A new algorithm for residue multiplication modulo $2^{521} - 1$

Shoukat Ali and Murat Cenk

Institute of Applied Mathematics
Middle East Technical University, Ankara, Turkey
shoukat.1983@gmail.com
mcenk@metu.edu.tr

Abstract. We present a new algorithm for residue multiplication modulo the Mersenne prime $2^{521} - 1$ based on the Toeplitz matrix-vector product. For this modulo, our algorithm yields better result in terms of the total number of operations than the previously known best algorithm of R. Granger and M. Scott presented in Public Key Cryptography - PKC 2015. Although our algorithm has nine more multiplications than Granger-Scott multiplication algorithm, the total number of additions is forty-two less than their algorithm. Even if one takes a ratio of $1 : 4$ between multiplication and addition our algorithm still has less total number of operations. We also present the test results of both the multiplication algorithms on an Intel Sandy Bridge Corei5-2410M machine, with and without optimization option in GCC.

Keywords: residue multiplication, Toeplitz matrix-vector product, Mersenne prime, elliptic curve cryptography

1 Introduction

In elliptic curve crytography (ECC) point multiplication is a vital operation and used for key generation, key exchange using Diffie-Hellman and digital signature. For cryptographic sizes, a scalar multiplication requires several hundreds of modular multiplications and the cost of other primitive operations is negligible with respect to this operation. Therefore, a good amount of research has focused on improving the efficiency of the modular multiplication. In modular multiplication the bottleneck is reduction, therefore, Solinas [6] constructed a group of modulus, Generalized Mersenne Numbers (GMN), to speedup the modular reduction. That is why his four recommended moduli are fully part of the standards, NIST [10] and SECG [9]. On the other hand, reduction modulo the Mersenne prime $2^{521}-1$ is optimal because the cost of reduction is equivalent to modular addition due to the constant term 1.

Based on most of the past research, it is recommended the use of schoolbook multiplication for ECC sizes because the cost of overheads in other techniques outweighs the saving of the multiplication operation. Karatsuba technique is good for binary fields but for prime fields the bitlength was perceived not to be

sufficient enough to make the cost of saved multiplication worth more than the addition overhead. However, Bernstein et al. [3] have used two levels of refined Karatsuba followed by schoolbook multiplication in an ingenious way on the modulus of size 414-bit. They achieved a good efficiency by splitting the operands into limbs of size less than the word-size - 32-bit - of the machine in order to postpone the carry and avoiding the overflow in double-word. On the other hand, Granger-Scott [1] proposed an efficient algorithm for multiplication modulo the Mersenne prime $2^{521} - 1$. They achieved the efficiency because of the modulus form and finding out that residue multiplication can take as many word-by-word multiplication as squaring with very little extra addition as overhead.

We applied the technique proposed by Bernstein et al. [3] to modulus $2^{521} - 1$ and found that it is more costly in terms of the number of operations than the technique proposed by Granger-Scott [1]. Bernstein technique uses many optimization factors but we have considered the case where the operands, both the scalar and the point on the curve, comprise of 9-limb and applied both the techniques straight away.

The work of R. Granger and M. Scott [1] made us interested to explore and exploit the structure of modular multiplication. So we started to look at the problem from top-to-bottom by considering the residue structure once the reductions have been performed and to reckon the remainder from the given inputs. This shift of paradigm has already led to many good results.

In this paper we propose a new algorithm for residue multiplication modulo the Mersenne prime $2^{521} - 1$ that is cheaper in terms of the total number of operations – even if we take the ratio of multiplication to addition $1 : 4$ – than the recently proposed algorithm of Granger and Scott [1]. We have achieved this efficiency based on the representation and structure of multiplication modulo the Mersenne prime $2^{521} - 1$ as matrix-vector multiplication in general and Toeplitz matrix in particular. The Toeplitz matrix has the great properties of (1) partitioning of a Toeplitz matrix results into Toeplitz matrices (2) addition and subtraction of Toeplitz matrices is also a Toeplitz matrix (3) addition and subtraction require only computation of first row and first column and (4) Topelitz matrix-vector product can be performed efficiently. Using these four properties we have achieved a better efficiency than Granger-Scott algorithm [1]. We also tested two versions of our algorithm along with Granger-Scott multiplication algorithm [1] on different set of (random) values.

The rest of the paper is organized as follows. In Section 2, we explain the previous work on Toeplitz matrix-vector product (TMVP) with special focus on matrix and vector of size 3. Then, we investigate the variants of Topletiz matrix-vector multiplication in terms of the number of operations for our problem case in Section 3. Next, we apply and explain our proposed method in detail in Section 4. We provide our multiplication and squaring algorithms and compare the multiplication algorithm in [1] to our's in Section 5. Finally, we conclude our paper in Section 6.

2 Algorithms for Toeplitz matrix-vector product

We observed that residue multiplication modulo $2^{521}-1$ can be presented by Toeplitz matrix-vector multiplication (TMVP). A Toeplitz matrix or diagonal-constant matrix is a square matrix in which each descending diagonal from left to right is constant i.e. a Toeplitz matrix is of the following form:

$$\begin{bmatrix} a_0 & a_1 & a_2 & \dots & \dots & \dots & a_{n-1} \\ a_n & a_0 & a_1 & \ddots & \ddots & \ddots & a_{n-2} \\ a_{n+1} & a_n & a_0 & \ddots & \ddots & \ddots & a_{n-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & a_n & a_0 & a_1 \\ a_{2(n-1)} & \dots & \dots & \dots & a_{n+1} & a_n & a_0 \end{bmatrix}$$

One of the techniques for TMVP is to use the schoolbook method and for size n the time complexity is $\mathcal{O}(n^2)$. In the literature, there are algorithms better than the schoolbook. For example a study on this subject for multiplication over $\mathbb{F}_2$ can be found in [2]. For a TMVP of size 3 we have

$$\begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_0 & a_1 \\ a_4 & a_3 & a_0 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} m_3 + m_4 + m_6 \\ m_2 - m_4 + m_5 \\ m_1 - m_2 - m_3 \end{bmatrix} \tag{1}$$

where

$$\begin{aligned} &m_1 = (a_4 + a_3 + a_0)b_0, && m_2 = a_3(b_0 - b_1), && m_3 = a_0(b_0 - b_2), \\ &m_4 = a_1(b_1 - b_2), && m_5 = (a_0 + a_3 + a_1)b_1, && m_6 = (a_2 + a_0 + a_1)b_2 \end{aligned}$$

The total cost of m_i for $i = 1, \dots, 6$ is $6\mathbf{M}+8\mathbf{A}+6\mathbf{A}_d$ where $\mathbf{M}$ is the cost of a single precision/word multiplication, $\mathbf{A}$ is the cost of a single precision/word addition and $\mathbf{A}_d$ is the cost of a double precision/word addition. The cost of single precision addition is 8 because one can take common either $(a_3 + a_0)$ between m_1 and m_5 or $(a_0 + a_1)$ between m_5 and m_6. For all those machines where the ratio of multiplication to addition is greater than or equal to $1:3$ this observation is worth to apply. For larger sizes, we can use this technique recursively and for size n it results in a time complexity of $\mathcal{O}(n^{1.63})$ which is better than schoolbook.

3 Multiplication modulo $2^{521}-1$ using TMVP

Suppose F and G are two large integers of 521-bit to be multiplied and we are working on a 64-bit architecture. We partition each into nine limbs. Since $521/9 = 57.88$, so each limb comprises of at most 58-bit stored in a 64-bit word as shown below.

	58	116	174	232	290	348	406	464
f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
64-bit	64-bit	64-bit	64-bit	64-bit	64-bit	64-bit	64-bit	64-bit

	58	116	174	232	290	348	406	464
g_0	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
64-bit	64-bit	64-bit	64-bit	64-bit	64-bit	64-bit	64-bit	64-bit

The limbs f_8 and g_8 are 57-bit and we would like to work in modulo $p = 2^{521} - 1$. Let $Z = FG \bmod (2^{521}-1)$. Then, the limbs will be $Z = [Z_0, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8]$ where

$$
\begin{aligned}
Z_0 &= f_0g_0 + 2f_8g_1 + 2f_7g_2 + 2f_6g_3 + 2f_5g_4 + 2f_4g_5 + 2f_3g_6 + 2f_2g_7 + 2f_1g_8,\\
Z_1 &= f_1g_0 + f_0g_1 + 2f_8g_2 + 2f_7g_3 + 2f_6g_4 + 2f_5g_5 + 2f_4g_6 + 2f_3g_7 + 2f_2g_8,\\
Z_2 &= f_2g_0 + f_1g_1 + f_0g_2 + 2f_8g_3 + 2f_7g_4 + 2f_6g_5 + 2f_5g_6 + 2f_4g_7 + 2f_3g_8,\\
Z_3 &= f_3g_0 + f_2g_1 + f_1g_2 + f_0g_3 + 2f_8g_4 + 2f_7g_5 + 2f_6g_6 + 2f_5g_7 + 2f_4g_8,\\
Z_4 &= f_4g_0 + f_3g_1 + f_2g_2 + f_1g_3 + f_0g_4 + 2f_8g_5 + 2f_7g_6 + 2f_6g_7 + 2f_5g_8,\\
Z_5 &= f_5g_0 + f_4g_1 + f_3g_2 + f_2g_3 + f_1g_4 + f_0g_5 + 2f_8g_6 + 2f_7g_7 + 2f_6g_8,\\
Z_6 &= f_6g_0 + f_5g_1 + f_4g_2 + f_3g_3 + f_2g_4 + f_1g_5 + f_0g_6 + 2f_8g_7 + 2f_7g_8,\\
Z_7 &= f_7g_0 + f_6g_1 + f_5g_2 + f_4g_3 + f_3g_4 + f_2g_5 + f_1g_6 + f_0g_7 + 2f_8g_8,\\
Z_8 &= f_8g_0 + f_7g_1 + f_6g_2 + f_5g_3 + f_4g_4 + f_3g_5 + f_2g_6 + f_1g_7 + f_0g_8
\end{aligned}
$$

Since the modulus is a Mersenne prime, the reduction is just addition of two times of higher degree components – those whose exponent is greater than 521 and on reduction one more than the residue exponent – to respective lower degree component. There are two important things (i) the summation of f_ig_j don't cause overflow in double-word and (ii) each Z_i will be reduced to the size as those of the input limbs. So the above expression in matrix-vector form will be

$$
\begin{bmatrix}
f_0 & 2f_8 & 2f_7 & 2f_6 & 2f_5 & 2f_4 & 2f_3 & 2f_2 & 2f_1\\
f_1 & f_0 & 2f_8 & 2f_7 & 2f_6 & 2f_5 & 2f_4 & 2f_3 & 2f_2\\
f_2 & f_1 & f_0 & 2f_8 & 2f_7 & 2f_6 & 2f_5 & 2f_4 & 2f_3\\
f_3 & f_2 & f_1 & f_0 & 2f_8 & 2f_7 & 2f_6 & 2f_5 & 2f_4\\
f_4 & f_3 & f_2 & f_1 & f_0 & 2f_8 & 2f_7 & 2f_6 & 2f_5\\
f_5 & f_4 & f_3 & f_2 & f_1 & f_0 & 2f_8 & 2f_7 & 2f_6\\
f_6 & f_5 & f_4 & f_3 & f_2 & f_1 & f_0 & 2f_8 & 2f_7\\
f_7 & f_6 & f_5 & f_4 & f_3 & f_2 & f_1 & f_0 & 2f_8\\
f_8 & f_7 & f_6 & f_5 & f_4 & f_3 & f_2 & f_1 & f_0
\end{bmatrix}
\times
\begin{bmatrix}
g_0\\ g_1\\ g_2\\ g_3\\ g_4\\ g_5\\ g_6\\ g_7\\ g_8
\end{bmatrix}
\tag{2}
$$

If we use the schoolbook algorithm for computing (2), then for 521-bit Mersenne prime and 64-bit word, we have $n = 9$ and the cost will be

$$\textbf{Cost} = 81\mathbf{M} + 72\mathbf{A}_d = 81\mathbf{M} + 144\mathbf{A} = 225 \text{ operations}$$

The recursive use of (1) for $n = 9$ will result into

$$\textbf{Cost} = 6\big(6\mathbf{M} + 8\mathbf{A} + 6\mathbf{A}_d\big) + 34\mathbf{A} + 18\mathbf{A}_d = 36\mathbf{M} + 190\mathbf{A} = 226 \text{ operations}$$

Now, we first use TMVP for partitioning the matrix into sub-matrices of size $n/3$ and then use the schoolbook. We call it the mixed version and for our case

of $n = 9$ the cost will be

$$\textbf{Cost} = 6\big(3^2\mathbf{M} + 3(3-1)\mathbf{A}_d\big) + 34\mathbf{A} + 18\mathbf{A}_d = 54\mathbf{M} + 142\mathbf{A} = 196 \text{ operations}$$

Clearly this technique seems to be a good alternative to Granger-Scott algorithm [1] where the total cost of their observation is 205 operations. The cost of our observation can be further reduced by exploiting the structure of Toeplitz matrix as explained in detail in the next section.

4 Proposed Method: Multiplication modulo $2^{521} - 1$

Now, we apply the mixed version for TMVP. First, we establish sub-matrices and using (1) we obtain

$$\begin{bmatrix} A_{11} & 2A_{31} & 2A_{21} \\ A_{21} & A_{11} & 2A_{31} \\ A_{31} & A_{21} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{31} \end{bmatrix} = \begin{bmatrix} M_3 + M_4 + M_6 \\ M_2 - M_4 + M_5 \\ M_1 - M_2 - M_3 \end{bmatrix}$$

where the sub-matrices A_{ij} are of size 3×3 and not independent whereas the vectors B_{kl} are of size 3×1. From here onwards by a word we mean 64-bit and double-word 128-bit. From (1) we have

$$\begin{aligned} M_1 &= (A_{31} + A_{21} + A_{11})B_{11}, & M_2 &= A_{21}(B_{11} - B_{21}) \\ M_3 &= A_{11}(B_{11} - B_{31}), & M_4 &= 2A_{31}(B_{21} - B_{31}) \\ M_5 &= (A_{11} + A_{21} + 2A_{31})B_{21}, & M_6 &= (2(A_{31} + A_{21}) + A_{11})B_{31} \end{aligned}$$

Computing M_1:

$$A_{31} + A_{21} = \begin{bmatrix} f_6 & f_5 & f_4 \\ f_7 & & \\ f_8 & & \end{bmatrix} + \begin{bmatrix} f_3 & f_2 & f_1 \\ f_4 & & \\ f_5 & & \end{bmatrix} = \begin{bmatrix} f_6 + f_3 & f_5 + f_2 & f_4 + f_1 \\ f_7 + f_4 & & \\ f_8 + f_5 & & \end{bmatrix} = \begin{bmatrix} S_1 & S_2 & S_3 \\ S_4 & & \\ S_5 & & \end{bmatrix}$$

$$(A_{31} + A_{21}) + A_{11} = \begin{bmatrix} S_1 & S_2 & S_3 \\ S_4 & & \\ S_5 & & \end{bmatrix} + \begin{bmatrix} f_0 & 2f_8 & 2f_7 \\ f_1 & & \\ f_2 & & \end{bmatrix} = \begin{bmatrix} S_6 & S_7 & S_8 \\ S_9 & & \\ S_{10} & & \end{bmatrix}$$

$$(A_{31} + A_{21} + A_{11})B_{11} = \begin{bmatrix} S_6 & S_7 & S_8 \\ S_9 & & \\ S_{10} & & \end{bmatrix} \times \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix}$$

Hence, the total cost of M_1 is $9\mathbf{M}+10\mathbf{A}+6\mathbf{A}_d$

Computing M_6: From the above equation it is evident that M_6 has common operations with M_1 on computing $(A_{31} + A_{21})$ so we have

$$2(A_{31} + A_{21}) = \begin{bmatrix} 2S_1 & 2S_2 & 2S_3 \\ 2S_4 & & \\ 2S_5 & & \end{bmatrix}$$

$$(2(A_{31} + A_{21}) + A_{11} = \begin{bmatrix} 2S_1 & 2S_2 & 2S_3 \\ 2S_4 & & \\ 2S_5 & & \end{bmatrix} + \begin{bmatrix} f_0 & 2f_8 & 2f_7 \\ f_1 & & \\ f_2 & & \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{14} & & \\ S_{15} & & \end{bmatrix}$$

$$((2(A_{31}+A_{21})+A_{11})B_{31} = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{14} & & \\ S_{15} & & \end{bmatrix} \times \begin{bmatrix} g_6 \\ g_7 \\ g_8 \end{bmatrix}$$

Hence, the total cost of M_6 is $9\mathbf{M}+5\mathbf{A}+6\mathbf{A}_d+5(\mathbf{M}^*2)$, where the expression $5(\mathbf{M}^*2)$ means that there are 5 multiplications by 2.

Computing M_5: We observe some common operations between M_5 and M_1 as explained below

$$A_{11}+A_{21} = \begin{bmatrix} f_0 & 2f_8 & 2f_7 \\ f_1 & & \\ f_2 & & \end{bmatrix} + \begin{bmatrix} f_3 & f_2 & f_1 \\ f_4 & & \\ f_5 & & \end{bmatrix} = \begin{bmatrix} S_{16} \\ S_3 \\ S_2 \end{bmatrix}$$

$$(A_{11}+A_{21})+2A_{31} = \begin{bmatrix} S_{16} & 2f_8+f_2 & 2f_7+f_1 \\ S_3 & & \\ S_2 & & \end{bmatrix} + \begin{bmatrix} 2f_6 & 2f_5 & 2f_4 \\ 2f_7 & & \\ 2f_8 & & \end{bmatrix} = \begin{bmatrix} S_{17} & S_{15} & S_{14} \\ S_8 & & \\ S_7 & & \end{bmatrix}$$

$$(A_{11}+A_{21}+2A_{31})B_{21} = \begin{bmatrix} S_{17} & S_{15} & S_{14} \\ S_8 & & \\ S_7 & & \end{bmatrix} \times \begin{bmatrix} g_3 \\ g_4 \\ g_5 \end{bmatrix}$$

Hence, the total cost of M_5 is $9\mathbf{M}+2\mathbf{A}+6\mathbf{A}_d$

Computing M_2:

$$B_{11}-B_{21} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} - \begin{bmatrix} g_3 \\ g_4 \\ g_5 \end{bmatrix} = \begin{bmatrix} g_0-g_3 \\ g_1-g_4 \\ g_2-g_5 \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}$$

$$A_{21}(B_{11}-B_{21}) = \begin{bmatrix} f_3 & f_2 & f_1 \\ f_4 & & \\ f_5 & & \end{bmatrix} \times \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}$$

Hence, the total cost of M_2 is $9\mathbf{M}+3\mathbf{A}+6\mathbf{A}_d$

Computing M_3:

$$B_{11}-B_{31} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} - \begin{bmatrix} g_6 \\ g_7 \\ g_8 \end{bmatrix} = \begin{bmatrix} g_0-g_6 \\ g_1-g_7 \\ g_2-g_8 \end{bmatrix} = \begin{bmatrix} U_4 \\ U_5 \\ U_6 \end{bmatrix}$$

$$A_{11}(B_{11}-B_{31}) = \begin{bmatrix} f_0 & 2f_8 & 2f_7 \\ f_1 & & \\ f_2 & & \end{bmatrix} \times \begin{bmatrix} U_4 \\ U_5 \\ U_6 \end{bmatrix}$$

Hence, the total cost of M_3 is $9\mathbf{M}+3\mathbf{A}+6\mathbf{A}_d$

Computing M_4:

$$B_{21}-B_{31} = \begin{bmatrix} g_3 \\ g_4 \\ g_5 \end{bmatrix} - \begin{bmatrix} g_6 \\ g_7 \\ g_8 \end{bmatrix} = \begin{bmatrix} g_3-g_6 \\ g_4-g_7 \\ g_5-g_8 \end{bmatrix} = \begin{bmatrix} U_7 \\ U_8 \\ U_9 \end{bmatrix}$$

$$2A_{31}(B_{21} - B_{31}) = \begin{bmatrix} 2f_6 & 2f_5 & 2f_4 \\ 2f_7 & & \\ 2f_8 & & \end{bmatrix} \times \begin{bmatrix} U_7 \\ U_8 \\ U_9 \end{bmatrix}$$

Hence, the total cost of M_1 is $9\mathbf{M}+3\mathbf{A}+6\mathbf{A}_d+5(\mathbf{M}*2)$, again the expression $5(\mathbf{M}*2)$ means that there are 5 multiplication by 2.

Final Computation: At last, we have to compute

$$\begin{bmatrix} M_3 + M_4 + M_6 \\ M_2 - M_4 + M_5 \\ M_1 - M_2 - M_3 \end{bmatrix}$$

where each M_i is a 3×1 vector and the elements are of double-word size so the total cost is $18\mathbf{A}_d$. Finally, the overall cost of the whole method for the mixed version is $54\mathbf{M}(64) + 26\mathbf{A}(64) + 54\mathbf{A}(128) + 10(\mathbf{M}(64)*2)$.

5 Algorithms and comparison

The Algorithm 1 presents the residue multiplication modulo $2^{521}-1$ as discussed in detail in Section 4. The output of algorithm is in reduced form rather than the unique residue for two reasons: the first reason is that for a large amount of inputs this reduced form is most probably the unique residue. The second reason is to save operations when the algorithm is used for the point multiplication as it has been done by others in the literature. The unique residue can be computed after the result is returned by the point multiplication procedure. So the intermediate results during the point multiplication are in reduced coefficient form.

The squaring algorithm, Algorithm 2, is same as the one in [1] except the range of limbs and modulus. We are not comparing the two squaring algorithms. We find that the cost of multiplication algorithm in [1] working modulo $2p = t^9 - 2$, where $t = 2^{58}$, is $45\mathbf{M}(64) + 74\mathbf{A}(64) + 60\mathbf{A}(128) + 9(\mathbf{M}(64)*2) + 5(\mathbf{M}(128)*2)) + 1(\mathbf{M}(128)*4))$. As shown in the last paragraph of the Section 4 clearly, our algorithm is better in terms of multiplication by small constants i.e. $2, 4$. But for the real comparison of the two multiplication algorithms we consider multiplication and addition/subtraction operations. So, the cost of multiplication algorithm in [1] can be written as $45\mathbf{M}(64) + 194\mathbf{A}(64)$. While on the other hand, the cost of our Algorithm 1 is $54\mathbf{M}(64) + 28\mathbf{A}(64) + 62\mathbf{A}(128)$ or $54\mathbf{M}(64) + 152\text{A}(64)$ working in modulo $p = 2^{521} - 1$. Suppose the ratio of multiplication to addition is $1 : 4$, then the total number of operations in [1] and our algorithm will be 374 and 368 respectively. Hence, in terms of the total number of operations our algorithm performs better and even more as the ratio gets smaller and smaller.

Algorithm 1 Multiplication

Input: $F = [f_0, \ldots\ldots, f_8], G = [g_0, \ldots\ldots, g_8] \in [0, 2^{58}-1]^8 \times [0, 2^{57}-1]$
Output: $Z = [z_0, \ldots\ldots, z_8][0, 2^{58}-1]^8 \times [0, 2^{57}-1]$ where $Z \equiv FG \pmod{2^{521}-1}$

$tmp \leftarrow [2f_8, 2f_7, 2f_6, 2f_5]$
$T_1[0] \leftarrow F[4] + F[1]$, $T_1[1] \leftarrow F[5] + F[2]$
$T_1[2] \leftarrow F[6] + F[3]$, $T_1[3] \leftarrow F[7] + F[4]$
$T_1[4] \leftarrow F[8] + F[5]$
$T_6[0] \leftarrow 2T_1[0] + tmp[1]$, $T_6[1] \leftarrow 2T_1[1] + tmp[0]$
$T_6[2] \leftarrow 2T_1[2] + F[0]$, $T_6[3] \leftarrow 2T_1[3] + F[1]$
$T_6[4] \leftarrow 2T_1[4] + F[2]$
$T_5 \leftarrow F[0] + F[3]$, $T_5 \leftarrow T_5 + tmp[2]$
$T_1[0] \leftarrow T_1[0] + tmp[1]$, $T_1[1] \leftarrow T_1[1] + tmp[0]$
$T_1[2] \leftarrow T_1[2] + F[0]$, $T_1[3] \leftarrow T_1[3] + F[1]$
$T_1[4] \leftarrow T_1[4] + F[2]$
$T_2[0] \leftarrow G[2] - G[5]$, $T_2[1] \leftarrow G[1] - G[4]$, $T_2[2] \leftarrow G[0] - G[3]$
$T_3[0] \leftarrow G[2] - G[8]$, $T_3[1] \leftarrow G[1] - G[7]$, $T_3[2] \leftarrow G[0] - G[6]$
$T_4[0] \leftarrow G[5] - G[8]$, $T_4[1] \leftarrow G[4] - G[7]$, $T_4[2] \leftarrow G[3] - G[6]$

$X_{10} \leftarrow (F[1] \cdot T_2[0]) + (F[2] \cdot T_2[1]) + (F[3] \cdot T_2[2])$
$X_{11} \leftarrow (F[2] \cdot T_2[0]) + (F[3] \cdot T_2[1]) + (F[4] \cdot T_2[2])$
$X_{12} \leftarrow (F[3] \cdot T_2[0]) + (F[4] \cdot T_2[1]) + (F[5] \cdot T_2[2])$
$X_{13} \leftarrow (tmp[1] \cdot T_3[0]) + (tmp[0] \cdot T_3[1]) + (F[0] \cdot T_3[2])$
$X_{14} \leftarrow (tmp[0] \cdot T_3[0]) + (F[0] \cdot T_3[1]) + (F[1] \cdot T_3[2])$
$X_{15} \leftarrow (F[0] \cdot T_3[0]) + (F[1] \cdot T_3[1]) + (F[2] \cdot T_3[2])$
$X_{16} \leftarrow (2F[4] \cdot T_4[0]) + (tmp[3] \cdot T_4[1]) + (tmp[2] \cdot T_4[2])$
$X_{17} \leftarrow (tmp[3] \cdot T_4[0]) + (tmp[2] \cdot T_4[1]) + (tmp[1] \cdot T_4[2])$
$X_{18} \leftarrow (tmp[2] \cdot T_4[0]) + (tmp[1] \cdot T_4[1]) + (tmp[0] \cdot T_4[2])$

$C \leftarrow (T_1[2] \cdot G[2]) + (T_1[3] \cdot G[1]) + (T_1[4] \cdot G[0]) - X_{12} - X_{15}$
$z_8 \leftarrow C \texttt{ mod } 2^{57}$
$C \leftarrow (T_6[0] \cdot G[8]) + (T_6[1] \cdot G[7]) + (T_6[2] \cdot G[6]) + X_{13} + X_{16} + (C \texttt{ >> } 57)$
$z_0 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_6[1] \cdot G[8]) + (T_6[2] \cdot G[7]) + (T_6[3] \cdot G[6]) + X_{14} + X_{17} + (C \texttt{ >> } 58)$
$z_1 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_6[2] \cdot G[8]) + (T_6[3] \cdot G[7]) + (T_6[4] \cdot G[6]) + X_{15} + X_{18} + (C \texttt{ >> } 58)$
$z_2 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_6[3] \cdot G[5]) + (T_6[4] \cdot G[4]) + (T_5 \cdot G[3]) + X_{10} - X_{16} + (C \texttt{ >> } 58)$
$z_3 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_6[4] \cdot G[5]) + (T_5 \cdot G[4]) + (T_1[0] \cdot G[3]) + X_{11} - X_{17} + (C \texttt{ >> } 58)$
$z_4 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_5 \cdot G[5]) + (T_1[0] \cdot G[4]) + (T_1[1] \cdot G[3]) + X_{12} - X_{18} + (C \texttt{ >> } 58)$
$z_5 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_1[0] \cdot G[2]) + (T_1[1] \cdot G[1]) + (T_1[2] \cdot G[0]) - X_{10} - X_{13} + (C \texttt{ >> } 58)$;
$z_6 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow (T_1[1] \cdot G[2]) + (T_1[2] \cdot G[1]) + (T_1[3] \cdot G[0]) - X_{11} - X_{14} + (C \texttt{ >> } 58)$
$z_7 \leftarrow C \texttt{ mod } 2^{58}$

$c \leftarrow C \texttt{ >> } 58$ where 'c' is a single-word variable
$c \leftarrow c + z_8$, $z_8 \leftarrow c \texttt{ mod } 2^{57}$, $z_0 \leftarrow z_0 + (c \texttt{ >> } 57)$
Return $[z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8]$

Algorithm 2 Squaring

Input: $F = [f_0, \ldots\ldots, f_8], \in [0, 2^{58} - 1]^8 \times [0, 2^{57} - 1]$
Output: $Z = [z_0, \ldots\ldots, z_8][0, 2^{58} - 1]^8 \times [0, 2^{57} - 1]$ where $Z \equiv F^2 \pmod{2^{521} - 1}$

$C \leftarrow 2(F[0] \cdot F[8] + F[1] \cdot F[7] + F[2] \cdot F[6] + F[3] \cdot F[5]) + F[4] \cdot F[4]$
$z_8 \leftarrow C \texttt{ mod } 2^{57}$
$C \leftarrow F[0] \cdot F[0] + 4(F[1] \cdot F[8] + F[2] \cdot F[7] + F[3] \cdot F[6] + F[4] \cdot F[5]) + (C \texttt{ >> } 57)$
$z_0 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[1] + F[5] \cdot F[5]) + 4(F[2] \cdot F[8] + F[3] \cdot F[7] + F[4] \cdot F[6]) + (C \texttt{ >> } 58)$
$z_1 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[2]) + F[1] \cdot F[1] + 4(F[3] \cdot F[8] + F[4] \cdot F[7] + F[5] \cdot F[6]) + (C \texttt{ >> } 58)$
$z_2 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[3] + F[1] \cdot F[2] + F[6] \cdot F[6]) + 4(F[4] \cdot F[8] + F[5] \cdot F[7]) + (C \texttt{ >> } 58)$
$z_3 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[4] + F[1] \cdot F[3]) + F[2] \cdot F[2] + 4(F[5] \cdot F[8] + F[6] \cdot F[7]) + (C \texttt{ >> } 58)$
$z_4 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[5] + F[1] \cdot F[4] + F[2] \cdot F[3] + F[7] \cdot F[7]) + 4(F[6] \cdot F[8]) + (C \texttt{ >> } 58)$
$z_5 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[6] + F[1] \cdot F[5] + F[2] \cdot F[4]) + F[3] \cdot F[3] + 4(F[7] \cdot F[8]) + (C \texttt{ >> } 58)$
$z_6 \leftarrow C \texttt{ mod } 2^{58}$
$C \leftarrow 2(F[0] \cdot F[7] + F[1] \cdot F[6] + F[2] \cdot F[5] + F[3] \cdot F[4] + F[8] \cdot F[8]) + (C \texttt{ >> } 58)$
$z_7 \leftarrow C \texttt{ mod } 2^{58}$

$c \leftarrow C \texttt{ >> } 58$ where 'c' is a single-word variable
$c \leftarrow c + z_8$, $z_8 \leftarrow c \texttt{ mod } 2^{57}$, $z_0 \leftarrow z_0 + (c \texttt{ >> } 57)$
Return $[z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8]$

5.1 Implementation Result

For implementation we use Ubuntu LTS 14.04 on an Intel Sandy Bridge Corei5 − 2410M CPU with 8GB RAM. We have implemented our algorithm in C language using GCC compiler. The program contains both the multiplication algorithms as two procedures. Both algorithms are tested on the same set of (random) values with and without optimization options in GCC. The tests include two versions of Algorithm 1 in the first version the arrays and variables were taken signed integers and in the second version the arrays T_1, T_6 and variables T_5, c were taken as unsigned integers while the rest of the data were signed. The result of our tests in terms of the number of clock cycles is shown in Table 1.

Unlike the arithmetic complexity – total number of operations – the practical result depicts a different story especially when the level of optimization is increased. Similarly, from the Table 1 it is clear that the behavior of **V-1** is some what according to expectation with respect to higher optimization level while **V-2** is showing a strange behavior. To find out these reasons we generated the assembly code of our program for each option as in Table 1.

Option	Granger-Scott	V-1	V-2
No Optimization	758	627	618
-O (level 1)	413	358	306
-O2 (level 2)	266	366	384
-O3 (level 3)	266	355	317

Table 1. Number of clock cycles with and without optimization option in GCC. **V-1**=Algorithm 1 (Signed) **V-2**=Algorithm 1 ((Un)signed)

First we investigated the difference of clock cycles between Granger-Scott multiplication algorithm [1] and our multiplication algorithm based on the number of operations. We find that in the sum of products expressions unlike the Granger-Scott algorithm where the compiler performs the signed integer multiplication with one IMUL instruction, in our case the same operation is performed as the follows:

- SAR by 0x3f to determine the sign of operand1
- IMUL to obtain $\{0, -\text{operand2}\}$
- MUL multiplication of operands
- ADD $\{0, -\text{operand2}\}$ to the higher part/digit of the mul result

Therefore, the total number of multiplications and additions/subtractions operation in the assembly code of both **V-1** and **V-2** were more than the expected result. Since our algorithm is based on the idea of storing the result of common operations the space required by our algorithm is more as compared to Granger-Scott algorithm [1] especially with respect to the number of General Purpose Registers (GPRs) on Intel processors. We find that on our machine the GCC compiler uses only the GPRs for optimization which in turn makes the available registers become more scarce and resulting into more and more unncessary operations. So, we believe that this strange behavior of compiler is all due to the small number GPRs with respect to our algorithm requirement. We think that an architecture with larger number of GPRs may give better result with our algorithm for compiler optimization just like the Granger-Scott algorithm [1].

Since **V-2** contains less number of signed operations than **V-1** therefore, for the second behavior, clock cycle difference between **V-1** and **V-2**, we found the total number of operations in **V-1** were more than **V-2**.

6 Conclusion

In this paper we have shown that TMVP approach for residue mutliplication modulo the Mersenne prime $2^{521} - 1$ takes less number of operations than Granger-Scott algorithm [1]. We have tested both multiplication algorithms in C language using GCC compiler with and without optimization option on different set of (random) values. In spite of the arithmetic complexity and taking

a ratio of 1 : 4 between multiplication and addition we have found implementation results, clock cycles, different. Upon investigation we have reached to the conclusion that our algorithm requires more memory as compared to Granger-Scott multiplication algorithm [1] especially with respect to the General Purpose Registers (GPRs) on Intel machine. Therefore, we are exploring three frontiers (1) assembly level optimizaiton on Intel machines (2) architectures with large number of registers in general and GPRs in particular and (3) comparison of the mixed version to recursive version on such machine(s).

Acknowledgment

The second author is supported in part by TÜBİTAK under Grant No. BIDEB-114C052.

References

1. Robert Granger, Michael Scott. Faster ECC over $\mathbb{F}_{2^{521}-1}$. *Public-Key Cryptography–PKC 2015*. Lecture Notes in Computer Science Volume 9020, pages 539-553, 2015.
2. Haining Fan, M. Anwar Hasan. A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields. *IEEE Trans. Computers 56(2)*, pages 224-233, 2007.
3. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange. Curve41417: Karatsuba revisited. In *Cryptographic hardware and embedded systemsCHES 201416th international workshop, Busan, South Korea, September 2326, 2014, proceedings*, edited by Lejla Batina, Matthew Robshaw. Lecture Notes in Computer Science 8731, Springer, pages 316334, 2014.
4. Daniel J. Bernstein, Tanja Lange. Faster addition and doubling on elliptic curves. In *Advances in cryptologyASIACRYPT 2007, 13th international conference on the theory and application of cryptology and information security, Kuching, Malaysia, December 26, 2007, proceedings*, edited by Kaoru Kurosawa. Lecture Notes in Computer Science 4833, Springer, pages 2950, 2007.
5. Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *Public key cryptography-PKC 2006, 9th international conference on theory and practice in public-key cryptography, New York, USA, April 2426, 2006, proceedings*, edited by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin. Lecture Notes in Computer Science 3958, Springer, Pages 207228, 2006.
6. Jerome A. Solinas, Generalized Mersenne Numbers (GMN), Technical Report, National Security Agency, Ft. Meade, MD, USA, 1999.
7. Darrel Hankerson, Alfred Menezes, Scott Vanstone. Guide to Elliptic Curve Cryptography. Springer, 2004.
8. Diego F. Aranha, Paulo S. L. M. Barreto, Geovandro C. C. F. Pereira, Jeferson Ricardini. A note on high-security general-purpose elliptic curves. 2013. http://eprint.iacr.org/2013/647
9. Standards for Efficient Cryptography Group. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Version 2.0, 27 January 2010, available on http://www.secg.org/sec2-v2.pdf.

10. US Department of Commerce, National Institute of Standards and Technology (NIST). *Federal Information Processing Standards Publication (FIPS) 186-4, Digital Signature Standard (DSS)*. FIPS PUB 186-4, July 2013, available on http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf