A Speed-Area-Efficient Hardware ECPM-Engine in GF(p) over Generic Weierstrass Curves

Yujun Xie[1], Zhenhui He[2], Yuan Liu[1], Xin Zheng[2], Shuting Cai[2], and xiaoming xiong[2]

[1]Guangdong University of Technology - University Town Campus
[2]Guangdong University of Technology

November 16, 2023

Abstract

This paper proposes a 256-bit speed-area-efficient hardware elliptic curve point-multiplication engine (ECPM-engine) in GF(p) over generic Weierstrass curves, which is optimized by a new speed-area-efficient radix-64 Montgomery modular multiplication (R64MMM) and a novel Montgomery ladder scheduling. The R64MMM calls one 129-bit adder and one (64x64+129)-bit multiply-accumulator (64-129-MAC) in parallel to make a trade-off between speed and area. The novel Montgomery ladder scheduling is used to improve the utilization of MAC in ECPM operation. In this ECPM-engine, both MAC utilization in R64MMM operations and R64MMM utilization in ECPM operations are close to 100%. The result shows that the proposed ECPM-engine consumes 72k gates when the clock frequency is 714 MHz with a 90 nm standard cell library, and it computes one 256-bit ECPM in 0.14 ms.

A Speed-Area-Efficient Hardware ECPM-Engine in GF(p) over Generic Weierstrass Curves

Yujun Xie, Zhenhui He, Yuan Liu*, Xin Zheng*, Shuting Cai, and Xiaoming Xiong

This paper proposes a 256-bit speed-area-efficient hardware elliptic curve point-multiplication engine (ECPM-engine) in GF(p) over generic Weierstrass curves, which is optimized by a new speed-area-efficient radix-64 Montgomery modular multiplication (R64MMM) and a novel Montgomery ladder scheduling. The R64MMM calls one 129-bit adder and one (64x64+129)-bit multiply-accumulator (64-129-MAC) in parallel to make a trade-off between speed and area. The novel Montgomery ladder scheduling is used to improve the utilization of MAC in ECPM operation. In this ECPM-engine, both MAC utilization in R64MMM operations and R64MMM utilization in ECPM operations are close to 100%. The result shows that the proposed ECPM-engine consumes 72k gates when the clock frequency is 714 MHz with a 90 nm standard cell library, and it computes one 256-bit ECPM in 0.14 ms.

Introduction: Elliptic curve cryptography (ECC) [1] is an effective method to prevent data leakage, and Elliptic curve point-multiplication (ECPM) is the most critical operation of the ECC. The dedicated hardware accelerators can greatly improve the computational performance of ECPM. However, large hardware resource consumption is not benefit for lightweight IoT devices. It is a popular ECC research direction recently by improving the hardware performance of ECPM using speed-area-efficient hardware accelerators.

Many hardware accelerators of the ECPM are proposed [2-3, 6-13]. Hu [2] and Choi [3] propose two hardware accelerators, which use specific primes and curve domain parameters [4-5] to improve the computational performance of ECPM. However, their methods do not support generic Weierstrass curves. For the designs that support security at different levels, some researchers focus on lightweight implementation [6-10], developing ECPM processors to accelerate the ECPM operation. A scalable dual-field ECC processor is designed by Satoh [6], which calls one 64-bit multiplier to calculate the ECPM. Chen [7] presents a ECC processor, which adopts a systolic arithmetic. An optimized resource sharing mechanism in combined point doubling and point addition hardware architecture is proposed by Kudithi [8] to reduce hardware resources. Yeh [9] explores the hybrid modular arithmetic architecture to reduce both area and energy costs of ECC processor. A flexible dual-field ECC processor using the hardware-software approach is presented by Liu [10]. The ECPM processors aim at high-speed are proposed by chung [11], Hossain [12] , and Xie [13]. Their processors [11-13] consume too much hardware resources for lightweight applications.

This paper designs a speed-area-efficient hardware ECPM engine to accelerate the ECPM operation. To make a trade-off between speed and area, we propose a new speed-area-efficient radix-64 Montgomery modular multiplication (R64MMM) and a novel Montgomery ladder scheduling to implement a 256-bit speed-area-efficient hardware ECPM-engine in GF(p) over generic Weierstrass curves.

Preliminary: The coordinates (x, y) in GF(p) over Generic Weierstrass Curves satisfies the equation E [14]:

$$E : y^2 = x^3 + ax^2 + b \bmod p \tag{1}$$

Where $(4a^3 + 27b^2) \bmod p \neq 0$.

Let $P = (x_P, y_P) \in E$ and $Q = (x_Q, y_Q) \in E$. The ECPM can be expressed by:

$$\mathrm{ECPM} : Q = [k]P = \underbrace{P + P + \cdots + P}_{k \text{ times}} \tag{2}$$

The Montgomery ladder [15] is a general method to calculate the ECPM. It can resistant against simple power attack (SPA). So far, the Montgomery ladder formula proposed by Hamburg [16] is considered the the state-of-the-art method in GF(p) for the generic Weierstrass curve.

Before the calculation of this Montgomery ladder method [14], $P = (x_P, y_P)$ needs to be mapped to the Hamburg's ladder state (SETUP):

$$\begin{aligned} X_{QP} &= 0 \\ X_{RP} &= (3x_P^2 + a)^2 - 2x_P(2y_P)^2 \\ Y_Q &= (2y_P)^4 \\ Y_R &= (3x_P^2 + a)X_{RP} + (2y_P)^4 \\ G &= X_{RP}^2 \end{aligned} \tag{3}$$

where X_{QP}, X_{RP}, Y_Q, Y_R, and G are five values in Jacobian coordinates. They are the input of the Hamburg's Montgomery ladder.

The calculation of this Montgomery ladder is shown in algorithm 1 [16]. The values of X_{QP}, X_{RP}, Y_Q, Y_R, and G are updated in each Montgomery ladder operation, and the M is used to restore the final result of the Hamburg's Montgomery ladder.

Algorithm 1 Montgomery Ladder algorithm (LADDER)

Require: $(X_{QP}, X_{RP}, Y_Q, Y_R, G)$
Ensure: $(X_{QP}, X_{RP}, Y_Q, Y_R, G, M)$
1: $X'_{QP} = X_{QP} \cdot G$
2: $X'_{RP} = X_{RP} \cdot G$
3: $L = Y_Q \cdot Y_R$
4: $H = Y_R \cdot Y_R$
5: $J = X'_{RP} - L$
6: $M = J + X'_{RP} - H$
7: $X_{SP} = H \cdot L$
8: $K = J^2$
9: $X_{TP} = X'_{RP} \cdot J + X'_{QP} \cdot H$
10: $X_{TS} = (X_{TP} - X_{SP}$
11: $Y_S = (X_{TS} - K) \cdot H$
12: $Y_T = M \cdot X_{TS} + Y_S$
13: $G' = X_{TS}^2$
14: $(X_{QP}, X_{RP}, Y_Q, Y_R, G) = (X_{SP}, X_{TP}, Y_S, Y_T, G')$

After the calculation of Montgomery ladder algorithm, X_{QP}, X_{RP}, Y_Q, Y_R, G, and M need to exit the Hamburg's ladder state and restore to $Q = (x_Q, y_Q)$ (EXIT).

$$\begin{aligned} x_Q &= \frac{X_{QP}}{Z^2} + x_P \\ y_Q &= \frac{Y_Q}{2Z^3} \end{aligned} \tag{4}$$

Where $\frac{1}{Z} = \frac{2y_P((M/2)^2 - X_{QP} - X_{RP})}{3x_P Y_P}$ and $Y_P = Y_R - MX_{RP}$.

The ECPM claculation is shown in algorithm 2 [16].

Algorithm 2 Montgomery ladder ECPM

Require: $k, p \leq 2^n, p, P = (x_P, y_P)$
Ensure: $Q = (x_Q, y_Q) = [k]P$
1: $k = 2^n + (k - 2^n \bmod p)$
2: $(X_{QP}, X_{RP}, Y_Q, Y_R, G) = \mathrm{SETUP}(x_P, y_P)$
3: **for** $i = n - 1$ **to** 0 **do**
4: **if** $k_i == 1$ **then**
5: $(X_{QP}, X_{RP}, Y_Q, Y_R, G) = \mathrm{LADDER}(X_{QP}, X_{RP}, Y_Q, Y_R, G)$
6: **else**
7: $(X_{RP}, X_{QP}, Y_R, Y_Q, G) = \mathrm{LADDER}(X_{RP}, X_{QP}, Y_R, Y_Q, G)$
8: **end if**
9: **end for**
10: $(x_Q, y_Q) = \mathrm{EXIT}(X_{QP}, X_{RP}, Y_Q, Y_R, G, M)$

A New Speed-Area-Efficient R64MMM: The ECPM operation involves three steps: 1) SETUP, 2) LADDER, and 3) EXIT. These three steps need to call modular operations (modular multiplication (MM), addition and subtraction, and inversion) to calculate. MM consumes the most of the computation time of ECPM and it is the most critical operation in ECPM.

In order to improve the performance of the modular multiplication (MM), Montgomery modular multiplication (MMM) [16] uses shift operation instead of the division and subtraction operations of MM. The proposed radix-64 Montgomery modular multiplication (R64MMM) call one 64-129-MAC and 129-Adder in parallel. Algorithm 3 shows the

process of R64MMM without the reduction step, which can be calculated in 37 clock cycles. The calculation $(z_3, z_2) = \text{reg1} + \text{reg2}$ of $CN = 37$ is computed in the first clock cycle of the next R64MMM.

Algorithm 3 Radix-64 Montgomery multiplication (R64MMM)

Input: $x = \sum_{i=0}^{3} x^{(i)} 2^{\omega i}$, $y = \sum_{j=0}^{3} y^{(j)} 2^{\omega j}$, $m = \sum_{j=0}^{3} m^{(j)} 2^{\omega j}$
$\omega = 64$, $m' = (-m^0)^{-1} \bmod 2^{\omega j}$

Output: $s = (z_3, z_2, z_1, z_0) = x \cdot y \cdot 2^{-256}$ $(2m > s > 0)$

CN	64-129-MAC	129-Adder
	Round1: $i = 0$	
1	$\text{reg1} = x_i \cdot y_0 + z_0$	
2	$q = (\text{reg1}[63:0] \cdot m' + 0) \bmod 2^{\omega}$	
3	$(c, z_0) = (q \cdot m_0 + \text{reg1})$	
4	$\text{reg1} = (x_i \cdot y_1 + z_1)$	
5	$\text{reg2} = (q \cdot m_1 + c)$	
6	$\text{reg1} = (x_i \cdot y_2 + z_2)$	$(c, z_0) = \text{reg1} + \text{reg2}$
7	$\text{reg2} = (q \cdot m_2 + c)$	
8	$\text{reg1} = (x_i \cdot y_3 + z_3)$	$(c, z_1) = \text{reg1} + \text{reg2}$
9	$\text{reg2} = (q \cdot m_3 + c)$	
	Round2: $i = 1$	
10	$\text{reg1} = x_i \cdot y_0 + z_0$	$i = 0$: $(z_3, z_2) = \text{reg1} + \text{reg2}$
11~18	Round2 64-129-MAC operation	Round2 129-Addition
	Round3: $i = 2$	
19	$\text{reg1} = x_i \cdot y_0 + z_0$	$i = 1$: $(z_3, z_2) = \text{reg1} + \text{reg2}$
20~27	Round3 64-129-MAC operation	Round3 129-Addition
	Round4: $i = 3$	
28	$\text{reg1} = x_i \cdot y_0 + z_0$	$i = 2$: $(z_3, z_2) = \text{reg1} + \text{reg2}$
29~36	Round4 64-129-MAC operation	Round4 129-Addition
	Next R64MMM Round1: $i = 0$	
37	$\text{reg1} = x_i \cdot y_0 + z_0$	$i = 3$: $(z_3, z_2) = \text{reg1} + \text{reg2}$

c, z_3, reg1, and reg2 are 129-bit registers.
z_0, z_1, and z_2 are 128-bit registers.
q is a 64-bit register.
CN: Cycle Number.

Fig. 1 shows the proposed hardware architecture of the proposed R64MMM. It consists of one 129-64-MAC, one 129-bit adder, four 129-bit registers (c, z_3, reg1, and reg2), three 128-bit register (z_0, z_1, and z_2), and one 64-bit register (q).

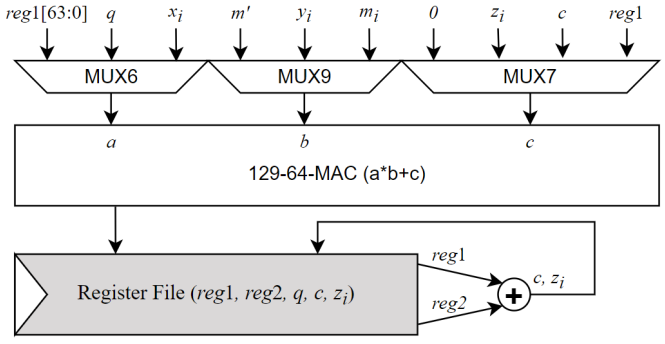

Fig. 1. Hardware architecture of the proposed R64MMM

A Novel Montgomery Ladder scheduling: To reduce the hardware resources, the proposed Montgomery Ladder scheduling just need to call one R64MMM and one 258 bit modular addition and subtraction (258-MA).

Let $x, y, m \in$ GF(p), and $0 \le x, y \le m$. MA is calculated by $(x \pm y) \bmod m$, which needs to call two 258-bit adders to calculate in one clock cycle. Moreover, the 258-MA can be reused to calculate two independent 258-bit additions and subtractions or one radix-2 Euclidean modular inversion (MI) [14] operation. The MI operation costs about 512 clock cycles, which is calculated only once in ECPM.

Table I shows the proposed scheduling of the LADDER, which requires 11 256-bit registers. R64MMM is used to calculate the MMM without the reduction step, and 258-MA is used to perform the reduction step (e.g. $Y_Q \leftarrow sY_Q$) and MA. Assume that two R64MMM operations are $s1 = x1 \cdot y1 \cdot 2^{-256}$ $(2m > s1 > 0)$ and $s2 = x2 \cdot y2 \cdot 2^{-256}$ $(2m > s2 > 0)$. The whole $s1 = x1 \cdot y1 \cdot 2^{-256}$ requires 37 clock cycles, includes two steps: 1) calculating the $CN = 1$~36 operation of the first R64MMM (s1), and 2) using the 129-Adder of the second R64MMM (s2) to calculate the $CN = 37$ operation of the first R64MMM (s1). Therefore, if n consecutive R64MMM without data hazard are calculated, the clock cycles are $36n + 1$.

Table 1: The proposed scheduling of the LADDER

Require: $(X_{QP}, X_{RP}, Y_Q, Y_R, G)$
Ensure: $(X_{QP}, X_{RP}, Y_Q, Y_R, G, M)$

Cycle	R64MMM	R64MMM-LAST	258-MA
1~36	$sY_Q = Y_Q \cdot Y_R$		
37	$sY_R = Y_R \cdot Y_R$	$sY_Q\ (z_3, z_2)$	
38~72			$Y_Q \leftarrow sY_Q$
73	$sX_{QP} = X_{QP} \cdot G$	$sY_R\ (z_3, z_2)$	
74~108			$Y_R \leftarrow sY_R$
109	$sX_{RP} = X_{RP} \cdot G$	$sX_{QP}\ (z_3, z_2)$	
110~144			$X_{QP} \leftarrow sX_{QP}$
145	$sZa = X_{QP} \cdot Y_R$	$sX_{RP}\ (z_3, z_2)$	
145			$X_{RP} \leftarrow sX_{RP}$
146			$J = X_{RP} - Y_Q$
147			$M = J + X_{RP}$
148~180			$M = M - Y_R$
181	$sZb = X_{RP} \cdot J$	$sZa\ (z_3, z_2)$	
181~216			$Za \leftarrow sZa$
217	$sX_{QP} = Y_Q \cdot Y_R$	$sZb\ (z_3, z_2)$	
218~252			$Zb \leftarrow sZb$
253	$sY_Q = J \cdot J$	$sX_{QP}\ (z_3, z_2)$	
254			$X_{QP} \leftarrow sX_{QP}$
255			$X_{RP} = Za + Zb$
256~288			$X_{RQ} = X_{RP} - X_{QP}$
289	$sY_{RQ} = M \cdot X_{RQ}$	$sY_Q\ (z_3, z_2)$	
290			$Y_Q \leftarrow sY_Q$
291~324			$Y_Q = X_{RQ} - Y_Q$
325	$sY_Q = Y_Q \cdot Y_R$	$sY_{RQ}\ (z_3, z_2)$	
326~360			$Y_{RQ} \leftarrow sY_{RQ}$
361	$sG = X_{RQ} \cdot X_{RQ}$	$sY_Q\ (z_3, z_2)$	
362			$Y_Q \leftarrow sY_Q$
363~396			$Y_R = Y_{RQ} + Y_Q$
397	Next-LADDER sY_Q	$sG\ (z_3, z_2)$	
398			$G \leftarrow sG$

1. $X \leftarrow sX$: $X = (sX > \bmod m\ ?\ sX - m : sX)$, required one 258-bit adder.
2. $sX\ (z_3, z_2)$: The last R64MMM calculation of the $i = 3$: $(z_3, z_2) = \text{reg1} + \text{reg2}$.

A Speed-Area-Efficient Hardware ECPM-Engine: We implement a speed-area-efficient hardware ECPM-engine, which supports the calculations of the SETUP, LADDER, and EXIT. To reduce the hardware resources, the proposed ECPM-engine just calls one R64MMM and one 258-MA. This ECPM-engine is used to calculate the 256-bit ECPM, which needs 255 LADDER. The 255 LADDER requires $(255 \cdot 11) \cdot 36 + 1 = 100,981$ clock cycles, and the whole ECPM needs about $\text{cycle}_{\text{SETUP}} + \text{cycle}_{\text{LADDER}} + \text{cycle}_{\text{EXIT}} = 266 + 100,981 + 512 + 297 \approx 102k$ clock cycles.

This ECPM-engine is synthesized by Design Compiler with a 90 nm standard cell library. The result shows that area costs about 72k gates when the clock frequency is 714 MHz. One ECPM operation can be calculated in 0.14 ms by the proposed ECPM-engine. We compare the proposed ECPM-engine with other similar designs, which support 256-bit generic Weierstrass curve [6-13]. The result is shown in Table2.

In Table 2, the hardware-software approach by Liu [10] can improve flexibility but reduce speed. A hybrid modular arithmetic architecture is proposed by Yeh [9] to reduce both area and energy costs. Kudithi [8] designs an optimized resource sharing mechanism to reduce hardware costs. Designs [11-13] aim at high-speed implementations, which achieve high-performance ECPM operations. But they consume too many hardware resources for lightweight applications.

The proposed speed-area-efficient design is focus on achieving lower hardware resource and making a trade-off between speed and area. The area of this design is the smallest in Table2 designs. In terms of the

Table 2: Comparison with other Hardware Implementations of ECPM

Design	Technology	Supported Curve	Area (gates)	Frequency (MHz)	Cycle (k)	Time (ms)	AT	Normalized AT	PA resistance
Ours	90nm	Any	72k	714	102	0.14	10.08	1.0	SPA
[10]	55nm	Any	189k	316	–	1.45	448.44	44.49	SPA,DPA,ZVP
[9]*	90nm	Any	83k	250	212.5	0.85	70.55	7.00	SPA
[8]	40nm	Any	214k	500	300	0.6	288.90	28.66	–
[7]	130nm	Any	122k	556	562	1.01	85.31	8.46	–
[6]	130nm	Any	120k	138	369	2.68	222.65	22.09	–
[13]	90nm	Any	996k	329	5.9	0.017	16.93	1.68	SPA
[12]	65nm	Any	447k	546	397	0.73	451.81	44.82	–
[11]	90nm	Any	540k	185	22.3	0.12	64.80	6.43	SPA

AT = Area · Time · 90nm / Technology.

* indicates that the design supports 192-bit prime and binary fields, instead of a 256-bit prime field.

area-time (AT), this design is also at least 600% and 68% superior to lightweight designs [6-10] and high-speed designs [11-13], respectively.

Conclusion: A 256-bit speed-area-efficient hardware ECC-engine in GF(p) over generic Weierstrass curves is proposed in this paper. The major contributions of this ECC-engine are:

1 A novel R64MMM: The R64MMM calls one 64-129-MAC and one 129-bit adder in parallel. The reduction step of MMM is moved to the process of Montgomery ladder. The Mac utilization in R64MMM operation is close to 100%.
2 A novel Montgomery ladder scheduling: This Montgomery ladder scheduling just calls one R64MMM and one 258-MA to calculate ECPM. The 258-bit MA can be reused to calculate the reduction step of MMM and modular inversion. The R64MMM utilization in ECPM operation is close to 100%.
3 The proposed ECPM-engine consumes 72k gates when the clock frequency is 714 MHz with a 90 nm standard cell library, and it computes one ECPM in 0.14 ms. These results are better than previous designs in terms of AT.

Acknowledgment: This work was supported by the Key-Area Research & Development Program of Guangdong Province under Grant 2021B0101410004 and the National Natural Science Foundation of China under Grant 62274044.

Y. Xie, Z. He, Y. Liu, X. Zheng, S. Cai, and X. Xiong are with School of Integrated Circuits, Guangdong University of Technology, Guangzhou 510006, China.

E-mail: eeliuyuan@gdut.edu.cn, xinzheng@gdut.edu.cn

References

1 N. Koblitz, “Elliptic Curve Cryptosystems,” *Math. Comp.*, vol. 48, pp. 203–209, 1987.
2 X. Hu, X. Zheng, S. Zhang, W. Li, S. Cai, and X. Xiong, “A high-performance elliptic curve cryptographic processor of SM2 over GF(p),” *Electronics*, vol. 8, no. 4, p. 431, Apr. 2019.
3 P. Choi, M. K. Lee, and D. K. Kim, "ECC Coprocessor Over a NIST Prime Field Using Fast Partial Montgomery Reduction," *IEEE Trans. on Circuits Syst. I: Reg. Papers*, vol. 68, no. 3, pp. 1206-1216, Mar. 2021.
4 C. F. Kerry, A. Secretary, C. R. Director, “FIPS Pub 186-4 Fedral Information processing standards publication Digital Signature Standard (DSS),” 2013.
5 R. P. Gallant, R. J. Lambert, and A. S. Vanstone, “Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms,” in Advances in Cryptology—CRYPTO 2001 (Lecture Notes in Computer Science). Heidelberg, Germany: Springer, 2001, pp. 190–200.
6 A. Satoh, and K. Takano, “A scalable dual-filed elliptic curve cryptographic Processor,” *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, Apr. 2003.
7 G. Chen, G. Bai, and H. Chen, “A High-Performance Elliptic Curve Cryptographic Processor for General Curves Over GF(p) Based on a Systolic Arithmetic Unit,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 412–416, May 2007.
8 T. Kudithi, R. Sakthivel, “An efficient hardware implementation of the elliptic curve cryptographic processor over prime field, Fp,” *Int. J. Circ. Theor. App.*, vol. 48, no. 8, pp. 1256–1273, Mar. 2020.
9 L. Y. Yeh, P. J. Chen, C. C. Pai, and T. T. Liu, “An energy-efficient dual-field dlliptic curve cryptography processor for Internet of Things applications,” *IEEE Trans. Circuits Syst. II*, Exp. Briefs, vol. 67, no. 9, pp. 1614–1618, Sep. 2020.
10 Z. Liu, D. Liu, and X. Zou, “An Efficient and Flexible Hardware Implementation of the Dual-Field Elliptic Curve Cryptographic Processor,” *IEEE Trans. Ind. Electron*, vol. 64, no. 3, pp. 2353–2362, Nov. 2016.
11 S. C. Chung, J. W. Lee, H. C. Chang, and C. Y. Lee, “A high-performance elliptic curve cryptographic processor over GF(p) with SPA resistance,” *2012 IEEE International Symposium on Circuits and Systems*, pp. 1456–1459, 2012.
12 M. S. Hossain, Y. Kong, E. Saeedi, and N. C. Vayalil, “High-performance elliptic curve cryptography processor over NIST prime fields,” *IET Comput. Digit. Tec.*, vol. 11, no. 1, pp. 33–42, Nov. 2016.
13 Y. Xie, Y. Liu, X. Zheng, W. Zhu, J. Li, J. Li, S. Cai, and X. Xiong. "A Dual-Core High-Performance Processor for Elliptic Curve Cryptography in GF(p) Over Generic Weierstrass Curves". *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 11, pp. 4523–4527, Nov. 2022.
14 D. Hankerson, A. Menezes, and S. V. Springe, “Guide to Elliptic Curve Cryptography,” Springer, 2004.
15 P. L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". *Math. Comput.*, vol. 48, pp. 243-264, 1987.
16 M. Hambury. "Faster Montgomery and double-add ladders for short Weierstrass curves". *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 4, pp. 189-208.
17 P. L. Montgomery. "Modular multiplication without trial division". *Mathematics of computation*, vol. 44, no. 170, pp. 519-521, 1985.