Reduction Modulo $2^{448} - 2^{224} - 1$

Kaushik Nath and Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B. T. Road
Kolkata - 700108
India
{kaushikn_r,palash}@isical.ac.in

Abstract

An elliptic curve known as Curve448 over the finite field $\mathbb{F}_p$, where $p = 2^{448} - 2^{224} - 1$ has been proposed as part of the Transport Layer Security (TLS) protocol, version 1.3. Elements of $\mathbb{F}_p$ can be represented using 7 limbs where each limb is a 64-bit quantity. In this paper, we describe efficient algorithms for reduction modulo p that are required for performing field arithmetic in $\mathbb{F}_p$. A key feature of our algorithms is that we provide the relevant proofs of correctness. Based on the proofs of correctness we point out the incompleteness of the reduction methods in the previously known fastest code for implementing arithmetic in $\mathbb{F}_p$.

Keywords: Curve448, Goldilocks prime, modulo reduction, elliptic curve cryptography.

MSC (2010): 94A60.

1 Introduction

As part of the Transport Layer Security (TLS) protocol, version 1.3 [7], RFC 7748 [2] specifies the Montgomery form elliptic curve Curve448 and its birationally equivalent Edwards form elliptic curve Edwards448. The curve Edwards448 was originally proposed in [1] where it was named Ed448-Goldilocks. The underlying field for Curve448 and Edwards448 is $\mathbb{F}_p$ where p is the prime $2^{448} - 2^{224} - 1$.

Implementation of elliptic curve operations require arithmetic over the underlying field $\mathbb{F}_p$. Specifically, addition, subtraction, multiplication and squaring are required. Additionally, for implementing Montgomery ladder for Curve448, it is required to implement multiplication by a small constant. All of these operations require reduction modulo p.

For 64-bit architecture, an element of $\mathbb{F}_p$ can be represented using 7 limbs where each limb is a 64-bit quantity. Such a representation can be considered to be a packed or, saturated limb representation of the elements of $\mathbb{F}_p$. Alternatively, elements of $\mathbb{F}_p$ may be represented using 8 limbs where each limb is a 56-bit quantity stored in a 64-bit word. Such a representation can be considered to be a redundant or, unsaturated limb representation. For modern Intel processors such as Skylake and later processors, the implementation of field arithmetic using the saturated limb representation turns out to be faster than that of the unsaturated limb representation.

In this work, we consider the 7-limb saturated limb representation of elements of $\mathbb{F}_p$. Our focus is on the reduction algorithms which are required to implement field arithmetic operations in $\mathbb{F}_p$. For all the field arithmetic operations mentioned above, we present explicit reduction algorithms along with proofs of correctness. The algorithms proceed over several iterations successively reducing the size of the input. As part of the proof of correctness, it is required to argue that the algorithms terminate. The termination argument has a certain amount of subtlety. To the best of our knowledge, no previous work had provided the explicit reduction algorithms and consequently, also did not provide proofs of correctness.

The fastest publicly available software implementation for implementing Montgomery ladder of Curve448 using 7-limb representation is the program code corresponding to [4]. We have examined the code in details. It turns out that the reduction algorithms used in the code are incomplete. On certain kinds of inputs, the code will lead to overflow conditions and hence to incorrect results. This, however, is a very low probability event and cannot be captured using some randomly generated known answer tests (KATs). A brief discussion on this issue has been provided in Appendix A. We believe that

it is important to have proofs of correctness of the reduction algorithms to ensure that the algorithms works correctly for all possible inputs.

Remarks:

1. The present work considers saturated limb representation of the elements of $\mathbb{F}_p$. The unsaturated limb representation is relevant for Haswell and previous generations of processors. So, reduction algorithms and associated correctness proofs for the unsaturated limb representations are also of interest. Explicit descriptions of such algorithms and obtaining their proofs of correctness can be done in a manner similar to the case of reduction algorithms for the saturated limb representation that is described in the present work. The algorithms and the associated proofs for the unsaturated limb representation, however, are quite a bit simpler than that of the saturated limb representation. In view of this, we do not describe the algorithms and corresponding proofs of correctness for the unsaturated limb representation.

2. A previous work [3] provides explicit reduction algorithms and proofs of correctness for various pseudo-Mersenne primes. The prime $p = 2^{448} - 2^{224} - 1$ considered in the present work is not a pseudo-Mersenne prime. So, the reduction algorithms described in [3] do not apply to p. This necessitates the separate consideration of reduction algorithms and their proofs of correctness specifically for the prime p.

2 Arithmetic in $\mathbb{F}_p$

Let $p = 2^{448} - 2^{224} - 1$ and $\theta = 2^{64}$. For $d \geq 0$, define the polynomial

$$f(\theta) \quad = \quad f_0 + f_1\theta + \cdots + f_d\theta^d \tag{1}$$

where $f_0, f_1, \ldots, f_d$ are non-negative integers. Following convention, we will call the f_i's to be limbs of $f(\theta)$.

As mentioned above, we consider the 7-limb representation of the elements of $\mathbb{F}_p$. So, elements of $\mathbb{F}_p$ can be represented as a polynomial $f(\theta) = f_0 + f_1\theta + \cdots + f_6\theta^6$ where $0 \leq f_0, f_1, \ldots, f_6 < \theta$. Note that the set of all such $f(\theta)$ is in one-one correspondence with the set of integers $\{0, 1, \ldots, 2^{448}-1\}$. Since, $p < 2^{448} - 1$, a degree 6 polynomial $f(\theta)$ with $0 \leq f_0, f_1, \ldots, f_6 < \theta$ is not necessarily reduced modulo p. So, some elements of $\mathbb{F}_p$ have non-unique representation. This, however, is a not a problem for intermediate quantities in an elliptic curve computation. It is only the final result that is reduced to have a unique representation modulo p. Avoiding obtaining unique representations for the intermediate quantities leads to an overall faster algorithm for performing the elliptic curve computation. Consequently, given a polynomial $h(\theta) = h_0 + h_1\theta + \cdots + h_d\theta^d$, by reduction modulo p, we will denote the task of obtaining a polynomial $f(\theta) = f_0 + f_1\theta + \cdots + f_6\theta^6$ with $0 \leq f_0, f_1, \ldots, f_6 < \theta$ such that $f(\theta) \equiv h(\theta) \bmod p$.

For $i \geq 2$, let x and y be two $64i$-bit integers. Suppose, it is required to compute the integer product $x \cdot y$. If $x = y$, then this corresponds to the squaring operation, while if $x \neq y$, then a general multiplication operation is required. Intel processors from Broadwell (launched in 2014) onwards provide a special set of 64-bit multiplication and addition instructions which allow very fast computation of the product $x \cdot y$. For $i = 4$, the multiplication and squaring algorithms have been illustrated using diagrams in two Intel white papers [6, 5]. Explicit descriptions of the squaring and multiplication algorithms in the general case have been provided in [3].

A field multiplication/squaring in $\mathbb{F}_p$ consists of the following two broad steps. Suppose that $f(\theta)$ and $g(\theta)$ are two 7-limb integers from the set $\{0, 1, \ldots, 2^{448} - 1\}$ representing elements of $\mathbb{F}_p$. In the first step, the integer product $h(\theta)$ of $f(\theta)$ and $g(\theta)$ is obtained. The quantity $h(\theta)$ can be written as a 14-limb quantity $h(\theta) = h_0 + h_1\theta + \cdots + h_{13}\theta^{13}$, where $0 \leq h_0, h_1, \ldots, h_{13} < 2^{64}$. The second step consists of reducing $h(\theta)$ to a 7-limb integer which is congruent to $h(\theta)$ modulo p.

The Montgomery ladder algorithm for Curve448 requires multiplying a 7-limb quantity $f(\theta)$ by the constant $c = 39082$, which fits within a single 64-bit quantity. The integer product $c \cdot f(\theta)$ can be computed much faster than a general integer multiplication of two 7-limb quantities. The result $c \cdot f(\theta)$ can be written as an 8-limb quantity where all the limbs are 64-bit quantities. A reduction algorithm is to be applied to this 8-limb quantity to reduce it to a 7-limb quantity which represents an element of $\mathbb{F}_p$.

The integer addition of two 7-limb integers $f(\theta)$ and $g(\theta)$ results in an 8-limb integer. In this case, the last limb is a single bit. Nevertheless, the result of the addition has to be reduced to a 7-limb quantity.

Subtraction of two elements $f(\theta)$ and $g(\theta)$ in $\mathbb{F}_p$ is more problematic. The integer operation $f(\theta) - g(\theta)$ can turn out to be negative. To avoid handling negative numbers a suitable multiple of p is added to the result. This creates subtleties in the reduction algorithm.

3 Reduction in $\mathbb{F}_p$

It has been mentioned in Section 2, that the integer product of two 7-limb quantities results in a 14-limb quantity which has to be reduced to a 7-limb quantity. Further, multiplication of a 7-limb quantity by a 64-bit constant and the addition of two 7-limb quantities both lead an 8-limb quantity which has to be reduced to a 7-limb quantity.

In Section 3.1 below, we describe the method for reducing a 14-limb quantity to a 7-limb quantity. As part of this algorithm, it is required to reduce an 8-limb quantity to a 7-limb quantity. Correspondingly, this part can be used to reduce the result obtained either after multiplication by a 64-bit constant or after addition of two 7-limb quantities. This is pointed out in Section 3.2. The case of subtraction in $\mathbb{F}_p$ is described in Section 3.3.

3.1 Reduction from 14-Limb to 7-Limb

Let $h(\theta)$ be the 14-limb polynomial which is to be reduced. The polynomial $h(\theta)$ represents an integer z of $2 \cdot 448 = 896$ bits. A top-level view of the reduction algorithm is the following.

$$\begin{array}{l} \text{for } i \leftarrow 0 \text{ to } 3 \text{ do} \\ \quad z \leftarrow (z \bmod 2^{448}) + (2^{224}+1)\lfloor z/2^{448} \rfloor \\ \text{end for.} \end{array} \tag{2}$$

A formal description of the idea in (2) is given in Function reduce448 of Algorithm 1. All the operations in Function reduce448 can be performed using 64-bit arithmetic instructions available in modern processors. For showing correctness of the algorithm it is required to argue that the output is indeed congruent to the input modulo p. Further, it is also required to argue that the procedure terminates without any overflow.

Let $h^{(0)}(\theta) = h(\theta)$. Function reduce448 takes the 14-limb polynomial $h^{(0)}(\theta)$ as input and reduces it through the intermediate polynomials $h^{(1)}(\theta)$, $h^{(2)}(\theta)$, $h^{(3)}(\theta)$ finally producing the 7-limb output polynomial $h^{(4)}(\theta)$. A summary of the properties of the polynomials $h^{(1)}(\theta)$, $h^{(2)}(\theta)$, $h^{(3)}(\theta)$ and $h^{(4)}(\theta)$ and the different steps of reduce448 that produces these polynomials are as follows:

- $h^{(1)}(\theta)$ has 11 limbs. The first 10 limbs of $h^{(1)}(\theta)$ are 64-bit quantities while the last limb is a 33-bit quantity. The computation of $h^{(1)}(\theta)$ is achieved by Steps 4-14.
- $h^{(2)}(\theta)$ has 8 limbs. The last limb is 2 bits long. The computation of $h^{(2)}(\theta)$ from $h^{(1)}(\theta)$ is achieved by Steps 15-25.
- $h^{(3)}(\theta)$ has 8 limbs. The last limb is 1-bit long and further, if $h_7^{(3)} = 0$, then $h_4^{(3)} = h_5^{(3)} = h_6^{(3)} = 0$. The computation of $h^{(3)}(\theta)$ from $h^{(2)}(\theta)$ is achieved by Steps 26-32.
- $h^{(4)}(\theta)$ has 7 limbs where each limb is a 64-bit quantity. The computation of $h^{(4)}(\theta)$ from $h^{(3)}(\theta)$ is achieved by Steps 33-37.

We have stated some properties of $h^{(1)}(\theta)$, $h^{(2)}(\theta)$, $h^{(3)}(\theta)$ and $h^{(4)}(\theta)$. These are formally proved in Theorem 1. We note some points.

1. The last limb of $h^{(2)}(\theta)$ is a 2-bit quantity which is stored in a 64-bit word. The fact that the other 62 bits of the last limb do not store useful information does not help the rest of reduction.
2. The property of $h^{(3)}(\theta)$ that if $h_7^{(3)} = 0$, then $h_4^{(3)} = h_5^{(3)} = h_6^{(3)} = 0$ is required to argue that the procedure terminates without any overflow in the next iteration.

Theorem 1. *Suppose the input* $h^{(0)}(\theta) = h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_{13}^{(0)}\theta^{13}$ *to* reduce448 *is such that* $0 \le h_i^{(0)} < 2^{64}$ *for* $i = 0, 1, \ldots, 13$. *Then the output* $h^{(4)}(\theta)$ *of* reduce448 *is such that* $h^{(4)}(\theta) = h_0^{(4)} + h_1^{(4)}\theta + \cdots + h_6^{(4)}(\theta)$ *with* $0 \le h_j^{(4)} < 2^{64}$ *for* $j = 0, 1, \ldots, 6$. *Further,* $h^{(4)}(\theta) \equiv h^{(0)}(\theta) \bmod p$.

Proof. We have the prime $p = 2^{448} - 2^{224} - 1$, and since $\theta = 2^{64}$, we have

$$2^{448} \;=\; \theta^7 \;\equiv\; 2^{224} + 1 \;=\; 2^{\eta/2}\theta^3 + 1 \mod p, \tag{3}$$

where $\eta = 64$.

Algorithm 1 Reduction from 14-limb to 7-limb. In the algorithm, $\eta = 64$.

function reduce448($h^{(0)}(\theta)$)
input: $h^{(0)}(\theta) = h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_{13}^{(0)}\theta^{13}$ such that $0 \le h_i^{(0)} < 2^{64}$ for $i = 0, 1, \ldots, 13$.
output: $h^{(4)}(\theta) = h_0^{(4)} + h_1^{(4)}\theta + \cdots + h_1^{(4)}\theta^6$ such that $0 \le h_i^{(4)} < 2^{64}$ for $i = 0, 1, \ldots, 6$ and $h^{(4)}(\theta) \equiv h^{(0)}(\theta) \bmod p$.
    $t \leftarrow h_0^{(0)} + h_7^{(0)}$; $r_0^{(0)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    for $i \leftarrow 1$ to 6 **do** $t \leftarrow h_i^{(0)} + h_{7+i}^{(0)} +$ carry; $r_i^{(0)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$ **end for**
    $r_7^{(0)} \leftarrow$ carry
    $s_{10}^{(0)} \leftarrow \lfloor h_{13}^{(0)}/2^{\eta/2} \rfloor$
    for $i \leftarrow 9$ down to 4 **do** $s_i^{(0)} \leftarrow (2^{\eta/2} h_{i+4}^{(0)}) \bmod 2^\eta + \lfloor h_{i+3}^{(0)}/2^{\eta/2} \rfloor$ **end for**
    $s_3^{(0)} \leftarrow (2^{\eta/2} h_7^{(0)}) \bmod 2^\eta$
    $h_0^{(1)} \leftarrow r_0^{(0)}$; $h_1^{(1)} \leftarrow r_1^{(0)}$; $h_2^{(1)} \leftarrow r_2^{(0)}$
    $t \leftarrow s_3^{(0)} + r_3^{(0)}$; $h_3^{(1)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    for $i \leftarrow 4$ to 7 **do** $t \leftarrow s_i^{(0)} + r_i^{(0)} +$ carry; $h_i^{(1)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$ **end for**
    for $i \leftarrow 8$ to 9 **do** $t \leftarrow s_i^{(0)} +$ carry; $h_i^{(1)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$ **end for**
    $h_{10}^{(1)} \leftarrow s_{10}^{(0)} +$ carry
    $t \leftarrow h_0^{(1)} + h_7^{(1)}$; $r_0^{(1)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    for $i \leftarrow 1$ to 3 **do** $t \leftarrow h_i^{(1)} + h_{7+i}^{(1)} +$ carry; $r_i^{(1)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$ **end for**
    for $i \leftarrow 4$ to 6 **do** $t \leftarrow h_i^{(1)} +$ carry; $r_i^{(1)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$ **end for**
    $r_7^{(1)} \leftarrow$ carry
    $s_7^{(1)} \leftarrow \lfloor h_{10}^{(1)}/2^{\eta/2} \rfloor$
    for $i \leftarrow 6$ down to 4 **do** $s_i^{(1)} \leftarrow (2^{\eta/2} h_{i+4}^{(1)}) \bmod 2^\eta + \lfloor h_{i+3}^{(1)}/2^{\eta/2} \rfloor$ **end for**
    $s_3^{(1)} \leftarrow (2^{\eta/2} h_7^{(1)}) \bmod 2^\eta$
    $h_0^{(2)} \leftarrow r_0^{(1)}$; $h_1^{(2)} \leftarrow r_1^{(1)}$; $h_2^{(2)} \leftarrow r_2^{(1)}$
    $t \leftarrow s_3^{(1)} + r_3^{(1)}$; $h_3^{(2)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    for $i \leftarrow 4$ to 6 **do** $t \leftarrow s_i^{(1)} + r_i^{(1)} +$ carry; $h_i^{(2)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$ **end for**
    $h_7^{(2)} \leftarrow s_7^{(1)} + r_7^{(1)} +$ carry
    $t \leftarrow h_0^{(2)} + h_7^{(2)}$; $h_0^{(3)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_1^{(2)} +$ carry; $h_1^{(3)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_2^{(2)} +$ carry; $h_2^{(3)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_3^{(2)} + (2^{\eta/2} h_7^{(2)}) \bmod 2^\eta +$ carry; $h_3^{(3)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_4^{(2)} +$ carry; $h_4^{(3)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_5^{(2)} +$ carry; $h_5^{(3)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_6^{(2)} +$ carry; $h_6^{(3)} \leftarrow t \bmod 2^\eta$; $h_7^{(3)} \leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_0^{(3)} + h_7^{(3)}$; $h_0^{(4)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_1^{(3)} +$ carry; $h_1^{(4)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_2^{(3)} +$ carry; $h_2^{(4)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $t \leftarrow h_3^{(3)} + (2^{\eta/2} h_7^{(3)}) \bmod 2^\eta +$ carry; $h_3^{(4)} \leftarrow t \bmod 2^\eta$; carry $\leftarrow \lfloor t/2^\eta \rfloor$
    $h_4^{(4)} \leftarrow h_4^{(3)} +$ carry; $h_5^{(4)} \leftarrow h_5^{(3)}$; $h_6^{(4)} \leftarrow h_6^{(3)}$
    return $h^{(4)}(\theta) = h_0^{(4)} + h_1^{(4)}\theta + \cdots + h_6^{(4)}\theta^6$
end function.

Reduction from $h^{(0)}(\theta)$ to $h^{(1)}(\theta)$. The input $h^{(0)}(\theta)$ to reduce448 can be written as

$$\begin{aligned}
h^{(0)}(\theta) &= (h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_6^{(0)}\theta^6) + (h_7^{(0)}\theta^7 + h_8^{(0)}\theta^8 + \cdots + h_{13}^{(0)}\theta^{13}), \\
&= (h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_6^{(0)}\theta^6) + (h_7^{(0)} + h_8^{(0)}\theta + \cdots + h_{13}^{(0)}\theta^6)\theta^7, \\
&\equiv (h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_6^{(0)}\theta^6) + (h_7^{(0)} + h_8^{(0)}\theta + \cdots + h_{13}^{(0)}\theta^6)(2^{\eta/2}\theta^3 + 1), \text{ [using (3)]}, \\
&= (h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_6^{(0)}\theta^6) + (h_7^{(0)} + h_8^{(0)}\theta + \cdots + h_{13}^{(0)}\theta^6) + \\
&\quad (h_7^{(0)}\theta^3 + h_8^{(0)}\theta^4 + \cdots + h_{13}^{(0)}\theta^9)2^{\eta/2}. \qquad (4)
\end{aligned}$$

Steps 4-6 add the two polynomials $(h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_6^{(0)}\theta^6)$ and $(h_7^{(0)} + h_8^{(0)}\theta + \cdots + h_{13}^{(0)}\theta^6)$ limb-wise by forwarding the 1-bit carry, producing the polynomial $(r_0^{(0)} + r_1^{(0)}\theta + \cdots + r_7^{(0)}\theta^7)$. Hence, from (4) we write

$$h^{(0)}(\theta) \equiv \underbrace{(r_0^{(0)} + r_1^{(0)}\theta + \cdots + r_7^{(0)}\theta^7)}_{\text{through Steps 4-8}} + (h_7^{(0)}\theta^3 + h_8^{(0)}\theta^4 + \cdots + h_{13}^{(0)}\theta^9)2^{\eta/2}. \tag{5}$$

where $0 \leq r_0^{(0)}, r_1^{(0)}, \ldots, r_6^{(0)} < 2^\eta$, and $0 \leq r_7^{(0)} \leq 1$. For $j = 7, 8, \ldots, 13$, let us define

$$h_j^{(0)} = h_{j,0}^{(0)} + h_{j,1}^{(0)}2^{\eta/2}, \text{ where } h_{j,0}^{(0)} = h_j^{(0)} \bmod 2^{\eta/2}, \text{ and } h_{j,1}^{(0)} = \lfloor h_j^{(0)}/2^{\eta/2} \rfloor. \tag{6}$$

Using the definitions given in (6), (5) can be further written as

$$\begin{aligned} h^{(0)}(\theta) &\equiv (r_0^{(0)} + r_1^{(0)}\theta + \cdots + r_7^{(0)}\theta^7) + \\ &\quad ((h_{7,0}^{(0)} + h_{7,1}^{(0)}2^{\eta/2})\theta^3 + (h_{8,0}^{(0)} + h_{8,1}^{(0)}2^{\eta/2})\theta^4 + \cdots + (h_{13,0}^{(0)} + h_{13,1}^{(0)}2^{\eta/2})\theta^9)2^{\eta/2}, \\ &= (r_0^{(0)} + r_1^{(0)}\theta + \cdots + r_7^{(0)}\theta^7) + \\ &\quad 2^{\eta/2}h_{7,0}^{(0)}\theta^3 + (h_{7,1}^{(0)} + 2^{\eta/2}h_{8,0}^{(0)})\theta^4 + (h_{8,1}^{(0)} + 2^{\eta/2}h_{9,0}^{(0)})\theta^5 + \cdots + \\ &\quad (h_{12,1}^{(0)} + 2^{\eta/2}h_{13,0}^{(0)})\theta^9 + h_{13,1}^{(0)}\theta^{10}, \text{ [since } \theta = 2^\eta\text{]}. \end{aligned} \tag{7}$$

Steps 7-9 perform the computations given in the last two lines of (7) to produce the polynomial $(s_3^{(0)}\theta^3 + s_4^{(0)}\theta^4 + \cdots + s_{10}^{(0)}\theta^{10})$, where $0 \leq s_3^{(0)}, s_4^{(0)}, \cdots, s_9^{(0)} < 2^\eta$, and $0 \leq s_{10}^{(0)} < 2^{\eta/2}$. Then, Step 10 copies $r_j^{(0)}$ to $h_j^{(1)}, j = 0, 1, 2$ followed by Steps 11-14, which add the two polynomials $(r_3^{(0)}\theta^3 + r_4^{(0)}\theta^4 + \cdots + r_7^{(0)}\theta^7)$ and $(s_0^{(0)}\theta^3 + s_1^{(0)}\theta^4 + \cdots + s_7^{(0)}\theta^{10})$ producing the polynomial $h^{(1)}(\theta) = h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^{10}$. Hence, from (7) we write

$$\begin{aligned} h^{(0)}(\theta) &\equiv (r_0^{(0)} + r_1^{(0)}\theta + \cdots + r_7^{(0)}\theta^7) + \underbrace{(s_3^{(0)}\theta^3 + s_4^{(0)}\theta^4 + \cdots + s_{10}^{(0)}\theta^{10})}_{\text{through Steps 7-9}}, \\ &= \underbrace{(h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^{10})}_{\text{through Steps 11-14}} = h^{(1)}(\theta), \end{aligned} \tag{8}$$

where $0 \leq h_0^{(1)}, h_1^{(1)}, \cdots, h_9^{(1)} < 2^\eta$, and $0 \leq h_{10}^{(1)} \leq 2^{\eta/2}$.

Reduction from $h^{(1)}(\theta)$ to $h^{(2)}(\theta)$. Polynomial $h^{(1)}(\theta)$ can further be written as

$$\begin{aligned} h^{(1)}(\theta) &= (h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6) + (h_7^{(1)} + h_8^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^3)\theta^7, \\ &\equiv (h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6) + (h_7^{(1)} + h_8^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^3)(2^{\eta/2}\theta^3 + 1), \text{ [using (3)]}, \\ &= (h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6) + (h_7^{(1)} + h_8^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^3) + \\ &\quad (h_7^{(1)}\theta^3 + h_8^{(1)}\theta^4 + \cdots + h_{10}^{(1)}\theta^6)2^{\eta/2}. \end{aligned} \tag{9}$$

Steps 15-18 adds the two polynomials $(h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6)$ and $(h_7^{(1)} + h_8^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^3)$ limb-wise by forwarding the 1-bit carry, producing the polynomial $(r_0^{(1)} + r_1^{(1)}\theta + \cdots + r_7^{(1)}\theta^7)$. Hence, from (9) we write

$$\begin{aligned} h^{(1)}(\theta) &\equiv (h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6) + (h_7^{(1)} + h_8^{(1)}\theta + \cdots + h_{10}^{(1)}\theta^3) + \\ &\quad (h_7^{(1)}\theta^3 + h_8^{(1)}\theta^4 + h_9^{(1)}\theta^5 + h_{10}^{(1)}\theta^6)2^{\eta/2}, \\ &= \underbrace{(r_0^{(1)} + r_1^{(1)}\theta + \cdots + r_7^{(1)}\theta^7)}_{\text{through Steps 15-18}} + (h_7^{(1)}\theta^3 + h_8^{(1)}\theta^4 + \cdots + h_{10}^{(1)}\theta^6)2^{\eta/2}. \end{aligned} \tag{10}$$

where $0 \leq r_0^{(1)}, r_1^{(1)}, \ldots, r_6^{(1)} < 2^\eta$, and $0 \leq r_7^{(1)} \leq 1$. For $j = 7, 8, 9, 10$, let us define

$$h_j^{(1)} = h_{j,0}^{(1)} + h_{j,1}^{(1)}2^{\eta/2}, \text{ where } h_{j,0}^{(1)} = h_j^{(1)} \bmod 2^{\eta/2}, \text{ and } h_{j,1}^{(1)} = \lfloor h_j^{(1)}/2^{\eta/2} \rfloor. \tag{11}$$

Using the definitions given in (11), (10) can be further written as

$$\begin{aligned} h^{(1)}(\theta) &\equiv (r_0^{(1)} + r_1^{(1)}\theta + \cdots + r_7^{(1)}\theta^7) + \\ &\quad ((h_{7,0}^{(1)} + h_{7,1}^{(1)}2^{\eta/2})\theta^3 + (h_{8,0}^{(1)} + h_{8,1}^{(1)}2^{\eta/2})\theta^4 + \cdots + (h_{10,0}^{(1)} + h_{10,1}^{(1)}2^{\eta/2})\theta^6)2^{\eta/2}, \\ &= (r_0^{(1)} + r_1^{(1)}\theta + \cdots + r_7^{(1)}\theta^7) + \\ &\quad 2^{\eta/2}h_{7,0}^{(1)}\theta^3 + (h_{7,1}^{(1)} + 2^{\eta/2}h_{8,0}^{(1)})\theta^4 + (h_{8,1}^{(1)} + 2^{\eta/2}h_{9,0}^{(1)})\theta^5 + (h_{9,1}^{(1)} + 2^{\eta/2}h_{10,0}^{(1)})\theta^6 + \\ &\quad h_{10,1}^{(1)}\theta^7 \text{ [since } \theta = 2^\eta\text{]}. \end{aligned} \tag{12}$$

Steps 19-21 perform the computations in the second and third line of (12) to produce the polynomial $(s_3^{(1)}\theta^3 + s_4^{(1)}\theta^4 + \cdots + s_7^{(1)}\theta^7)$, where $0 \le s_3^{(1)}, s_4^{(1)}, s_5^{(1)}, s_6^{(1)} < 2^\eta$, and $0 \le s_7^{(1)} \le 1$. Then, Step 22 copies $r_j^{(1)}$ to $h_j^{(2)}$, $j = 0, 1, 2$ followed by Steps 23-25, which add the two polynomials $(r_3^{(1)}\theta^3 + r_4^{(1)}\theta^4 + \cdots + r_7^{(1)}\theta^7)$ and $(s_3^{(1)}\theta^3 + s_4^{(1)}\theta^4 + \cdots + s_7^{(1)}\theta^7)$ producing the polynomial $h^{(2)}(\theta) = h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_7^{(2)}\theta^7$. Hence, from (12) we write

$$\begin{aligned} h^{(1)}(\theta) &\equiv (r_0^{(1)} + r_1^{(1)}\theta + \cdots + r_7^{(1)}\theta^7) + \underbrace{(s_3^{(1)}\theta^3 + s_4^{(1)}\theta^4 + \cdots + s_7^{(1)}\theta^7}_{\text{through Steps 19-21}}, \\ &= \underbrace{(h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_7^{(2)}\theta^7)}_{\text{through Steps 22-25}} = h^{(2)}(\theta), \end{aligned} \tag{13}$$

where $0 \le h_0^{(2)}, h_1^{(2)}, \cdots, h_6^{(2)} < 2^\eta$, and $0 \le h_7^{(2)} < 2^2$.

Reduction from $h^{(2)}(\theta)$ to $h^{(3)}(\theta)$. Polynomial $h^{(2)}(\theta)$ can further be written as

$$\begin{aligned} h^{(2)}(\theta) &\equiv h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6 + h_7^{(2)}(2^{\eta/2}\theta^3 + 1), \text{ [using (3)]}, \\ &= (h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6) + (h_7^{(2)} + 2^{\eta/2}h_7^{(2)}\theta^3). \end{aligned} \tag{14}$$

Steps 26-32 adds the polynomial $(2^{\eta/2}\theta^3 + 1)h_7^{(2)} = (h_7^{(2)} + 2^{\eta/2}h_7^{(2)}\theta^3)$ to the polynomial $(h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6)$, which produces $(h_0^{(3)} + h_1^{(3)}\theta + \cdots + h_7^{(3)}\theta^7)$, where $0 \le h_0^{(3)}, h_1^{(3)}, \cdots, h_6^{(3)} < 2^\eta$, and $0 \le h_7^{(3)} \le 1$. Hence, from (14) we write

$$\begin{aligned} h^{(2)}(\theta) &\equiv (h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6) + (h_7^{(2)} + 2^{\eta/2}h_7^{(2)}\theta^3), \\ &= \underbrace{(h_0^{(3)} + h_1^{(3)}\theta + \cdots + h_7^{(3)}\theta^7)}_{\text{through Steps 26-32}} = h^{(3)}(\theta), \end{aligned} \tag{15}$$

where $0 \le h_0^{(3)}, h_1^{(3)}, \cdots, h_6^{(3)} < 2^\eta$, and $0 \le h_7^{(3)} \le 1$. In Step 32, $h_7^{(3)} = 1$ if and only if $h_6^{(2)} = 2^\eta - 1$ and $\mathsf{carry} = 1$, which implies $t = 2^\eta$, which further implies $h_6^{(3)} = t \bmod 2^\eta = 0$. In a similar manner, it can be argued that the condition $\mathsf{carry} = 1$ arises in Step 32 if and only if Step 31 results in setting $h_5^{(3)}$ to 0 and continuing one more step backwards, Step 30 results in setting $h_4^{(3)}$ to 0. Hence, if $h_7^{(3)} = 1$ we have the conditions

$$h_4^{(3)} = h_5^{(3)} = h_6^{(3)} = 0. \tag{16}$$

Reduction from $h^{(3)}(\theta)$ to $h^{(4)}(\theta)$. Polynomial $h^{(3)}(\theta)$ can further be written as

$$\begin{aligned} h^{(3)}(\theta) &\equiv h_0^{(3)} + h_1^{(3)}\theta + \cdots + h_6^{(3)}\theta^6 + h_7^{(3)}(2^{\eta/2}\theta^3 + 1), \text{ [using (3)]}, \\ &= (h_0^{(3)} + h_1^{(3)}\theta + \cdots + h_6^{(3)}\theta^6) + (h_7^{(3)} + 2^{\eta/2}h_7^{(3)}\theta^3). \end{aligned} \tag{17}$$

If $h_7^{(3)} = 0$, then after Steps 33-37 we get $h_j^{(4)} = h_j^{(3)}, j = 0, 1, \ldots, 6$; else, if $h_7^{(3)} = 1$, then using (16) we can say that the reduction surely terminates by the addition in Step 37. This implies after Steps 33-37 $0 \le h_j^{(4)} < 2^\eta, j = 0, 1, 2, 3$, and $h_4^{(4)} = 1, h_5^{(4)} = 0, h_6^{(4)} = 0$. Hence, in any case from (17) it follows that

$$\begin{aligned} h^{(3)}(\theta) &\equiv (h_0^{(3)} + h_1^{(3)}\theta + \cdots + h_6^{(3)}\theta^6) + (h_7^{(3)} + 2^{\eta/2}h_7^{(3)}\theta^3), \\ &= \underbrace{(h_0^{(4)} + h_1^{(4)}\theta + \cdots + h_6^{(4)}\theta^6)}_{\text{through Steps 33-37}} = h^{(4)}(\theta), \end{aligned} \tag{18}$$

where $0 \le h_0^{(4)}, h_1^{(4)}, \cdots, h_6^{(4)} < 2^\eta$. Also, by combining (8), (13), (15), and (18) we have $h^{(4)}(\theta) \equiv h^{(0)}(\theta) \bmod p$, which proves the theorem. □

3.2 Reduction from 8-Limb to 7-Limb

The proof of Theorem 1 establishes the bound $0 \leq h_7^{(2)} < 4$. The reduction from $h^{(2)}(\theta)$ to $h^{(3)}(\theta)$ performed by reduce448, however, only assumes that $h_7^{(2)}$ is a 64-bit quantity. In particular, the proof of this part of the reduction does not use the fact that $h_7^{(2)}$ is a 2-bit quantity. Also, we know of no way to speed up the reduction by using the bound $h_7^{(2)} < 4$.

In view of the above, given an 8-limb quantity, the reduction to 7-limb can be performed as follows. Consider the 8-limb quantity to be $h^{(2)}(\theta)$ and apply the part of reduce448 which reduces $h^{(2)}(\theta)$ to $h^{(4)}(\theta)$. The correctness of the reduction is guaranteed by the part of the proof of Theorem 1 which argues the correctness of the reduction from $h^{(2)}(\theta)$ to $h^{(3)}(\theta)$ and from $h^{(3)}(\theta)$ to $h^{(4)}(\theta)$.

3.3 Subtraction

Let $f(\theta)$ and $g(\theta)$ be 7-limb quantities representing elements of $\mathbb{F}_p$. The requirement is to compute $(f(\theta) - g(\theta)) \bmod p$. Function sub448 of Algorithm 2 performs this computation.

The description of Function sub448 uses the instruction sub which is defined as follows. Let x and y be 64-bit quantities and $\mathfrak{b}_0$ be a bit. The instruction $\mathsf{sub}(x, y, \mathfrak{b}_0)$ produces as output the pair $(z, \mathfrak{b}_1)$ where z is a 64-bit quantity and $\mathfrak{b}_1$ is a bit. The definitions of z and $\mathfrak{b}_1$ are as follows.

$$z = \begin{cases} x - (y + \mathfrak{b}_0) & \text{if } x \geq y + \mathfrak{b}_0, \\ 2^{64} + x - (y + \mathfrak{b}_0) & \text{if } x < y + \mathfrak{b}_0; \end{cases}$$

$$\mathfrak{b}_1 = \begin{cases} 0 & \text{if } x \geq y + \mathfrak{b}_0, \\ 1 & \text{if } x < y + \mathfrak{b}_0. \end{cases}$$

The assembly instruction sub can be used to implement $\mathsf{sub}(x, y, 0)$ while the assembly instruction sbb can be used to implement the more general $\mathsf{sub}(x, y, \mathfrak{b}_0)$.

Algorithm 2 Subtraction in $\mathbb{F}_{2^{448}-2^{224}-1}$.

1: **function** sub448($(f(\theta), g(\theta))$)
2: **input:** 7-limb quantities $f(\theta)$ and $g(\theta)$ such that $0 \leq f_i, g_j < 2^{64}$ for $i, j = 0, 1, \ldots, 6$.
3: **output:** $h^{(2)}(\theta) = h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_1^{(2)}\theta^6$ such that $0 \leq h_i^{(2)} < 2^{64}$ for $i = 0, 1, \ldots, 6$ and $h^{(2)}(\theta) \equiv (f(\theta) - g(\theta)) \bmod p$.
4: $\mathfrak{b} \leftarrow 0$
5: **for** $i \leftarrow 0$ to 6 **do** $(h_i^{(0)}, \mathfrak{b}) \leftarrow \mathsf{sub}(f_i, g_i, \mathfrak{b})$ **end for**
6: $\mathfrak{d} \leftarrow \mathfrak{b}$; $t \leftarrow \mathfrak{d} \ll 32$
7: $\mathfrak{b} \leftarrow 0$
8: $(h_0^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_0^{(0)}, \mathfrak{d}, \mathfrak{b})$
9: $(h_1^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_1^{(0)}, 0, \mathfrak{b})$
10: $(h_2^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_2^{(0)}, 0, \mathfrak{b})$
11: $(h_3^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_3^{(0)}, t, \mathfrak{b})$
12: $(h_4^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_4^{(0)}, 0, \mathfrak{b})$
13: $(h_5^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_5^{(0)}, 0, \mathfrak{b})$
14: $(h_6^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_6^{(0)}, 0, \mathfrak{b})$
15: $\mathfrak{d} \leftarrow \mathfrak{b}$; $t \leftarrow \mathfrak{d} \ll 32$
16: $\mathfrak{b} \leftarrow 0$
17: $(h_0^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_0^{(1)}, \mathfrak{d}, \mathfrak{b})$
18: $(h_1^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_1^{(1)}, 0, \mathfrak{b})$
19: $(h_2^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_2^{(1)}, 0, \mathfrak{b})$
20: $(h_3^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_3^{(1)}, t, \mathfrak{b})$
21: $(h_4^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_4^{(1)}, 0, \mathfrak{b})$
22: $h_5^{(2)} \leftarrow h_5^{(1)}$; $h_6^{(2)} \leftarrow h_6^{(1)}$
23: **return** $h^{(2)}(\theta) = h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6$
24: **end function**.

The correctness of sub448 is stated in the following theorem.

Theorem 2. *The output* $h^{(2)}(\theta) = h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6$ *of* sub448 *satisfies* $0 \leq h_i^{(2)} < 2^{64}$ *for* $i = 0, 1, \ldots, 6$ *and* $h^{(2)}(\theta) \equiv (f(\theta) - g(\theta)) \bmod p$.

Proof. The limbs $h_i^{(2)}$, $i = 0, 1, \ldots, 6$ are obtained as the first components of the outputs of some invocations of the sub instruction. Consequently, it follows that all of these are 64-bit quantities. This settles the point about the bounds on these limbs. So, we have to argue two things. First, $h^{(2)}(\theta) = (f(\theta) - g(\theta)) \bmod p$ and second that the procedure terminates without any overflow. The congruency argument is obtained from the following observations.

1. Let $\delta = 2^{224} + 1$. The subtraction of $\mathfrak{d}$ from $h_0^{(0)}$ in Step 8 and of t from $h_3^{(0)}$ in Step 11 correspond to the subtraction of δ from the integer represented by $h^{(0)}(\theta)$. Similarly, the subtraction of $\mathfrak{d}$ from $h_1^{(0)}$ in Step 17 and of t from $h_3^{(1)}$ in Step 20 correspond to the subtraction of δ from the integer represented by $h^{(1)}(\theta)$.

2. Suppose $f(\theta) \geq g(\theta)$ (as integers). Then after Step 5, we have $h^{(0)}(\theta) = f(\theta) - g(\theta)$ and $\mathfrak{b} = 0$. As a consequence of $\mathfrak{b} = 0$ at Step 5, it follows that $h^{(0)}(\theta) = h^{(1)}(\theta) = h^{(2)}(\theta)$ establishing the result for this particular case.

3. In view of the previous point, assume $f(\theta) < g(\theta)$. In this case, after Step 5, we have that $h^{(0)}$ represents the integer $2^{448} + f(\theta) - g(\theta)$ and $\mathfrak{b} = 1$. Steps 8-14 subtract δ from $h^{(0)}(\theta) = 2^{448} + f(\theta) - g(\theta)$.

   (a) If $h^{(0)}(\theta) \geq \delta$, then after Step 14, $h^{(1)}(\theta)$ represents the integer $h^{(0)}(\theta) - \delta = 2^{448} + f(\theta) - g(\theta) - \delta = p + f(\theta) - g(\theta) \equiv (f(\theta) - g(\theta)) \bmod p$ and $\mathfrak{b} = 0$. As a consequence of $\mathfrak{b} = 0$ at Step 14, it follows that $h^{(2)}(\theta) = h^{(1)}(\theta)$ establishing the result for this case.

   (b) If $h^{(0)}(\theta) < \delta$, then after Step 14, $h^{(1)}(\theta)$ represents the integer $2^{448} + h^{(0)}(\theta) - \delta = 2^{448} + 2^{448} + f(\theta) - g(\theta) - \delta = 2^{448} + p + f(\theta) - g(\theta)$ and $\mathfrak{b} = 1$. Steps 17-21 subtract δ from $h^{(1)}(\theta) = 2^{448} + p + f(\theta) - g(\theta)$ to obtain $h^{(2)}(\theta) = h^{(1)}(\theta) - \delta = 2^{448} + p + f(\theta) - g(\theta) - \delta = 2p + f(\theta) - g(\theta) \equiv (f(\theta) - g(\theta)) \bmod p$.

It only remains to argue that $\mathfrak{b}$ produced by the sub instruction in Step 21 is necessarily 0. If the value of $\mathfrak{b}$ in the input of sub in Step 21 is 0, then of course, the value of $\mathfrak{b}$ produced by this sub call is also 0. So, suppose that the value of $\mathfrak{b}$ in the input of sub in Step 21 is 1. Then the value of $\mathfrak{b}$ produced by this sub call is 0 if and only if $h_4^{(1)} \geq 1$. The value of $\mathfrak{b}$ in the input of sub in Step 21 is 1 only if the value of $\mathfrak{b}$ produced by the sub call in Step 14 is 1. Arguing backwards, the value of $\mathfrak{b}$ produced by the sub call in Step 12 must be 1. The input to the sub call in Step 12 is $(h_4^{(0)}, 0, \mathfrak{b})$ and so the value of $\mathfrak{b}$ produced by this sub call is 1 if and only if $h_4^{(0)} = 0$ and the value of $\mathfrak{b}$ in the input to this sub call is 1. Then, it follows that $h_4^{(1)} = 2^{64} - 1 \geq 1$ as required. □

4 Conclusion

In this work we have provided explicit description of reduction algorithms required for computation in the field $\mathbb{F}_p$ with $p = 2^{448} - 2^{224} - 1$. The correctness of the algorithms have been rigorously proven. We highlight the importance of having correctness proofs by pointing out in Appendix A that the reduction algorithms used in the previous fastest known code are incomplete.

References

[1] Mike Hamburg. Ed448-goldilocks, a new elliptic curve. *IACR Cryptology ePrint Archive*, 2015:625, 2015.

[2] Adam Langley and Mike Hamburg. Elliptic curves for security. Internet Research Task Force (IRTF), Request for Comments: 7748, https://tools.ietf.org/html/rfc7748, 2016. Accessed on 16 September, 2019.

[3] Kaushik Nath and Palash Sarkar. Efficient Arithmetic in (Pseudo-)Mersenne Prime Order Fields. *IACR Cryptology ePrint Archive*, 2018:985, 2018.

[4] Thomaz Oliveira, Julio López Hernandez, Hüseyin Hisil, Armando Faz-Hernández, and Francisco Rodríguez-Henríquez. How to (pre-)compute a ladder - improving the performance of X25519 and X448. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 172–191. Springer, 2017.

[5] E. Ozturk, J. Guilford, and V. Gopal. Large integer squaring on Intel architecture processors, intel white paper. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/large-integer-squaring-ia-paper.pdf, 2013.

[6] E. Ozturk, J. Guilford, V. Gopal, and W. Feghali. New instructions supporting large integer arithmetic on Intel architecture processors, intel white paper. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-large-integer-arithmetic-paper.pdf, 2012.

[7] Version 1.3 TLS Protocol. RFC 8446. https://datatracker.ietf.org/doc/rfc8446/?include_text=1, 2018. Accessed on 16 September, 2019.

A Incompleteness of the Reduction Algorithms Used in the Software Implementation Accompanying [4]

The work [4] provides performance results of Montgomery ladder computation of Curve448. Explicit reduction algorithms required for implementing arithmetic in $\mathbb{F}_p$ are not described in the paper. We have examined the accompanying code[1] and have been able to obtain the reduction algorithms that have been actually used in the code. For the convenience of description, in the following, we will refer to [4] for the reduction algorithms.

Below we mention the reduction algorithms used in [4] and explain why they are incomplete.

Case of 14-limb to 7-limb reduction: As shown in `reduce448` and proved in Theorem 1, the correct output is $h^{(4)}(\theta)$. The work [4], however, provides $h^{(3)}(\theta)$ as the output. From the proof of Theorem 1 one may observe that it is not possible to argue that $h^{(3)}(\theta)$ will have no overflows.

Case of 8-limb to 7-limb reduction: As mentioned in Section 3.2, the 8-limb quantity is to be considered $h^{(2)}(\theta)$ and the relevant portion of `reduce448` applied to $h^{(2)}(\theta)$ to obtain the correct result as $h^{(4)}(\theta)$. The work [4], however, performs the reduction of $h^{(2)}(\theta)$ to $h^{(3)}(\theta)$ only up to Step 30 of `reduce448` (i.e., up to the computation of $h_4^{(3)}$) and provides this intermediate quantity as the output. Moreover, the code for addition does not compute $h_2^{(3)}$ of Step 28 considering the carry-out of the previous step. Since the computation of $h_2^{(3)}$ has not been done using an `adc`, the carry-out of this add-with-carry step is also not taken care of while computing $h_3^{(3)}$ in Step 29. Similar discrepancies are also there in the implementation of reduction after multiplying with a small constant.

Case of subtraction: The work [4] performs the steps of `sub448` up to Step 12 (i.e., up to the computation of $h_4^{(1)}$) and provides this intermediate quantity as the output. Similar to addition, the reduction code for subtraction does not compute $h_2^{(1)}$ of Step 10 considering the bit $\mathfrak{b}$ of the previous step. Since the computation of $h_2^{(1)}$ has not been done using an `sbb`, the bit $\mathfrak{b}$ of this step is also not taken care of while computing $h_3^{(1)}$ in Step 11.

While the proofs of correctness of the reduction algorithms that we describe show the incompleteness of the reduction computations done in [4], the steps that have been ignored in [4] affect the final result with very low probability. We highlight two examples.

1. The 8-limb to 7-limb reduction in [4] stops the computation at Step 30 of `reduce448`. The possibility that in the first instruction of Step 30, $h_4^{(2)} = 2^{64}-1$ and `carry` $= 1$ is ignored. Heuristically assuming $h_4^{(2)}$ to be a uniformly distributed 64-bit quantity, the probability that it equals $2^{64}-1$ is 2^{-64} which is a low probability event.

2. Similarly, for subtraction, the procedure in [4] stops at Step 12 of `sub448`. In effect, this ignores the possibility that $h_4^{(0)} = 0$ and the value of $\mathfrak{b}$ provided as input to `sub` in Step 12 is 1. Again, heuristically assuming $h_4^{(0)}$ to be uniformly distributed, the probability that $h_4^{(0)} = 0$ is 2^{-64}.

Consequently, the incorrect results arising out of the incomplete reductions in [4] are very hard to capture using some randomly generated known answer tests (KATs). Nevertheless, in certain cases it is possible to work out examples on which the reductions in [4] lead to incorrect results. A brief case study of the software implementation of addition given in [4] is provided below.

Case study of addition. We reproduce below the code of addition modulo $2^{448}-2^{224}-1$ from [4]. In the code, the `mov` instructions are used for the load/store operations. The `add` instructions in pink add the 7 limbs of the two polynomials, the instructions in red captures the carry, and the instructions in blue perform the reduction.

Let us consider two input polynomials $f(\theta) = f_0 + f_1\theta + \cdots + f_6\theta^6$ and $g(\theta) = g_0 + g_1\theta + \cdots + g_6\theta^6$ such that

$$f_0 = f_1 = f_2 = f_4 = f_5 = 2^{64}-3, f_3 = 2^{64}-2^{32}-1, f_6 = 2^{64}-1$$

and

$$g_0 = g_1 = g_2 = g_4 = g_5 = 2, g_3 = 2^{32}, g_6 = 1$$

[1] Program code from https://github.com/armfazh/rfc7748_precomputed/blob/master/src/fp448_x64.c was accessed on November 1, 2019.

. The polynomial after adding f and g is $h^{(0)}(\theta) = h_0^{(0)} + h_1^{(0)}\theta + \cdots + h_6^{(0)}\theta^6 + h_7^{(0)}\theta^7$, where

$$h_0^{(0)} = h_1^{(0)} = h_2^{(0)} = h_3^{(0)} = h_4^{(0)} = h_5^{(0)} = 2^{64} - 1, h_6^{(0)} = 0, h_7^{(0)} = 1.$$

Here, $h_7^{(0)}$ holds the carry-out which is captured by the register `rbx` in the code below. Now, after the performing the reduction steps on $h^{(0)}(\theta)$, the final output produced by the code is $h^{(1)}(\theta) = h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6$, where

$$h_0^{(1)} = h_1^{(1)} = h_4^{(1)} = 0, h_3^{(1)} = 2^{32} - 1, h_2^{(1)} = h_5^{(1)} = 2^{64} - 1, h_6^{(1)} = 0.$$

The code is not taking care of the carry-out produced from the additions occurring in the second and fourth limbs making the reduction incomplete. The correct reduced output should be the polynomial $h^{(1)}(\theta) = h_0^{(1)} + h_1^{(1)}\theta + \cdots + h_6^{(1)}\theta^6$, where

$$h_0^{(1)} = h_1^{(1)} = h_2^{(1)} = h_4^{(1)} = h_5^{(1)} = 0, h_3^{(1)} = 2^{32}, h_6^{(1)} = 1.$$

```
inline void add_EltFp448_1w_x64(uint64_t *c, uint64_t *a, uint64_t *b) {
#if __ADX__
  __asm__ __volatile__(
    "movq    (%2),  %%rax ;"
    "movq   8(%2),  %%rcx ;"
    "movq  16(%2),  %%rdx ;"
    "movq  24(%2),  %%r8  ;"
    "movq  32(%2),  %%r9  ;"
    "movq  40(%2),  %%r10 ;"
    "movq  48(%2),  %%r11 ;"
    "clc ;"
    "adcx    (%1),  %%rax ;"
    "adcx   8(%1),  %%rcx ;"
    "adcx  16(%1),  %%rdx ;"
    "adcx  24(%1),  %%r8  ;"
    "adcx  32(%1),  %%r9  ;"
    "adcx  40(%1),  %%r10 ;"
    "adcx  48(%1),  %%r11 ;"
    "setc    %%bl         ;"
    "movzx   %%bl,  %%rbx ;"
    "addq   %%rbx,  %%rax ;"
    "adcq   $0,     %%rcx ;"
    "shlq   $32,    %%rbx ;"
    "addq   %%rbx,  %%r8  ;"
    "adcq   $0,     %%r9  ;"
    "movq   %%rax,   (%0) ;"
    "movq   %%rcx,  8(%0) ;"
    "movq   %%rdx, 16(%0) ;"
    "movq   %%r8 , 24(%0) ;"
    "movq   %%r9 , 32(%0) ;"
    "movq   %%r10, 40(%0) ;"
    "movq   %%r11, 48(%0) ;"
  :
  : "r" (c), "r" (a), "r"  (b)
  : "memory", "cc", "%rax", "%rbx", "%rcx",
    "%rdx", "%r8", "%r9", "%r10", "%r11"
  );
#else
  __asm__ __volatile__(
  "movq    (%2),  %%rax ;"
  "movq   8(%2),  %%rcx ;"
  "movq  16(%2),  %%rdx ;"
  "movq  24(%2),  %%r8  ;"
  "movq  32(%2),  %%r9  ;"
  "movq  40(%2),  %%r10 ;"
  "movq  48(%2),  %%r11 ;"
  "addq    (%1),  %%rax ;"
  "adcq   8(%1),  %%rcx ;"
  "adcq  16(%1),  %%rdx ;"
  "adcq  24(%1),  %%r8  ;"
  "adcq  32(%1),  %%r9  ;"
  "adcq  40(%1),  %%r10 ;"
  "adcq  48(%1),  %%r11 ;"
  "setc    %%bl         ;"
  "movzx   %%bl,  %%rbx ;"
  "addq   %%rbx,  %%rax ;"
  "adcq   $0,     %%rcx ;"
  "shlq   $32,    %%rbx ;"
  "addq   %%rbx,  %%r8  ;"
  "adcq   $0,     %%r9  ;"
  "movq   %%rax,   (%0) ;"
  "movq   %%rcx,  8(%0) ;"
  "movq   %%rdx, 16(%0) ;"
  "movq   %%r8 , 24(%0) ;"
  "movq   %%r9 , 32(%0) ;"
  "movq   %%r10, 40(%0) ;"
  "movq   %%r11, 48(%0) ;"
  :
  : "r" (c), "r" (a), "r"  (b)
  : "memory", "cc", "%rax", "%rbx", "%rcx",
   "%rdx", "%r8", "%r9", "%r10", "%r11"
  );
#endif
}
```