usually used for $\mathbb{GF}(2^m)$.

3 A novel modular multiplication algorithm

Multi-precision numbers are divided into digits of b bits each and are denoted by uppercase letters in the sequel. A single digit is represented by a lowercase letter with an index indicating its position. Thus, the multi-precision number X with a length of n digits can be represented as

$$X = x_{n-1} \cdot 2^{(n-1)b} + x_{n-2} \cdot 2^{(n-2)b} + \dots + x_1 \cdot 2^b + x_0.$$

Operations on multi-precision numbers are denoted by the circled version of the usual operation sign (e.g., \odot for multiplication and \oplus for addition). Operations on single digits are denoted by the usual operation sign (e.g., \cdot for multiplication and $+$ for addition). The multi-precision numbers are either elements of a ring or of the finite field $\mathbb{GF}(P)$. In the latter case all calculations are reduced modulo the prime number P . Although we do not consider other finite fields in the following, the new multiplier should be easily extendable to these fields, as we do not require any special properties.

3.1 Basic algorithm

The proposed algorithm relies on an improved multi-precision multiplication formula used for the multiplication and the reduction phase, thus accelerating both. Through interleaving both phases the number of necessary memory accesses is considerably decreased.

The Karatsuba algorithm[4] is based on the fact that it is possible to multiply two numbers with 2 digits each exploiting only 3 digit multiplications:

$$\begin{aligned} X \odot Y &= (x_1 \cdot 2^b + x_0) \cdot (y_1 \cdot 2^b + y_0) \\ &= x_1 \cdot y_1 \cdot 2^{2b} + ((x_1 + x_0) \cdot (y_1 + y_0) - x_1 \cdot y_1 - x_0 \cdot y_0) \cdot 2^b + x_0 \cdot y_0 \end{aligned}$$

If this technique is used in a recursive manner (i.e., the digits x_0, x_1, y_0 and y_1 will be split again and so on), the Karatsuba multiplication has a complexity of $O(n^{\log_2 3}) \approx O(n^{1.585})$. But it is hard to implement this algorithm efficiently, because the recursion requires a considerable overhead and the operands of the middle sum are considerably longer than digit length on lower recursion levels. The basic idea of the Karatsuba multiplication is used to reduce the number of digit multiplications needed for one multi-precision multiplication: For the multi-precision multiplication step we combine two digit multiplications into one digit multiplication of the form $(x_i + x_j) \cdot (y_i + y_j)$. This results in the following formula for accelerated multiplication:

$$\begin{aligned} X \odot Y &= (x_{n-1} \cdot 2^{(n-1)b} + x_{n-2} \cdot 2^{(n-2)b} + \dots + x_1 \cdot 2^b + x_0) \odot \\ &\quad (y_{n-1} \cdot 2^{(n-1)b} + y_{n-2} \cdot 2^{(n-2)b} + \dots + y_1 \cdot 2^b + y_0) \\ &= \sum_{i=n-1}^1 \sum_{j=0}^{i-1} (x_i + x_j) \cdot (y_i + y_j) \cdot 2^{(i+j)b} \\ &\quad + \sum_{i=n-1}^0 x_i \cdot y_i \left(- \sum_{j=i+1}^{n-1} 2^{(i+j)b} + 2^{2ib} - \sum_{j=0}^{i-1} 2^{(i+j)b} \right). \end{aligned} \quad (1)$$

For $n = 2$ this is identical to the Karatsuba scheme, but recursive techniques are not applied. With this formula it is possible to execute a multi-precision multiplication (without reduction) with only $n(n+1)/2$ digit multiplications. From a mathematical point of view Equation (1) is equivalent to the MSK-algorithm in [2]. Note that the complexity is still quadratic, but significantly better than that of the well known School-algorithm[7]. Table 1 shows the complexity for a few values of n . Note furthermore, that Equation (1) requires the ability to multiply numbers with a bit-length of $b+1$, i.e., numbers which are 1 bit larger than one digit.

n	School-algorithm	accelerated multiplication	recursive Karatsuba
2	4	3	3
3	9	6	—
4	16	10	9
8	64	36	27
16	256	136	81

Table 1: Complexity comparison in terms of multiplications

In the following we use the above given multi-precision multiplication for accelerating the reduction phase, too: The product of $X \odot Y$ has to be reduced by a multiple of the prime number P , so that the result is positive and smaller than P . We denote this factor by Z , which has to fulfill $0 \leq X \odot Y \ominus P \odot Z < P$. Assuming that Z is known, it is possible to use the accelerated

multiplication for $P \odot Z$ (here in a different order).

$$\begin{aligned}
P \odot Z &= (p_{n-1} \cdot 2^{(n-1)b} + p_{n-2} \cdot 2^{(n-2)b} + \dots + p_1 \cdot 2^b + p_0) \odot \\
&\quad (z_{n-1} \cdot 2^{(n-1)b} + z_{n-2} \cdot 2^{(n-2)b} + \dots + z_1 \cdot 2^b + z_0) \\
&= \sum_{i=n-2}^0 \sum_{j=i+1}^{n-1} (p_i + p_j) \cdot (z_i + z_j) \cdot 2^{(i+j)b} \\
&\quad + \sum_{i=n-1}^0 p_i \cdot z_i \left(- \sum_{j=i+1}^{n-1} 2^{(i+j)b} + 2^{2ib} - \sum_{j=0}^{i-1} 2^{(i+j)b} \right) \tag{2}
\end{aligned}$$

Now one can combine both the multiplication phase (Equation (1)) and the reduction phase (Equation (2)) – thus interleaving them – to

$$\begin{aligned}
X \odot Y \ominus P \odot Z &= \\
&\sum_{i=n-1}^0 \left\{ \sum_{j=0}^{i-1} (x_i + x_j) \cdot (y_i + y_j) \cdot 2^{(i+j)b} - \sum_{j=i+1}^{n-1} (p_i + p_j) \cdot (z_i + z_j) \cdot 2^{(i+j)b} \right. \\
&\quad \left. + (p_i \cdot z_i - x_i \cdot y_i) \left(\sum_{j=i+1}^{n-1} 2^{(i+j)b} - 2^{2ib} + \sum_{j=0}^{i-1} 2^{(i+j)b} \right) \right\}.
\end{aligned}$$

Sorting the calculations according to the order of the digits we get

$$\begin{aligned}
X \odot Y \ominus P \odot Z &= \\
&\sum_{i=n-1}^0 \left\{ \sum_{j=0}^{i-1} ((x_i + x_j) \cdot (y_i + y_j) + p_i \cdot z_i - x_i \cdot y_i) \cdot 2^{(i+j)b} \right. \\
&\quad - (p_i \cdot z_i - x_i \cdot y_i) 2^{2ib} \\
&\quad \left. + \sum_{j=i+1}^{n-1} (p_i \cdot z_i - x_i \cdot y_i - (p_i + p_j) \cdot (z_i + z_j)) \cdot 2^{(i+j)b} \right\}. \tag{3}
\end{aligned}$$

Equation (3) reveals that it is possible to execute the algorithm efficiently, if z_i can be determined before the i th step of the main loop. For this determination of z_i the *Look Ahead*-mechanism is needed. As a precise evaluation would invalidate all complexity gains, this mechanism should only execute a partial calculation, resulting in an estimated z_i . The value of $(p_i \cdot z_i - x_i \cdot y_i)$ can be calculated once at the beginning of each step of the main loop, so only one multiplication has to be executed at each step of the inner loop.

In the following we set $r_i = (p_i \cdot z_i - x_i \cdot y_i)$ and use the variable A as an intermediate result to sum up all terms of the inner loops.

3.2 *Look Ahead* - procedure

The basic idea of the *Look Ahead* (from now on abbreviated as LA) is to pre-evaluate the terms for the highest digits in the next step of the main loop. In the most simple case it computes $A + (r_i - (p_i + p_{n-1}) \cdot (z_i + z_{n-1})) \cdot 2^{(i+n-1)b}$, of course with the exception of the term containing

the unknown z_i . Because P is known, it is easily possible to calculate an approximate value z_i by means of a partial division.

As above mentioned the LA performs only a partial calculation of each step of the main loop because of efficiency reasons. This may cause missing carry computations resulting in small errors in the values of z_i . The accuracy of the estimation of z_i depends of the number of digits taken into account. The above example uses only the digit $n+i-1$ for LA, therefore we call it One-level-LA. We have also experimented with pre-evaluating the two (digits $n+i-1$ and $n+i-2$) or three highest ($n+i-1$, $n+i-2$ and $n+i-3$) digits. We will denote these as Two- and Three-level-LA, respectively.

Exceptions from the general LA formula have to be taken into account for the calculation of some of the first z_i . To compute z_{n-1} in the case of One-level-LA we have to multiply the highest digits of X and Y and to divide the product by the highest digit of P . For higher level LA the computations of z_i with low values of i also change slightly. It is easy to extract the exact formulas from Equation (3).

To circumvent the necessary division, which is a very costly operation, we use a multiplication with the reciprocal of the highest digits of P followed by a bit shift (similar to the Barrett reduction[7], see next section for an example). As this reciprocal only depends on P , it can be handled like a constant value. It can be calculated by $q_{rec} = \frac{2^{2c}}{q}$, where q denotes the first c bits of P and c is a parameter for adjusting the needed accuracy. A larger c leads to a better accuracy. The bit length of q_{rec} is roughly equal to c and the number of digits of q_{rec} affects the number of needed digit multiplications. Therefore, c should be chosen as a multiple of b , because partly filled digits of q_{rec} still need complete multiplications while achieving only the accuracy for the valid bits.

Although one can chose parameters resulting in high accuracy, small errors will still occur. Basically there are two possibilities to handle inaccurate z_i :

- It is accepted that z_i can be over-long, i.e., it does not need to fit into one digit. This leads to a redundant representation for Z and inaccuracies during the estimation of z_i can be compensated by z_{i-1} . This situation has the disadvantage, that the operations used for implementing the algorithm have to be able to cope with over-long digits.

Note that this case also promises some resistance against SCA[5], because it is possible to chose the lower bits of z_i randomly and the calculation flow is always identical.

- z_i must always fit into one digit. Consequently, any inaccuracy in the estimation of z_i cannot be accounted for during the estimation of z_{i-1} . The solution is to compute a correction in form of a partial reduction, which can be executed at any time after an inaccuracy has been detected. Because such corrections generate a significant amount of overhead the accuracy of the z_i should be high, which means that the probability for a correction is low. If the probability for a correction is low enough, the corrections can be ignored for complexity considerations.

3.3 Look Ahead - application example

Algorithm 1 denotes in pseudo-code one possible realization of Equation (3) with no over-long z_i . The check for the correction is done directly after the LA.

For this example we will use the Two-level-LA: The first step of it is the calculation of \tilde{A} . For z_{n-1} \tilde{A} is given by

$$\tilde{A} = (x_{n-1} \cdot y_{n-1}) \cdot 2^b + (x_{n-1} \cdot y_{n-2} + x_{n-2} \cdot y_{n-1}).$$

For the general case the calculation of z_i it is more complicated. The two highest digits of A are

Algorithm 1 *Look Ahead* application algorithm

```

1:  $A = 0$ 
2: for  $i = n - 1$  to  $i = 0$  do
3:   Look Ahead: estimate  $z_i$ 
4:   if  $z_i$  exceeds one digit then
5:      $A = A \pm P \cdot 2^{(i+1)b}$ 
6:   end if
7:    $r = p_i \cdot z_i - x_i \cdot y_i$ 
8:    $A = A + \sum_{j=0}^{i-1} ((x_i + x_j) \cdot (y_i + y_j) + r) \cdot 2^{(i+j)b}$ 
9:    $A = A - r \cdot 2^{(2i)b}$ 
10:   $A = A + \sum_{j=i+1}^{n-1} (r - (p_i + p_j) \cdot (z_i + z_j)) \cdot 2^{(i+j)b}$ 
11: end for
12: if  $A > P$  or  $A < 0$  then
13:    $A = A \pm P$ 
14: end if

```

calculated from

$$\begin{aligned}
& A + (r_i - (p_i + p_{n-1}) \cdot (z_i + z_{n-1})) \cdot 2^{(i+n-1)b} \\
& + (r_i - (p_i + p_{n-2}) \cdot (z_i + z_{n-2})) \cdot 2^{(i+n-2)b} \\
& + (r_{i-1} - (p_{i-1} + p_{n-1}) \cdot (z_{i-1} + z_{n-1})) \cdot 2^{(i+n-2)b} \\
= & A - (x_i \cdot y_i + p_{n-1} \cdot z_{n-1} + p_i \cdot z_{n-1} + p_{n-1} \cdot z_i) \cdot 2^{(i+n-1)b} \\
& - (x_i \cdot y_i + p_{n-2} \cdot z_{n-2} + p_i \cdot z_{n-2} + p_{n-2} \cdot z_i) \cdot 2^{(i+n-2)b} \\
& - (x_{i-1} \cdot y_{i-1} + p_{n-1} \cdot z_{n-1} + p_{i-1} \cdot z_{n-1} + p_{n-1} \cdot z_{i-1}) \cdot 2^{(i+n-2)b}.
\end{aligned}$$

By subtracting all terms except the ones containing z_i and z_{i-1} we get

$$\begin{aligned}
\tilde{A} = & A - (x_i \cdot y_i + p_{n-1} \cdot z_{n-1} + p_i \cdot z_{n-1}) \cdot 2^{(i+n-1)b} \\
& - (x_i \cdot y_i + p_{n-2} \cdot z_{n-2} + p_i \cdot z_{n-2}) \cdot 2^{(i+n-2)b} \\
& - (x_{i-1} \cdot y_{i-1} + p_{n-1} \cdot z_{n-1} + p_{i-1} \cdot z_{n-1}) \cdot 2^{(i+n-2)b}.
\end{aligned}$$

Note that this equation is not valid for $i = n - 2$ and $i = 0$, because at these steps of the main loop the terms added to the two highest digits have a different form. But by executing the LA with these terms instead we also get \tilde{A} , the calculations are even simplified:

- For $i = n - 2$ the term

$$- (x_i \cdot y_i + p_{n-2} \cdot z_{n-2} + p_i \cdot z_{n-2}) \cdot 2^{(i+n-2)b}$$

has to be substituted with

$$+ (x_i \cdot y_i) \cdot 2^{(i+n-2)b}.$$

- For $i = 0$ there is no need for the term

$$- (x_{i-1} \cdot y_{i-1} + p_{n-1} \cdot z_{n-1} + p_{i-1} \cdot z_{n-1}) \cdot 2^{(i+n-2)b}.$$

The second step of the LA is the calculation of the current z_i -value. This is done by the multiplication of the highest significant digits of \tilde{A} with the reciprocal, for which we chose $c = 2b$, which leads to

$$q_{rec} = \frac{2^{4b}}{p_{n-1} \cdot 2^b + p_{n-2}}.$$

The equation for calculating z_i is given by

$$z_i = q_{rec} \cdot (\tilde{a}_{i+n+1} \cdot 2^{2b} + \tilde{a}_{i+n} \cdot 2^b + \tilde{a}_{i+n-1}) \div 2^{3b},$$

where \div is implemented as bit shift operation.

3.4 Complexity estimation

Although the most important metric is the number of necessary digit multiplications, we also consider the amount of digit additions, memory accesses and memory usage (summarized in Table 2).

Multiplications: In each step of the main loop $n + 1$ digit multiplications are needed:

- 2 multiplications for the computation of r
- $n - 1$ multiplications for the computation of both sums (step 8 and 10 of Algorithm 1)

In this implementation we also need 3 multiplications for the Two-level-LA and further 3 ones for the multiplication with q_{rec} – one multiplication of two 2-digit-numbers executed with the Karatsuba algorithm. This results in a total of $n^2 + 7n$ digit multiplications. Accepting a lower accuracy by using One-level-LA and smaller c resulting in a shorter q_{rec} – e.g. in the case of over-long z_i – the complexity can be lowered to $n^2 + 3n$.

Note that calculations for the LA contain digit multiplications, whose results can be reused in subsequent steps of the main loop if stored in a local register. We took advantage of this while counting the number of multiplications, as we did in our implementation.

Additions: In each step of the main loop a total of $4n$ digit additions is needed:

- 4 additions for the computation of r and its subtraction in the i th step of the inner loop
- $4(n - 1)$ additions for the $n - 1$ remaining steps of the inner loop

In this implementation we need 20 additional digit additions in each general step of the main loop. Ignoring the special cases this results in a whole of $4n^2 + 20n$ additions. Generally, the amount of additions in the LA depends on the LA-level as well as implementation constraints and can be as small as $6n$.

Memory accesses: In every main step each of the n valid digits of A is read and written once. Assuming that p_i, z_i and all other values needed for the LA are stored in local registers, there is an additional amount of $2n$ memory read accesses for x_i, y_i, x_j, y_j, p_j , and z_j in each step of the main loop. This results in a total of $4n^2$ memory accesses. Our implementation also stored q_{rec} in memory, thus leading to $2n$ additional memory accesses.

Memory usage: In addition to the operands the numbers Z (with n digits) and A (with $2n + 1$ digits) have to be stored. Because the already calculated digits of Z increase in the same degree as the valid digits of A decrease, it is possible to store Z inside the memory of A . All other intermediate results, e.g. r, q_{rec} or the part of \tilde{A} computed in the LA, exhibit a constant digit count and can therefore be stored in local registers. Assuming this is done, $2n + 1$ digits of memory are required. Our implementation stored q_{rec} in memory and didn't re-use the storage of A for Z , thus needing $n + 2$ additional digits of memory.

	Theoretical value	Achieved value
Digit multiplications	$n^2 + 3n$	$n^2 + 7n$
Digit additions	$4n^2 + 6n$	$4n^2 + 20n$
Memory accesses	$4n^2$	$4n^2 + 2n$
Memory usage	$2n + 1$	$3n + 3$

Table 2: Complexity estimation

4 Implementation

The described algorithm does not specify the exact execution order of the LA thus allowing a wide range of optimizations depending on the underlying hardware. Goal of our implementation was to test the basic properties of the new algorithm focusing on the achievable performance.

For test purposes we integrated our multiplication algorithm into a point multiplication for elliptic curve cryptography[‡] in $\mathbb{GF}(P)$. As platform we used a Xilinx Virtex II Pro FPGA[12]. For portability reasons we considered only the logic resources (CLBs), e.g., ignoring dedicated multipliers. The algorithms for the elliptic curve operations *ECAdd* and *ECDouble* were taken from the IEEE 1363-specifications[3].

The EC multiplication was implemented by a variation of the Lim/Lee exponentiation algorithm[6], with $h = 3$, which resulted in 8 pre-calculated points (in turn, this results in $a = \lceil \frac{\text{bit length}}{3} \rceil$). We denote the pre-computation phase *ECInit* and the actual multiplication phase *ECMult*.

- *ECInit* requires $2a$ *ECDouble*- and a *ECAdd*-operations, which corresponds to $36a$ finite field modular multiplications.
- *ECMult* results in a *ECDouble*- and $a + 1$ *ECAdd*-operations as well as one finite field inversion – implemented using *Fermat’s Little Theorem*[7] – which equals a total of $26a + 16 + (2 \cdot \text{bit length})$ modular multiplications.

For the detailed implementation the alternative without over-long z_i was realized. Thus, we do not need to account for overflow bits and reach a higher utilization of the digit operations. To optimize the algorithm, the parameters must minimize the number of operations needed for LA and corrections fulfilling the following equation:

$$OP_{LA+corr} = OP_{LA} + OP_{corr} \cdot p_{corr}.$$

OP_{LA} and OP_{corr} denote the number of operations needed for one execution of the LA and of one correction, respectively. p_{corr} is the probability that one correction is necessary per step of the main loop (note that $p_{corr} > 1$ may occur). There is an optimum, where further improving p_{corr} increases OP_{LA} faster than it decreases the total of used operations for corrections.

- The digit length b was chosen to be 16 bits. We set c as a multiple of b thus utilizing all bits of any digits which were part of the calculation. As can be seen in Table 3, the optimum is $c = 2b = 32$ bit. $c = b$ results in too many corrections and the improved p_{corr} of $c = 3b$ does not offset the increased OP_{LA} . In all following tables the values were generated by the execution of multiplications of random numbers exploiting the test implementation.
- The optimum LA-level is a Two-level-LA because the smaller p_{corr} of a Three-level-LA does not offset the increase of OP_{LA} as can be seen from Table 4. Values for the One-level-LA are not presented here, because they resulted in $p_{corr} > 1$, which does not allow to ignore the OP_{corr} for complexity considerations.

[‡]for more details on elliptic curve cryptography see [1]

c	corrections	p_{corr}
16	230979	0.230979
32	34	$3.4 \cdot 10^{-5}$
48	23	$2.3 \cdot 10^{-5}$

Table 3: p_{corr} for different c (Two-level-LA, 160 bit and 100000 multiplications each)

bit length	Two-level-LA		Three-level-LA	
	corrections	p_{corr}	corrections	p_{corr}
112	1523	$2.176 \cdot 10^{-5}$	615	$8.786 \cdot 10^{-6}$
128	1887	$2.359 \cdot 10^{-5}$	601	$7.513 \cdot 10^{-6}$
160	2557	$2.557 \cdot 10^{-5}$	757	$7.57 \cdot 10^{-6}$
192	3310	$2.758 \cdot 10^{-5}$	925	$7.708 \cdot 10^{-6}$
224	4003	$2.859 \cdot 10^{-5}$	1166	$8.329 \cdot 10^{-6}$
256	4865	$3.040 \cdot 10^{-5}$	1368	$8.55 \cdot 10^{-6}$

Table 4: p_{corr} for different bit length (10 million multiplications each)

The execution order of the test implementation is the same as depicted in Algorithm 1. Thus, the correction is done directly after the LA, if z_i does not fit into one digit, i.e., an overflow occurred. An addition is executed, if the overflow is negative. Otherwise, the correction is done by a subtraction.

5 Results

For testing the outlined implementation the curve parameter specifications from [11] were considered. Table 5 shows the mean execution times over 20 point multiplications for each parameter set. For comparison purposes we substituted the proposed algorithm with the Montgomery multiplication leaving everything else unchanged. The results of this implementation are also depicted in Table 5. Both implementations execute the finite field multiplication at a rate of 100 MHz while the rest runs at only 33.3 MHz. It can be seen that the theoretical advantage of the new multiplication algorithm translates well into the practical implementation.

Note that not using the dedicated multipliers led to a costly digit multiplication implementation. This complicates comparisons with other FPGA implementations, which use the dedicated multipliers and thus exhibit in general superior performance figures - at the expense of portability to different FPGA architectures.

6 Conclusion

We presented a novel multiplication algorithm for $\mathbb{GF}(P)$ which unveils a complexity of approximately $n^2 + 7n$ (the exact value depends on the implementation details). To our knowledge this is superior to the complexity values of any other modular multiplication algorithm for $\mathbb{GF}(P)$. Important for the algorithm is the *Look Ahead*-mechanism, which can be implemented in different ways. The selection of appropriate parameters depends on the constraints of the application case. For comparison purposes we also implemented the Montgomery multiplication and demonstrated that the theoretical improvement of the complexity is well visible in a real implementation. Note

Parameter set	Proposed algorithm		Montgomery	
	<i>ECInit</i>	<i>ECMult</i>	<i>ECInit</i>	<i>ECMult</i>
secp112r1	15.911ms	14.182ms	17.376ms	15.755ms
secp128r1	21.702ms	21.151ms	24.777ms	24.474ms
secp160k1	37.829ms	37.134ms	46.263ms	45.931ms
secp192k1	59.392ms	59.456ms	76.498ms	77.063ms
secp224k1	88.920ms	89.330ms	119.211ms	120.792ms
secp256k1	126.696ms	126.834ms	175.476ms	178.522ms

Table 5: Execution times of the test implementation

that the algorithm was designed focussing on hardware implementations. Whether or with which parameters it will have advantages in software is still an open question.

References

- [1] BLAKE, I. F., SEROUSSI, G., AND SMART, N. P. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.
- [2] ERNST, M., JUNG, M., MADLENER, F., HUSS, S. A., AND BLÜMEL, R. A reconfigurable system on chip implementation for elliptic curve cryptography over $\mathbb{GF}(2^n)$. In *Cryptographic Hardware and Embedded Systems - CHES 2002* (2002), B. S. Kaliski, Çetin K. Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer-Verlag London, 2003, pp. 381–399.
- [3] IEEE 1363. *Standard Specifications for Public-Key Cryptography – Annex A*, 2000. <http://grouper.ieee.org/groups/1363/>.
- [4] KARATSUBA, A., AND OFMAN, Y. Multiplication of multidigit numbers on automata. *Sov. Phys.-Dok (Engl. transl.)* vol. 7, no. 7 (1963), pp. 595–596.
- [5] KOCHER, P., JAFFE, J., AND JUN, B. Differential power analysis. In *Advances in cryptology — CRYPTO '99* (1999), M. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 388–397.
- [6] LIM, C. H., AND LEE, P. J. More flexible exponentiation with precomputation. In *Advances in Cryptography — CRYPTO'94* (August 1994), Y. G. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, pp. 95–107.
- [7] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of Applied Cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press, 1997.
- [8] MILLER, V. S. Use of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO '85* (18–22 Aug. 1985), H. C. Williams, Ed., vol. 218 of *Lecture Notes in Computer Science*, Springer-Verlag, 1986, pp. 417–426.
- [9] MONTGOMERY, P. L. Modular multiplication without trial division. *Mathematics of Computation* 44, 170 (Apr. 1985), 519–521.

- [10] RIVEST, R., SHAMIR, A., AND ADLEMAN, L. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM* 21, 2 (Feb. 1978), 120–126.
- [11] STANDARDS FOR EFFICIENT CRYPTOGRAPHY GROUP (SECG). *Specification of Standards for Efficient Cryptography — SEC 2: Recommended Elliptic Curve Domain Parameters*, September 2000.
- [12] XILINX. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheets*, June 2005. <http://www.xilinx.com/products/>.