***I.J. Information Technology and Computer Science*, 2023, 4, 1-9**
Published Online on August 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2023.04.01

Software Implementation of Modular Reduction by Pseudo-mersenne Primes

Mariia Kovtun*
Department of Computer Engineering, National Aviation University, 1, Liubomyra Huzara ave., Kyiv, 03058, Ukraine
Cipher Company, 25/27, Nahirna Str., Kyiv, 04107, Ukraine
E-mail: mg.kovtun@gmail.com
ORCID iD: https://orcid.org/0000-0002-3021-2659
*Corresponding author

Vladyslav Kovtun
Cipher Company, 25/27, Nahirna Str., Kyiv, 04107, Ukraine
E-mail: vlad.kovtun@cipher.com.ua
ORCID iD: https://orcid.org/0000-0002-4303-3510

Oleksandr Stokipnyi
Cipher Company, 25/27, Nahirna Str., Kyiv, 04107, Ukraine
E-mail: alex.stokipny@cipher.com.ua
ORCID iD: https://orcid.org/0009-0007-4346-9684

Andrew Okhrimenko
Department of System Analysis and Information Technology, Mariupol State University, 6/4, Preobrazhenska Str., Kyiv, 03037, Ukraine
E-mail: andrew.okhrimenko@gmail.com
ORCID iD: https://orcid.org/0000-0001-8270-2863

Received: 20 March 2023; Revised: 03 May 2023; Accepted: 11 June 2023; Published: 08 August 2023

Abstract: Modern cryptosystems allow the use of operation in prime fields with special kind of modules that can speed up the prime field operation: multiplication, squaring, exponentiation. The authors took into account in the optimizations: the CPU architecture and the multiplicity of the degree of the modulus in relation to the machine word width. As example, shown adopted module reduction algorithms hard-coded for modern CPU in special form of pseudo-Mersenne prime $p = 2^{130} - 5$ used in MAC algorithm Poly1305, $p = 2^{255} - 19$ - in electronic signature algorithm EdDSA and $p = 2^{768} - 9659$ - in short message encryption algorithm DSTU 9041. These algorithms have been software implemented on both 32-bit and 64-bit platforms and compared with Barrett modular reduction algorithm for different pseudo-Mersenne and generalized-Mersenne modules. Timings for proposed and Barrett algorithms for different modules are presented and discussed.

Index Terms: Pseudo-mersenne Prime, Modular Reduction, Multiplication, Barrett Reduction, Electronic Signature, MAC, Finite Field, TLS.

1. Introduction

Modern information systems are characterized by inconceivable connections, without which they cannot function effectively. These systems can be distributed in both trusted and aggressive environments such as the local network and the external Internet. However, current recommendations for developing OWASP applications [1] emphasize the use of the secure HTTPS protocol based on the highly secure TLS v1.2 and TLS v1.3 protocols [2]. The TLS protocol employs various cryptographic algorithms that provide basic services such as confidentiality, integrity, authorship, and non-repudiation for data transmitted over the network.

TLS v1.3 has certain technical improvements, such as an optimized handshake procedure to establish a secure connection, and it also allows clients to resume interactions with servers more quickly. These methods are designed to minimize connection establishing delay and the amount of lost connections on unstable networks, which are frequently claimed as a justification to only provide unencrypted HTTP connections. A further reduction in the time for establishing a handshake is possible by reducing the time required to perform the operation of signature creation signing and verification, as well as key agreement [2], which are just based on the ECDSA-ECDH, EdDSA-EdDH algorithms, as well as using symmetric encryption in TLS v1.3 [2] according to AEAD-ChaCha20-Poly1305.

Various modern classical cryptographic algorithms, such as electronic signature ECDSA [3], EdDSA [4], key agreement scheme ECDH on elliptic curves [5] and on Edwards elliptic curves EdDH [6], short message encryption DSTU 9041:2020 [7] were analyzed. These algorithms use computationally heavy operations over points on elliptic curves over prime fields. Such operations as point scalar multiplication, point addition and point doubling based on operations over point coordinates as prime field elements. The speed up of modular reduction operation will speed up all arithmetic field operation.

Additionally an authenticated encryption with additional data AEAD-ChaCha20-Poly1305 [8], and Poly1305 [8] for calculating message authentication codes were analyzed as well. These algorithms based on operations over prime field elements. The speed up of modular reduction operation will speed up all arithmetic field operation.

The objective of this research is to reduce the time to establish a handshake and key agreement in TLS by reducing the time to perform the operations of electronic signature creation and verification by reducing the time to perform basic arithmetic operations in the prime field by reducing the time to perform the modulo reduction operation.

The object of research is the process of cryptographic transformations when establishing a TLS connection.

The subject of the study is the execution time of the modulo reduction operation, which underlies the arithmetic operations in the prime field.

In the process of research, the following tasks are solved: a review of existing modulo reduction algorithms of a special type - pseudo-Mersenne, which are used to form a prime field, which is used as a base for elliptic curves; development of optimized reduction algorithms by a special pseudo-Mersenne modulus, based on known algorithms; software implementation for modern hardware platforms is being carried out; comparison of optimized algorithms with the universal Barrett reduction algorithm.

2. Background

2.1. Prime Mersenne Modules and their Uses

In the prime fields, the operation of reduction by a large prime modulo plays a crucial role, as it is implicitly present in every operation in the field, such as addition, multiplication, squaring, exponentiation, root extraction, and inversion. Well-known algorithms for modulo reduction operations include Montgomery and Barrett algorithms [9,10]. Barrett algorithm is best for small arguments, and Montgomery method is best for sequential multiplications (ordinary exponentiation modulo). There is also a modification of the Barrett algorithm [11], which allows parallelizing the process of modulo reduction into two and multiply threads due to delayed carry and modified Barrett with single fold [12] requires incrementally more pre-computation, but reduces the overall amount of multiplication and addition that is required. While these algorithms have their own areas of application, they are computationally complex.

Researches [13,14] shows that the unordinary primes use are enable significantly simplify the reduction algorithms, in terms of computational complexity - reduce the number of arithmetic operations. In cryptography, they have found application: prime numbers of a general form, as well as prime numbers of a special form [15]:
prime Mersenne

$$p = b^m - 1 \tag{1}$$

pseudo-Mersenne

$$p = b^m - c \tag{2}$$

and as well as generalized pseudo-Mersenne

$$p = b^m + c_{n-1}b^{m-1} + \ldots c_0 \,. \tag{3}$$

This method is not linked to the simplicity of the modules to its size.

The “sparseness” property of such prime numbers of a special form [13-16] makes it possible to significantly reduce the number of arithmetic operations required for modulo reduction. The authors of this paper describe a modified algorithm for modular reduction of pseudo-Mersenne primes and their practical experience in efficiently implementing algorithms for modulo reduction operations, considering CPU architecture and the degree of the highest terms of the module.

2.2. *Based Modulo Reduction Algorithm*

Pseudo-Mersenne primes have become widely used in standards NIST FIPS 186-4 [17], SEC2 [18], DSTU 9041:2019 [7], RFC 8032 [4], and have the form $p = b^m - 1$, where the number $b = 2^w$, w - usually corresponds to the width machine word, m - is the number of machine words that the prime number occupies, and c - is a small integer that is not bigger than machine word size. The properties of these prime numbers were taken into account in the development of Algorithm 1 [15], shown in Fig.1. As input, the algorithm receives a large integer, which is obtained by multiplying two integers that do not bigger the modulus. This algorithm is given preference over Barrett algorithm because it uses fewer heavy operations, such as multiplications. Because it considers the structure of a prime number.

In step 1, a large double-precision integer z is represented as the higher and lower parts obtained as a result of division by b^m, as an integer quotient q and the remainder of the division r. In fact, it is necessary to operate with machine words: the higher machine words after b^m should be saved into q, and the lower words into r. Further, in step 2, as the result of multiplying the quotient by a small integer q, is calculated Rq. After that, in step 3, the remainder r of the division and the quotient q are added, followed by saving the result into C. Steps 4-5 are checked in most cases, while the loop in step 6 is performed no more than twice.

```
**Algorithm 1**. Reduction of $z < b^{2m}$ by a modulus $p = b^m - c$.
**Input**: $z < b^{2m}$, $p = b^m - c$, where $0 < c < b$ and $m \ge 2$.
**Output**: $zMODp$.
1. $q = z\,DIV\,b^m$, $r = z\,MOD\,b^m$
2. $(Rq, q) = q \cdot c$
3. $(C, r) = r + q$
4. if $Rq > 0$ then
    4.1 $(C1, r) = r + Rq \cdot c$
    4.2 $C = C + C1$
5. end if
6. while $C > 0$ or $r \ge p$ do
    6.1 $r = (r - p)\,MOD\,b^m$
    6.2 $C = C - 1$
7. end while
8. return $r$
```

Fig.1. Algorithm of reduction of $z < b^{2m}$ by a modulus $p = b^m - c$

3. Modified Modulo Reduction Method

Unfortunately, Algorithm 1 is applicable for prime numbers that have a power that is a divisible of the machine word width near the highest term, for example 2^{128} or 2^{256}. For modulo reduction, where the leading term has a degree that is not a divisible of the machine word width, Algorithm 1 requires significant modification.

For this it is necessary to consider three possible cases of modulo reduction of a large number of double precision, taking into account the CPU architecture, which will be discussed in more detail below: occupies an odd number of machine words, an example can be a pseudo-Mersenne module $p = 2^{130} - 5$ from RFC 8439 [8]; an even number of machine words, but the degree of the modulus is not a multiple of the machine word width, where the modulus $p = 2^{255} - 19$ from RFC 8032 [4] is used; the degree of the module is a multiple of the length of the machine word, where the pseudo-Mersenne modules $p = 2^{256} - 1539$, $p = 2^{384} - 7467$, $p = 2^{512} - 6579$ and $p = 2^{768} - 22467$ from DSTU 9041:2020 are also used.

The authors propose to align the higher part of the dividend: the bits of the dividend exceeding the modulus are aligned along the machine word boundary so that further operations can be performed with it. The modification of the lower part of the dividend was also performed, from which the higher part was removed - the bits of the dividend are higher than the module.

Suppose that w is the machine word width, according to the chosen architecture; n is the number of machine words that a large double precision number occupies. The number of machine words resulting from the multiplication is calculated as $n = \lceil 2 \cdot m / w \rceil$, and $a = m\ MOD\ w$ - number of significant modulo bits in high word.

As an example, consider the pseudo-Mersenne modulo of the form $p = 2^{130} - 5$, which is used in the Poly1305 message authentication code from RFC 8439 [8]. The result of the multiplication is a large double precision number, which is subject to modulo reduction, where the values of bits from 260 to 287 are zero, and all the rest are significant. Using the representation of the dividend in the form of machine words, the selection of the higher part and the lower part is performed. The higher part is aligned to the machine word boundary, and in the lower part the higher bits exceeding the modulus are set to zero. These actions are described in Algorithm 2 step 1-7, shown on Fig.3. Since the calculation result occupies only $\lceil n/2 \rceil$ machine words, the higher words r and q are set to zero. Fig.2 graphically shows quotient alignment on a machine word boundary for a 32-bit architecture (x86, ARM32, RV32) and clearing the higher part of the remainder from division.

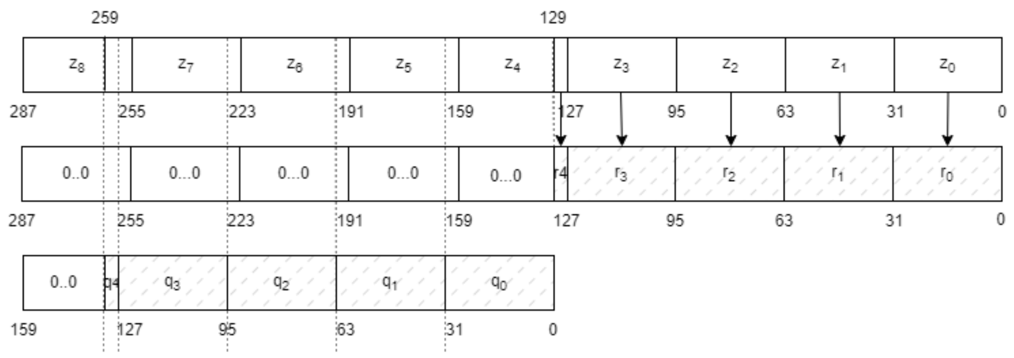


Fig.2. Graphical representation of the filling initial values r and q for a module $p = 2^{130} - 5$ with $w = 32$

Further, in step 8, the quotient q is multiplied by the number 5, and in the next step 9, the parameter d is calculated, which allows to find out how many significant bits are in the word after the previous multiplication and for further verification in step 13. The probability of an event hitting the block of the verification item 13 is 1/5, which is confirmed by empirical estimates. In steps 10-12, the quotient is added to the remainder of the division, after which the 30 most significant bits in the 5th word are set to zero and, which indicates the presence of non-zero least significant two bits, is calculated. The parameter c is necessary for checking in the loop of item 15, where the remainder of the division by the modulus decreases, until the remainder of the division becomes less than the modulo. Empirical estimates confirm that the probability of falling into the cycle of item 15 is 1/2 for a dividend that has a uniform distribution of unit bits, and its degree is close to the maximum permissible.

```
**Algorithm 2**. Reduction of $z < 2^{260}$ by a modulus $p = 2^{130} - 5$.
**Input**: $z < 2^{260}$, $p = 2^{130} - 5$, $w = 32$.
**Output**: $z = z \ MOD \ p$.
1. $q_0 = (z_5 << 30) | (z_4 >> 2)$
2. $q_1 = (z_6 << 30) | (z_5 >> 2)$
3. $q_2 = (z_7 << 30) | (z_6 >> 2)$
4. $q_3 = (z_8 << 30) | (z_7 >> 2)$
5. $q_4 = (z_8 >> 2) \& 0x00000003$
6. $r_0 = z_0$, $r_1 = z_1$, $r_2 = z_2$, $r_3 = z_3$
7. $r_4 = z_4 \& 0x00000003$
8. $q = 5 \cdot q$, $q_5 = 0$
9. $d = (q_4 >> 2)$, $q_4 \& = 0x00000003$
10. $r += q$
11. $c = ((r_5 << 30) | (r_4 >> 2))$, $r_5 = 0$
12. $r_4 \& = 0x00000003$
13. if $(d != 0)$ then
    13.1. $r += 5 \cdot d$
    13.2. $c += (r_4 >> 2)$, $r_5 = 0$
    13.3. $r_4 \& = 0x00000003$
14. end if
15. while $((c != 0) || (r > p))$ do
    15.1. $r -= p$, $r_5 = 0$
    15.2. $r_4 \& = 0x00000003$
    15.3. $c = (c != 0) \ ? \ (c - 1) : 0$
16. end while
17. return $r$.
```

Fig.3. Algorithm of reduction of $z < 2^{260}$ by a modulus $p = 2^{130} - 5$ with $w = 32$

The next example $p = 2^{255} - 19$ is the reduction of a pseudo-Mersenne prime from RFC 8032 [4]. When implementing Algorithm 3 (Fig.4) of reduction by such a modulo, a different tactic is proposed: shifts are performed in the opposite direction to form the initial values r and q. Because degree of modulo is not 1 bit enough to be the multiple of the machine word width. The difference from Algorithm 2 is that with each modification of r and q, it is necessary to set the most significant bit in the most significant word to zero.

In Algorithm 3, the probability of an event falling into the body of the check of step 17 is 2/3, and into the loop of step 19 is 2/5. It is confirmed by empirical estimates, for a dividend having a uniform distribution of non-zero bits and its degree is close to the maximum allowable.

The following is Algorithm 4 of reduction by $p = 2^{768} - 9659$ pseudo-Mersenne prime from DSTU 9041:2019

[7]. Unlike the previous Algorithms 2 and 3, the initial filling of the quotient q and the remainder r of the division is performed by ordinary copying without shifts of steps 1-4, since the degree of the modulus is a multiple of the length of the machine word. Fig. 5 graphically shows the filling of the quotient q and the remainder r of the division, for a 64-bit architecture (x86-64, ARM64, RV64).

```
Algorithm 3. Reduction of $z < 2^{510}$ by a modulus $p = 2^{255} - 19$.
Input: $z < 2^{510}$, $p = 2^{255} - 19$, $w = 32$.
Output: $r = z MOD\ p$.
1.  $q_0 = (z_8 << 1) | (z_7 >> 31)$
2.  $q_1 = (z_9 << 1) | (z_8 >> 31)$
3.  $q_2 = (z_{10} << 1) | (z_9 >> 31)$
4.  $q_3 = (z_{11} << 1) | (z_{10} >> 31)$
5.  $q_4 = (z_{12} << 1) | (z_{11} >> 31)$
6.  $q_5 = (z_{13} << 1) | (z_{12} >> 31)$
7.  $q_6 = (z_{14} << 1) | (z_{13} >> 31)$
8.  $q_7 = ((z_{15} << 1) | (z_{14} >> 31)) \& 0x7FFFFFFF$
9.  $r_0 = z_0$, $r_1 = z_1$ ,. $r_2 = z_2$, $r_3 = z_3$, $r_4 = z_4$,
    $r_5 = z_5$, $r_6 = z_6$
10. $r_7 = z_7 \& 0\times 7FFFFFFF$
11. $q = 19 \cdot q$
12. $d = ((q_8 << 1) | (q_7 >> 31))$
13. $q_7 \& = 0\times 7FFFFFFF$
14. $r += q$
15. $c = (r_7 >> 31)$
16. $r_7 \& = 0\times 7FFFFFFF$
17. if $(d != 0)$ then
    17.1. $r += 19 \cdot d$
    17.2. $c += (r_7 >> 31)$
    17.3. $r_7 \& = 0\times 7FFFFFFF$
18. end if
19. while $((c != 0) || (d > p))$ do
    19.1. $r -= p$
    19.2. $r_7 \& = 0\times 7FFFFFFF$
    19.3. $c = (c != 0)\ ?\ (c-1) : 0$
20. end while
21. return $r$
```

Fig.4. Algorithm of reduction of $z < 2^{510}$ by a modulus $p = 2^{255} - 19$ with $w = 32$

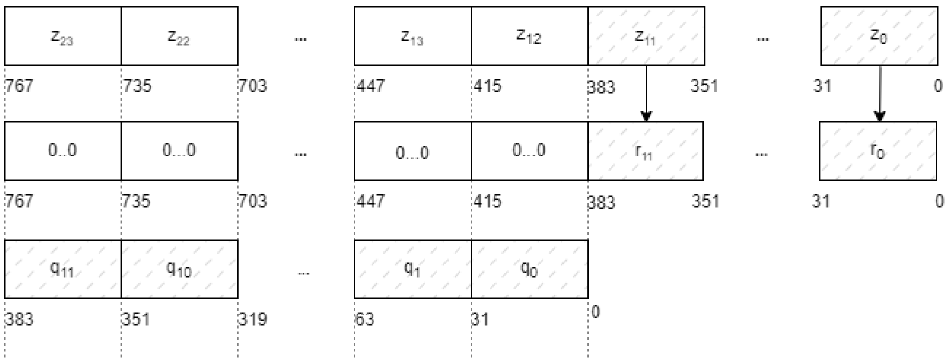


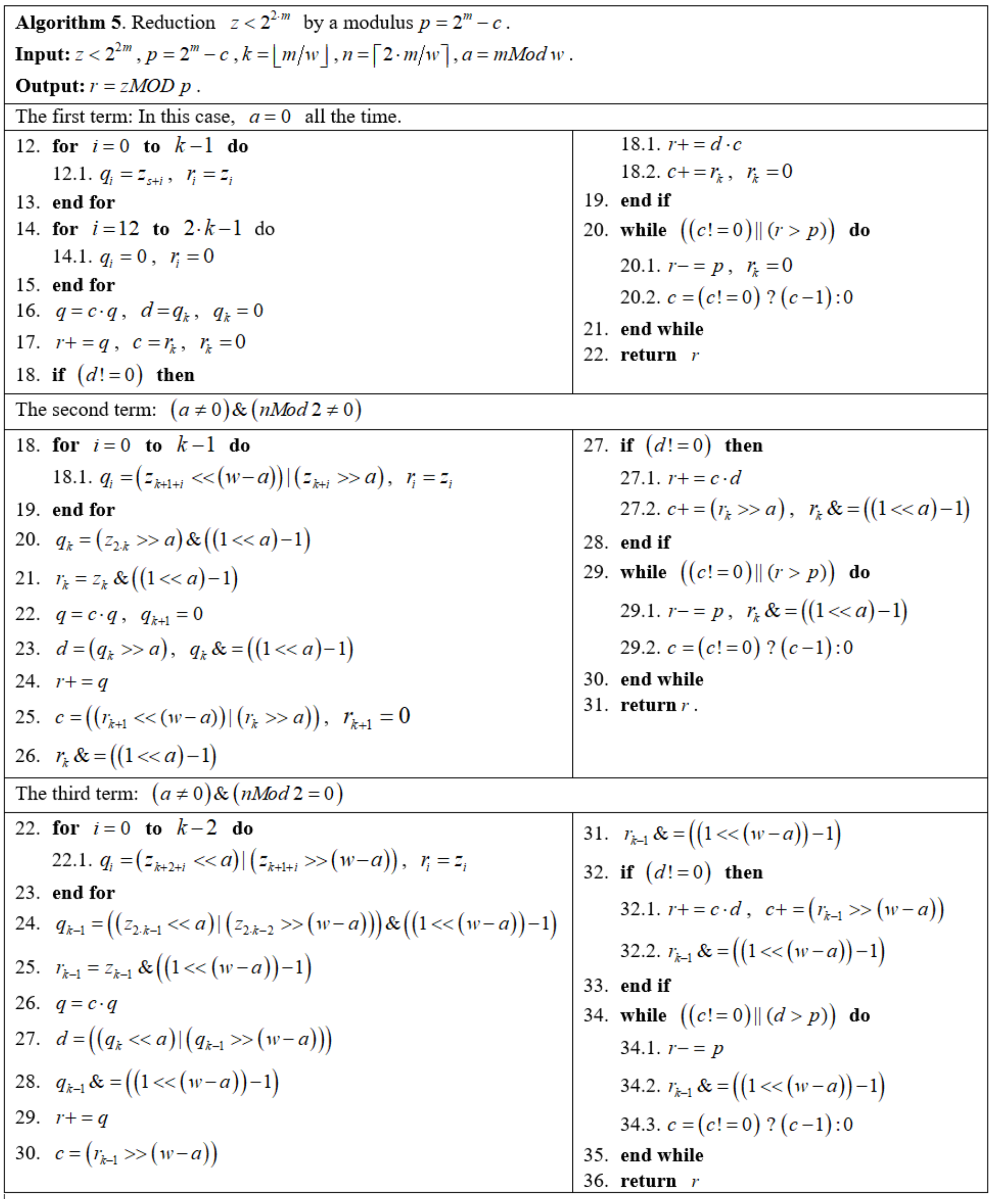
Fig.5. Graphical representation of the filling initial values r and q for a module $p = 2^{768} - 9659$ with $w = 64$

In Algorithm 4 (Fig.6), the probability of the event executing the body of the comparison in step 7 is 9/10, and in step 9 is 1/2. It is empirically confirmed, for a dividend having a uniform distribution of non-zero bits and its degree is close to the maximum allowable.

```
Algorithm 4. Reduction of $z < 2^{1536}$ by a modulus $p = 2^{768} - 9659$.
Input: $z < 2^{1536}$, $p = 2^{768} - 9659$, $w = 64$.
Output: $r = src\ MOD\ p$.
1.  for $i = 0$ to 11 do
    1.1. $q_i = z_{s+i}$, $r_i = z_i$
2.  end for
3.  for $i = 12$ to 23 do
    3.1. $q_i = 0$, $r_i = 0$
4.  end for
5.  $q = 9659 \cdot q$, $d = q_{12}$, $q_{12} = 0$
6.  $r += q$, $c = r_{12}$, $r_{12} = 0$
7.  if $(d != 0)$ then
    7.1. $r += d \cdot 9659$
    7.2. $c += r_{12}$, $r_{12} = 0$
8.  end if
9.  while $((c != 0) || (r > p))$ do
    9.1. $r -= p$, $r_{12} = 0$
    9.2. $c = (c != 0)\ ?\ (c-1) : 0$
10. end while
11. return $r$.
```

Fig.6. Algorithm of reduction of $z < 2^{1536}$ by a modulus $p = 2^{768} - 9659$ with $w = 64$

Based on the examples in this section, the following is a general Algorithm 5 for generating modulo large number reduction. With the help of which, based on the properties of a prime number, a developer can form the needed modulo reduction algorithm.



```
Algorithm 5. Reduction $z < 2^{2\cdot m}$ by a modulus $p = 2^m - c$.
Input: $z < 2^{2m}$, $p = 2^m - c$, $k = \lfloor m/w \rfloor$, $n = \lceil 2\cdot m/w \rceil$, $a = mMod\,w$.
Output: $r = zMOD\ p$.

The first term: In this case, $a = 0$ all the time.
12. for $i = 0$ to $k-1$ do
    12.1. $q_i = z_{s+i}$, $r_i = z_i$
13. end for
14. for $i = 12$ to $2\cdot k-1$ do
    14.1. $q_i = 0$, $r_i = 0$
15. end for
16. $q = c\cdot q$, $d = q_k$, $q_k = 0$
17. $r += q$, $c = r_k$, $r_k = 0$
18. if $(d != 0)$ then
    18.1. $r += d\cdot c$
    18.2. $c += r_k$, $r_k = 0$
19. end if
20. while $((c != 0) || (r > p))$ do
    20.1. $r -= p$, $r_k = 0$
    20.2. $c = (c != 0)\ ?\ (c-1) : 0$
21. end while
22. return $r$

The second term: $(a \neq 0)\,\&\,(nMod\,2 \neq 0)$
18. for $i = 0$ to $k-1$ do
    18.1. $q_i = (z_{k+1+i} << (w-a)) \,|\, (z_{k+i} >> a)$, $r_i = z_i$
19. end for
20. $q_k = (z_{2\cdot k} >> a)\,\&\,((1 << a)-1)$
21. $r_k = z_k\,\&\,((1 << a)-1)$
22. $q = c\cdot q$, $q_{k+1} = 0$
23. $d = (q_k >> a)$, $q_k\,\&= ((1 << a)-1)$
24. $r += q$
25. $c = ((r_{k+1} << (w-a)) \,|\, (r_k >> a))$, $r_{k+1} = 0$
26. $r_k\,\&= ((1 << a)-1)$
27. if $(d != 0)$ then
    27.1. $r += c\cdot d$
    27.2. $c += (r_k >> a)$, $r_k\,\&= ((1 << a)-1)$
28. end if
29. while $((c != 0) || (r > p))$ do
    29.1. $r -= p$, $r_k\,\&= ((1 << a)-1)$
    29.2. $c = (c != 0)\ ?\ (c-1) : 0$
30. end while
31. return $r$.

The third term: $(a \neq 0)\,\&\,(nMod\,2 = 0)$
22. for $i = 0$ to $k-2$ do
    22.1. $q_i = (z_{k+2+i} << a) \,|\, (z_{k+1+i} >> (w-a))$, $r_i = z_i$
23. end for
24. $q_{k-1} = ((z_{2\cdot k-1} << a) \,|\, (z_{2\cdot k-2} >> (w-a)))\,\&\,((1 << (w-a))-1)$
25. $r_{k-1} = z_{k-1}\,\&\,((1 << (w-a))-1)$
26. $q = c\cdot q$
27. $d = ((q_k << a) \,|\, (q_{k-1} >> (w-a)))$
28. $q_{k-1}\,\&= ((1 << (w-a))-1)$
29. $r += q$
30. $c = (r_{k-1} >> (w-a))$
31. $r_{k-1}\,\&= ((1 << (w-a))-1)$
32. if $(d != 0)$ then
    32.1. $r += c\cdot d$, $c += (r_{k-1} >> (w-a))$
    32.2. $r_{k-1}\,\&= ((1 << (w-a))-1)$
33. end if
34. while $((c != 0) || (d > p))$ do
    34.1. $r -= p$
    34.2. $r_{k-1}\,\&= ((1 << (w-a))-1)$
    34.3. $c = (c != 0)\ ?\ (c-1) : 0$
35. end while
36. return $r$
```

Fig.7. Method for generating the reduction algorithm by the pseudo-Mersenne modulo

4. Results

For confirmation of real algorithms efficiency, the performance measurement were being performed on AMD Ryzen 7 5800H@3.2 GHz CPU using Microsoft Visual C++ v14.2 compiler, considering 32 and 64-bit architectures. Randomly generated dividends with a uniform distribution of non-zero bits and a degree close to the maximum allowable were used as initial data.

Table 1 presents the timings for a sample of 10,000 experiments with a normal distribution and a 95% percentile. The results of evaluating the performance of modulo reduction, encryption and decryption, as well as signing and signature verifying are presented in it.

For clarity of results, Table 1 is padded with timings of modulo reduction by the Barrett algorithm and modulo reduction for the Mersenne prime $p = 2^{521} - 1$. The implementation for the 32-bit architecture is inferior to the 64-bit one by 2-2.5 times. The modifications of Algorithm 1 proposed in Algorithms 5 show their efficiency, in comparison with the Barrett algorithm, 8-9 times.

The above results demonstrate that the modulo reduction operation has a rather strong effect on the execution time of the cryptoprimitives itself, and even a slight decrease in the execution time of the modulo reduction operation can significantly reduce its execution time. For clarity, let's note that for the case of using EdDSA (RFC 8032) in TLS v1.2, the connection establishment time will be reduced by 2.5 times. The speed of encryption and decryption according to DSTU 9041:2020, using the proposed method, increases by 3-3.7 times.

Table 1. Performance estimates for modulo reduction by different algorithms for primes of a special form and it influence on corresponding cryptographic primitives

Module	Operation	Time, mks			
		x86		x86-64	
		Barrett's method	Modified method	Barrett's method	Modified method
RFC 8439					
$p = 2^{130} - 5$	Mod¹	0,1267	0,0158	0,0416	0,0046
	MAC²	53,2955 MB/s	177,039 MB/s	137,877 MB/s	318,428 MB/s
DSTU 9041:2020					
$p = 2^{256} - 1539$	Mod¹	0,3130	0,0344	0,0566	0,0053
	Enc/Dec³	2,891/ 2,87	1,029/ 0,811	0,389/ 0,423	0,161/ 0,169
$p = 2^{384} - 7467$	Mod¹	0,6399	0,0509	0,1031	0,0152
	Enc/Dec³	8,899/ 8,266	2,738/ 2,829	1,236/ 1,271	0,627/ 0.695
$p = 2^{512} - 6579$	Mod¹	1,068	0,0617	0,1692	0,0248
	Enc/Dec³	17,389/ 17,317	5.622/ 5.269	2,771/ 3,24	1,375/ 1,457
$p = 2^{768} - 22467$	Mod¹	2,0570	0,0858	0,3461	0,0299
	Enc/Dec³	56,277/ 53,738	14.651/ 16.878	8,848/ 8,698	3,777/ 3.953
RFC 8032					
$p = 2^{255} - 19$	Mod¹	0,3388	0,0174	0,0555	0,0058
	Sign/Verify⁴	193,408/ 1672,3	70,257/ 830,633	38,025/ 322,813	16,543/ 138,327
NIST FIPS 186-4					
$p = 2^{521} - 1$	Mod¹	1,1751	0,0225	0,2121	0,0047
	Sign/Verify⁵	1458,7/ 12674,9	655,29/ 3066,73	293.92/ 2304.35	11,64/ 697,32

1- timing of the modulo reduction operation, 2 - calculation time according to Poly1305 for a 3.5 kB message, 3 - encryption time according to DSTU 9041:2020 for a message size of 25 B, 4 - electronic signature creation and verification of message 64 B, 5 - electronic signature creation and verification of digest 64 B.

5. Conclusions

This article presents a modified method for reducing a large number by pseudo-Mersenne modulo, which takes into account the specifics of the degree of the highest term of a prime number, the width of the machine word, and initial alignment.

1. The proposed method makes it possible to reduce the time of cryptoprimitives execution, such as the signature creation and verification, which directly affects the time to establish a connection according to TLS thereby increasing the number of simultaneous connections, as well as the calculation of an authentication code. In fact, this is achieved by reducing the number of arithmetic operations.

2. The proposed modulo reduction generation method allows creating an algorithm for a specific module with the least number of arithmetic operations. This is applicable if the developer does not have a limits on program size. Otherwise, the universal Barrett algorithm is used, although inferior in speed.

3. When designing cryptosystems, one should select such modules, the degrees of which would be a multiple of the width of the machine word, now the most common, or close to it, like $p = 2^{255} - 19$.

The obtained results of the study can be used for all cryptographic primitives, where the pseudo-Mersenne modulo reduction operation is used. Thus, the reduction of large numbers modulo different from Mersenne and pseudo-Mersenne is a future direction for further research.

References

[1] Elie Saad and Rick Mitchell, "OWASP Web Security Testing Guide", OWASP Foundation, 2021. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/

[2] The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446, E. Rescorla, August 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446.

[3] Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), RFC 6979, T. Pornin, August 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6979

[4] Edwards-Curve Digital Signature Algorithm (EdDSA), RFC8032, S. Josefsson, I. Liusvaara, January 2017. [Online]. Available: https://tools.ietf.org/html/rfc8032

[5] Information technologies. Protection methods. Cryptographic methods based on elliptic curves. Generation of elliptic curves, DSTU ISO/IEC 15946-3:2019, August 2019.

[6] Elliptic Curves for Security, RFC7748, A. Langley, M. Hamburg, S. Turner, January 2016. [Online]. Available: https://tools.ietf.org/html/rfc7748.

[7] Information technologies. Cryptographic protection information. Short message encryption algorithm based on Edwards twisted elliptic curves, DSTU 9041:2020.

[8] ChaCha20 and Poly1305 for IETF Protocols, RFC8439, Y. Nir, A. Langley, June 2018. [Online]. Available: https://tools.ietf.org/html/rfc8439.

[9] P.D. Barrett, "Implementing the Riveat ShaInir and Adleman public key encryp tion algorithm on a standard digital signal processor," *Advances in Cryptology*, Proc. *Crypto'86*, *LNCS 263*, A.M. Odlyzko, Ed., Springer-Verlag, 1987, pp. 311- 323.

[10] C. D. Walter, "Hardware Aspects of Montgomery Modular Multiplication," in *Topics in Computational Number Theory Inspired by Peter L. Montgomery*, J. W. Bos and A. K. Lenstra, Eds. Cambridge: Cambridge University Press, 2017, pp. 40–81.

[11] A. Okhrimenko, V. Kovtun, "Method for Improving the Performance of a Prime Modulo Cast Operation," in *Information systems in management, education, industry*, Kharkiv, Ukraine, 2014, pp. 204–220.

[12] W. Hasenplaugh, G. Gaubatz and V. Gopal, *"Fast Modular Reduction," 18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, Montpellier, France, 2007, pp. 225-229, doi: 10.1109/ARITH.2007.18.

[13] J.A. Solinas, "Generalized Mersenne numbers," Technical Report CORR 99-39, University of Waterloo, 1999. [Online]. Available: https://www.researchgate.net/publication/238426205_Generalized_Mersenne_Prime

[14] H. Wu, "On Modular Reduction," July 2000. [Online]. Available: https://cacr.uwaterloo.ca/techreports/2000/tech_reports2000.html

[15] M. Taschwer, "Modular Multiplication Using Special Prime Moduli," in: Horster, P. (eds) *Kommunikationssicherheit im Zeichen des Internet*. DuD-Fachbeiträge. Vieweg+Teubner Verlag. https://doi.org/10.1007/978-3-322-89557-8_30

[16] J. C. Bajard, S. Duquesne, "Montgomery-friendly primes and applications to cryptography," in *Journal of Cryptographic Engineering*, 11(4), 399–415. https://doi.org/10.1007/s13389-021-00260-z

[17] Digital Signature Standard (DSS), NIST FIPS-186-4, National Institute of Standards and Technology, July 2013. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[18] SEC 2: Recommended Elliptic Curve Domain Parameters, SEC2, January 2010. [Online]. Available: https://www.secg.org/sec2-v2.pdf

Authors' Profiles

Mariia Kovtun: Ph.D of Information Security System. C++ software engineer at Cipher company and a researcher at the Research Laboratory for Counteracting Cyber Threats in the Aviation Industry (Computer Science and Technology) of the National Aviation University, Kyiv, Ukraine. Areas of scientific interests: applied cryptography, algebraic, elliptic and Edwards elliptic curves, efficient implementation, encryption with public key, cyber security.

Email: mg.kovtun@gmail.com

Vladyslav Kovtun: Project Manager at Cipher Company, Kyiv, Ukraine.

Areas of scientific interests: cryptography, algebraic curves, elliptic and hyper elliptic curves, efficient implementation.

Email: vlad.kovtun@cipher.com.ua

Oleksandr Stokipnyi: Head of R&D at Cipher Company, Kyiv, Ukraine

Areas of scientific interests: cryptography, efficient implementation distributed systems, public key infrastructure.

Email: alex.stokipny@cipher.com.ua

Andrew Okhrimenko: Senior Lecturer of the Department of system analysis and information technology of Mariupol State University, Kyiv, Ukraine. Areas of scientific interests: cryptography, large integer arithmetic's, efficient implementation.

Email: andrew.okhrimenko@gmail.com

How to cite this paper: Mariia Kovtun, Vladyslav Kovtun, Oleksandr Stokipnyi, Andrew Okhrimenko, "Software Implementation of Modular Reduction by Pseudo-mersenne Primes", International Journal of Information Technology and Computer Science(IJITCS), Vol.15, No.4, pp.1-9, 2023. DOI:10.5815/ijitcs.2023.04.01