A Note on "Reduction Modulo $2^{448} - 2^{224} - 1$"

Timothy Shelton
University of Kent
tjbs2@kent.ac.uk

Abstract

Nath and Sarkar propose algorithms to improve the efficiency of Diffie-Hellman key agreement using Curve448. In this note an error in the proof of correctness of the subtraction algorithm is described. An alternative argument is offered to fix this error without changing the algorithm or statement of correctness.

Introduction

Transport Layer Security (TLS) protocol version 1.3, RFC 8446, includes Curve448 in the list of supported groups for key exchange [3, p. 46]. Curve448 is an elliptic curve with underlying field $\mathbb{F}_p$ where $p = 2^{448} - 2^{224} - 1$. Therefore, to implement cryptographic algorithms using Curve448, efficient arithmetic modulo $2^{448} - 2^{224} - 1$ is required.

In [1], Nath and Sarkar propose algorithms for reduction and subtraction modulo $2^{448} - 2^{224} - 1$ that improve the speed of X448 shared secret and key generation operations compared to the work [2]. [1, Theorem 1] and [1, Theorem 2] state correctness of the algorithms for reduction and subtraction respectively and proofs are provided.

A review of the proof of Theorem 2 revealed a potential error in the argument that Algorithm 2 will terminate without overflow. This note identifies the error and offers an alternative argument to show [1, Algorithm 2] will terminate correctly.

Algorithm

Algorithm 2 and the definition of the sub instruction provided in [1] are reproduced here.

Definition. The instruction sub is defined as follows.

$$(z, \mathfrak{b}_{out}) \leftarrow \mathsf{sub}(x, y, \mathfrak{b}_{in}) \tag{1}$$

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

$$\mathfrak{b}_{out} = \begin{cases} 0 & \text{if } x \geq y + \mathfrak{b}_{in}, \\ 1 & \text{if } x < y + \mathfrak{b}_{in}; \end{cases} \tag{3}$$

Algorithm 2: Subtraction in $\mathbb{F}_p$

1 **function** $\mathsf{sub448}(f(\theta), g(\theta))$
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:** $h^{(2)}(\theta) = h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6$ such that $0 \leq h_i^{(2)} < 2^{64}$ for $i = 0, 1, \ldots, 6$ and $h^{(2)}(\theta) \equiv (f(\theta) - g(\theta)) \mod p$.

4 $\quad \mathfrak{b} \leftarrow 0$
5 $\quad$ **for** $i \leftarrow 0$ *to* 6 **do**
6 $\quad\quad (h_i^{(0)}, \mathfrak{b}) \leftarrow \mathsf{sub}(f_i, g_i, \mathfrak{b})$
7 $\quad$ **end for**

8 $\quad \mathfrak{d} \leftarrow \mathfrak{b}; \mathfrak{d}' \leftarrow \mathfrak{b} \ll 32$
9 $\quad \mathfrak{b} \leftarrow 0$
10 $\quad (h_0^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_0^{(0)}, \mathfrak{d}, \mathfrak{b})$
11 $\quad (h_1^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_1^{(0)}, 0, \mathfrak{b})$
12 $\quad (h_2^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_2^{(0)}, 0, \mathfrak{b})$
13 $\quad (h_3^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_3^{(0)}, \mathfrak{d}', \mathfrak{b})$
14 $\quad (h_4^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_4^{(0)}, 0, \mathfrak{b})$
15 $\quad (h_5^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_5^{(0)}, 0, \mathfrak{b})$
16 $\quad (h_6^{(1)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_6^{(0)}, 0, \mathfrak{b})$

17 $\quad \mathfrak{d} \leftarrow \mathfrak{b}; \mathfrak{d}' \leftarrow \mathfrak{b} \ll 32$
18 $\quad \mathfrak{b} \leftarrow 0$
19 $\quad (h_0^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_0^{(1)}, \mathfrak{d}, \mathfrak{b})$
20 $\quad (h_1^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_1^{(1)}, 0, \mathfrak{b})$
21 $\quad (h_2^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_2^{(1)}, 0, \mathfrak{b})$
22 $\quad (h_3^{(2)}, \mathfrak{b}) \leftarrow \mathsf{sub}(h_3^{(1)}, \mathfrak{d}', \mathfrak{b})$
23 $\quad h_4^{(2)} \leftarrow h_4^{(1)}; h_5^{(2)} \leftarrow h_5^{(1)}; h_6^{(2)} \leftarrow h_6^{(1)}$

24 $\quad$ **return** $h^{(2)}(\theta) = h_0^{(2)} + h_1^{(2)}\theta + \cdots + h_6^{(2)}\theta^6$
25 **end function**

The Errors

In the proof of Theorem 2 of [1] it is correctly argued that [1, Algorithm 2] outputs the correct result without overflow in cases 2 and 3(a). However, there is an error in the argument that Algorithm 2 will not overflow at Step 22.

The second sentence of the last paragraph states: *"If the value of $\mathfrak{b}$ in the input of sub in Step 22 is 0, then of course, the value of $\mathfrak{b}$ produced by this sub call is also 0."* This can be written as

$$\mathfrak{b}_{in} = 0 \Rightarrow \mathfrak{b}_{out} = 0. \tag{4}$$

This is equivalent to

$$\mathfrak{b}_{in} \neq 1 \Rightarrow \mathfrak{b}_{out} \neq 1. \tag{5}$$

The contrapositive of 5 is

$$\mathfrak{b}_{out} = 1 \Rightarrow \mathfrak{b}_{in} = 1 \tag{6}$$

which is true if and only if $h_3^{(1)} = \mathfrak{d}'$.

Case 3(b) is when $f(\theta) < g(\theta)$ and $h^{(0)}(\theta) < \delta$. Since $f(\theta) < g(\theta)$, overflow occurs in the final iteration of the loop in Steps 5–7. Overflow also occurs in Step 16 because $h^{(0)}(\theta) < \delta$ and $\mathfrak{b} = 1$ in Step 8 from the previous overflow. From this we know $\mathfrak{d}' = 2^{32}$ in Step 22. Therefore we can write Step 22 as

$$(h_3^{(2)}, \mathfrak{b}_{out}) \leftarrow \mathsf{sub}(h_3^{(1)}, 2^{32}, \mathfrak{b}_{in}). \tag{7}$$

From the definition of sub we know that in Step 22, $\mathfrak{b}_{out} = 1$ if and only if $h_3^{(1)} < 2^{32} + \mathfrak{b}_{in}$. Therefore implication 6 holds exactly when $h_3^{(1)} = \mathfrak{d}' = 2^{32}$ as otherwise the inequality may still hold with $\mathfrak{b}_{in} = 0$. However, this equality does not hold for any inputs $f(\theta)$ and $g(\theta)$ since, as part of the suggested changes, we will show $h_3^{(1)} \geq 2^{32} + 1$.

A similar statement is made near the end of the paragraph about the sub call in Step 13; *"the value of $\mathfrak{b}$ produced by this sub call is 1 if and only if the value of $\mathfrak{b}$ in the input to this sub call is 1 and $0 \leq h_3^{(0)} < 2^{32} + 1$"*. Using mathematical logic notation,

$$\mathfrak{b}_{out} = 1 \iff ((\mathfrak{b}_{in} = 1) \wedge (0 \leq h_3^{(0)} < 2^{32} + 1)). \tag{8}$$

The right-to-left implication of 8 holds by definition of sub and the value of $\mathfrak{d}'$ in case 3(b). However, the left-to-right implication fails in general since $h_3^{(0)} < 2^{32}$ will always produce $\mathfrak{b}_{out} = 1$ regardless of the value of $\mathfrak{b}_{in}$.

The argument that Algorithm 2 will not overflow at Step 22 seems like it is a general statement to argue that overflow cannot occur regardless of which case the inputs fall into. This is unnecessary as the arguments presented for cases 2 and 3(a) already establish the full result for those cases. Thus we can consider case 3(b) only in the final argument which simplifies it.

Suggested changes: As argued in the original proof by the authors, if $h_3^{(1)} \geq 2^{32} + 1$ then there will be no overflow in step 22. This inequality holds for case 3(b). The details of this argument follow.

Proof of Theorem 2 Case 3(b). From the definition of $\mathsf{sub}(h_3^{(1)}, \mathfrak{d}', \mathfrak{b}_{in})$ we know $\mathfrak{b}_{out} = 1$ if and only if $h_3^{(1)} < \mathfrak{d}' + \mathfrak{b}_{in}$ with $\mathfrak{d}' \in \{0, 2^{32}\}$ and $\mathfrak{b}_{in} \in \{0, 1\}$. Therefore it is sufficient to show $h_3^{(1)} \geq 2^{32} + 1$. Since $f(\theta) < g(\theta)$ in case 3(b), we have $\mathfrak{b} = 1$ in Step 8 and consequently $\mathfrak{d}' = 2^{32}$ in Step 13. Therefore, by the definition of sub and the values of the computation at Step 13, we have

$$\begin{aligned} h_3^{(1)} &= 2^{64} + h_3^{(0)} - (2^{32} + \mathfrak{b}) \\ &\geq 2^{64} - (2^{32} + \mathfrak{b}) \\ &\geq 2^{32} + 1 \end{aligned}$$

as required. $\square$

References

[1] Kaushik Nath and Palash Sarkar. Reduction modulo $2^{448} - 2^{224} - 1$. *Mathematical Cryptology*, (1):8–21, Jan. 2021. URL `https://journals.flvc.org/mathcryptology/article/view/123700`.

[2] Thomaz Oliveira, Julio López, Hüseyin Hışıl, Armando Faz-Hernández, and Francisco Rodríguez-Henríquez. How to (pre-)compute a ladder. In *Selected Areas in Cryptography – SAC 2017*, pages 172–191. Springer International Publishing, December 2017. doi: 10.1007/978-3-319-72565-9_9. URL `https://doi.org/10.1007/978-3-319-72565-9_9`.

[3] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL `https://rfc-editor.org/rfc/rfc8446.txt`.