Revisiting FPGA Implementation of Montgomery Multiplier in Redundant Number System for Efficient ECC Applications in GF(p)

Debapriya Basu Roy*, Debdeep Mukhopadhyay*
*Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

***Abstract*—The fast implementations of ECC in $GF(p)$ are generally implemented using specialized prime field, and henceforth, they are dependent on the structure of the prime. But, these implementations cannot be ported to generic curves which do not support such prime structures. Such generic curves are often used in various crypto-applications like pairing and post-quantum secure supersingular isogeny based key exchange. In those cases, modular multiplication is executed through Montgomery multiplier which is slower compared to modular multiplication using specialized primes. This work aims to reduce the speed gap between Montgomery multiplication and modular multiplication in specialized prime field by presenting an efficient implementation of Montgomery multiplier on FPGA using the redundant number system.**

Keywords: Montgomery multiplier, Redundant number system, ECC

I. Introduction

Public key cryptography is a necessary requirement for key exchange and digital signature algorithm. Among all the public key algorithms, elliptic curve cryptography (ECC) is preferred as it provides more security per key bit. However, computation in ECC is mathematically intensive and introduces significant delay for speed critical applications. To reduce such delays, NIST recommended curves or popular curves like `Curve-25519`, `Ed-1174` are built on prime fields having pseudo-Mersenne or Solinas prime [1]. Modular reduction operation on those prime fields can be computed very fast [2], [3].

+ However, the gain in speed is neutralized by the absence of flexibility in the choice of prime field. ECC applications like bilinear pairing [4] or post quantum supersingular isogeny based key exchange (`SIKE`) [4], [5] are built on non-standard prime field.Therefore designing high speed implementations of those architectures are challenging. The designer, in this case, has to stick with either Montgomery or Barett reductions [6], [7]. Usage of Montgomery multiplication is constant time and is faster compared to modular reduction based on trial division. However, the implementation of Montgomery multiplier in traditional non-redundant number system is still slower compared to fast reduction methodologies of NIST curves.

In this paper, we attempt to achieve the following:

- We revisit the implementation of Montgomery multiplier in redundant number system [8]. Redundant number system supports carry save arithmetic, reducing the critical path significantly. In [8], the authors combined this feature with block RAMs and 18×18 multipliers of Virtex-II FPGA to provide an efficient implementation of Montgomery multiplier. However, the modern 7 series FPGAs have 24×17 unsigned multipliers rather than 18×18 multipliers. This paper provides an updated version of Montgomery multiplier in the redundant number system which utilizes these asymmetric multipliers in an optimal manner.
- We also modify the architecture of Montgomery multiplier to execute multiple modular multiplications simultaneously. We propose 3 different variants of Montgomery multiplier which can simultaneously carry out 1, 2, and 3 modular multiplications respectively.

The rest of the paper is organized as follows. Section II will provide a brief description of existing implementations of Montgomery multiplier. Section III will focus on the basics of the Montgomery multiplication in redundant number base. In section IV, we will discuss the proposed architecture of efficient Montgomery multiplier. Section V will focus on the corresponding performance analysis. Finally, we will conclude our discussion in section VI

II. Related Work

The Montgomery multiplication was first proposed in [7]. After that, there have been multiple works which attempt to develop efficient architectures for its implementation. The authors in [9] proposed a scalable architecture for Montgomery multiplier using systolic array which was later improved in [10], [11], [12], [13]. Residue number system (RNS) is also used for implementing Montgomery multiplication for accelerating ECC [14], [15], [16]. However, RNS implementation of Montgomery multiplication requires two base conversions which increases the complexity.

In [17], the authors have proposed a Montgomery multiplier using quotient pipelining combining it with the redundant adder. To execute 256 bit modular multiplication, it requires 35 clock cycles with a frequency of 291 MHz. Our proposed architecture can carry out the same operation in 15 clock cycles, supporting simultaneous executions of multiple modular multiplications.

The present work aims to improve the implementation of Montgomery multiplication which was originally proposed in [8]. Additionally, we will compare the performance of the proposed architecture with the state of the art Montgomery multiplication which is used in SIKE.

III. Background

A. Redundant Number System [8]

A d digit non-redundant radix-2^r number X is represented as $(X_{d-1}, \dots, X_1, X_0)$ where $|X_i| = r$ and $X = \sum_{i=0}^{d-1} X_i.2^{ir}$. X represents a unique number up to $2^{dr} - 1$. A d digit and n bit redundant number X' can be represented at $(X'_{d-1}, \dots, X'_1, X'_0)$ where $|X'_i| = (r+n)$, $n < r$ and $X' = \sum_{i=0}^{d-1} X'_i.2^{ir}$. We can partition each X'_i into two parts: $X'_i[r-1:0]$ and $X'_i[r+n-1:r]$. $X'_i[r-1:0]$ are the principal bits and $X'_i[r+n-1:r]$ are the redundant bits. Redundant number can be converted into non-redundant number as follows:

$$\begin{array}{cclll} & \dots & X'_1[r+n-1:r] & X'_0[r+n-1:r] & \\ + & \dots & X'_2[r-1:0] & X'_1[r-1:0] & X'_0[r-1:0] \\ \hline & \dots & X_2 & X_1 & X_0 \end{array}$$

From here on, we will consider the value of n as 2 and every redundant digit X'_i is $r+2$ bits long.

1) Overflow condition: A d digit radix-2^r redundant number X' can represent a maximum value of $\sum_{i=0}^{d-1}(2^{r+2} - 1)2^{ir} > 2^{dr}$. However, the permissible value of X' is $2^{dr} - 1$ as it is an equivalent representation of d digit radix-2^r non-redundant number. Whenever the value of X' becomes more than $2^{dr} - 1$, we consider that as overflow.

2) Addition Operation in Redundant Number System: Addition of two d digit redundant number X' and Y' is shown below

$$\begin{aligned} Z'[0] &= X'_0[r-1:0] + Y'_0[r-1:0] \\ Z'[i] &= X'_i[r-1:0] + Y'_i[r-1:0] + X'_{i-1}[r+1:r] \\ &\quad + Y'_{i-1}[r+1:r] \qquad (1 \le i < d) \end{aligned}$$

3) Multiplication in Redundant Number system: Here we will consider multiplication between a d digit redundant number X' and a single digit redundant number Y'. We compute $P_i = X'_i.Y'$ by a $(r+2) \times (r+2)$ bit multiplier. The final multiplication result can be computed as below

$$\begin{aligned} K_0 &= P_0[r-1:0], \quad K_1 = P_0[2r-1:r] + P_1[r-1:0] \\ K_i &= P_{i-2}[2r+3:2r] + P_{i-1}[2r-1:r] + P_i[r-1:0] \\ K_d &= P_{d-2}[2r+3:2r] + P_{d-1}[2r-1:r] \\ K_{d+1} &= P_{d-1}[2r+3:2r] \qquad (2 \le i \le d-1) \end{aligned}$$

B. Montgomery Multiplication

Algorithm 1 Constant Time Montgomery Multiplication [18], [19]

INPUT: $M = \sum_{i=0}^{m-1} m_i \cdot 2^{ri}$,
$A = \sum_{i=0}^{m+2} a_i \cdot 2^{ri}$ with $a_{m+2} = 0$, $B = \sum_{i=0}^{m+1} b_i \cdot 2^{ri}$, $\overline{M} = (M' \bmod 2^r) \cdot M = \sum_{i=0}^{m+1} \overline{m}_i \cdot 2^{ri}$, $A, B < 2\overline{M}$, $4\overline{M} < 2^{rm}$, $R = 2^{r(m+2)}$
OUTPUT: $A \times B \times R^{-1} \bmod M$

```
1: $S_0 = 0$
2: for $i \leftarrow 0$ to $m+2$ do
3:     $q_i = S_i \bmod 2^r$
4:     $S_{i+1} = (S_i + q_i \cdot \overline{M})/2^r + a_i \cdot B$
5: return $S_{m+3} = A \times B \times R^{-1} \bmod M$
```

Montgomery multiplication was proposed in [7] which was adopted in [8] to design an efficient implementation of Montgomery multiplier in the redundant number system. However, such design is not constant time which an adversary can exploit [20]. To prevent this, a high radix Montgomery multiplication was proposed in [18], [19], shown in algorithm 1. It is a constant time implementation [21]. But, the range of the output S_{m+3} is $[0, 2\overline{M}]$. The input to the Montgomery multiplier can be transformed into Montgomery domain by performing Montgomery multiplication between the input and R^2.

IV. Proposed Montgomery Multiplier

The arithmetic operations involved in the Montgomery algorithm are: Multiplication $(a_i \cdot B)$, Multiplication and Accumulation $(S_i + q_i \cdot \overline{M})$, Right Shift $((S_i + q_i \cdot \overline{M})/2^r)$, Addition $((S_i + q_i \cdot \overline{M})/2^r + a_i \cdot B)$ and Mod $(S_{i+1} \bmod 2^r)$. We will illustrate these operations below.

A. Redundant Multiplication using Asymmetric Multipliers

Let $X' = (X'_{d-1}, \dots X'_1, X'_0)$ be a d digit redundant number$(|X'_i| = (r_2+2))$. Similarly Y' is a single digit redundant number having length (r_1+2) and $2r_2 < r_1 + r_2 + 4 < 3r_2$. For DSP blocks with 24×17 asymmetric multipliers, the maximum value of r_1 is 22 and r_2 is 15. The partial products $P_i = X'_i.Y'$ is computed using the asymmetric multiplier of dimension $(r_1+2) \times (r_2+2)$. We can compute the multiplication result as follows:

$$\begin{aligned} K_0 &= P_0[r_2-1:0], \quad K_1 = P_0[2r_2-1:r] + P_1[r_2-1:0] \\ K_i &= P_{i-2}[r_1+r_2+3:2r_2] + P_{i-1}[2r_2-1:r_2] \\ &\quad + P_i[r_2-1:0] \qquad (2 \le i \le d-1) \\ K_d &= P_{d-2}[r_1+r_2+3:2r_2] + P_{d-1}[2r_2-1:r_2] \\ K_{d+1} &= P_{d-1}[r_1+r_2+3:2r_2] \end{aligned}$$

$K_i = P_{i-2}[r_1+r_2+3:2r_2] + P_{i-1}[2r_2-1:r_2] + P_i[r_2-1:0] < 2^{r_2+2}$, ensuring that K is a redundant number

B. Multiplication and Accumulation

Let $X' = (X'_{d-1}, \dots X'_1, X'_0)$ be a d digit and $C = (C_d, \dots C_1, C_0)$ be a $d+1$ digit redundant where each $|X'_i| = |C_i| = (r_2+2)$. Y' is a single digit redundant number with length (r_1+2). The result of $X' \cdot Y' + C$ can be computed as follow:

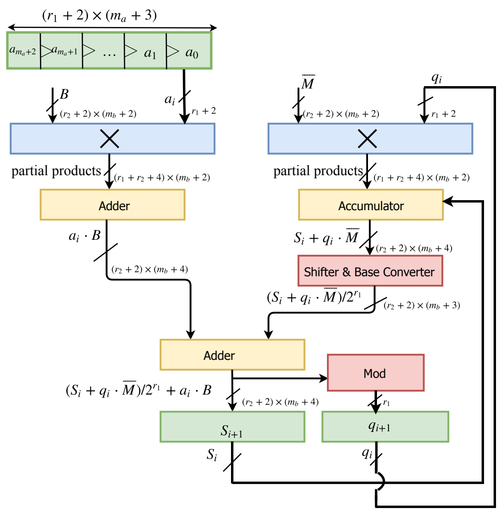
$$\begin{aligned} T_0 &= P_0[r_2-1:0] + C_0[r_2-1:0] \\ T_1 &= P_0[2r_2-1:r] + P_1[r_2-1:0] + C_1[r_2-1:0] \\ &\quad + C_0[r_2+1:r_2] \qquad (2 \le i \le d-1)r \\ T_i &= P_{i-2}[r_1+r_2+3:2r_2] + P_{i-1}[2r_2-1:r_2] \\ &\quad + P_i[r_2-1:0] + C_i[r_2-1:0] + C_{i-1}[r_2+1:r_2] \\ T_d &= P_{d-2}[r_1+r_2+3:2r_2] + P_{d-1}[2r_2-1:r_2] \\ &\quad + C_d[r_2-1:0] + C_{d-1}[r_2+1:r_2] \\ T_{d+1} &= P_{d-1}[r_1+r_2+3:2r_2] + C_d[r_2+1:r_2] \end{aligned}$$

The value of $T_i = P_{i-2}[r_1+r_2+3:2r_2] + P_{i-1}[2r_2-1:r_2] + P_i[r_2-1:0] + C_i[r_2-1:0] + C_{i-1}[r_2+1:r_2] < 3 \cdot 2^{r_2} + 2^{r_1+r_2+4-2r_2} + 2^2 < 2^{r_2+2}$.

C. Implementing Montgomery Multiplier

The execution time of the Montgomery multiplier depends upon the number of iterations of the for loop in algorithm 1. This depends upon the number of digits of operand A. Wwe represent A as a redundant number in radix-2^{r_1} ($A = \sum_{i=0}^{m_a+2} a_i \cdot 2^{r_1 i}$ with $a_{m_a+2} = 0$

Figure 1: Block Diagram of Proposed Multiplier



where $m_a=\lceil\frac{log_2(M)}{r_1}\rceil$). Similarly, we represent B as a redundant number in radix-2^{r_2} ($B=\sum_{i=0}^{m_b+1} b_i\cdot 2^{r_2 i}$, where $m_b=\lceil\frac{log_2(M)}{r_2}\rceil$). The other input $\overline{M}$ is fixed, precomputed and is represented as a redundant number in radix 2^{r_2}. The computational steps of the Montgomery multiplication are:

- The multiplication $a_i\cdot B$ is computed by the asymmetric multipliers as shown in section IV-A. The total number of DSP blocks required for this computation is equal to the number of digits of B (i.e. m_b+2). The value $a_i\cdot B$ is a redundant number in radix 2^{r_2}.
- Similarly, $S_i+q_i\cdot\overline{M}$ can be computed as shown in section IV-B. Here, q_i is a single digit redundant number in radix-2^{r_1} and $\overline{M}$ is m_b+2 digit redundant number in radix-2^{r_2}. The computation of $q_i\cdot\overline{M}$ will require m_b+2 number of DSP blocks. Thus the architecture requires $2m_b+4$ DSP blocks.
- The result of $S_i+q_i\cdot\overline{M}$ now needs to be right shifted by r_1 bits. The result of $S_i+q_i\cdot\overline{M}$ is in radix-2^{r_2}. To right shift this result by r_1 bits, r_1 bits from the LSB side are discarded. However, the resulting value will no longer remain a valid redundant number in radix-2^{r_2}. We need to convert the shifted result into a redundant number in radix-2^{r_2}.
- The next operation is the addition of this shifted result with previously computed $a_i\cdot B$. The result S_{i+1} will be a redundant number in radix-2^{r_2}.

The architectural diagram of the proposed Montgomery multiplier is shown in Fig. 1. To compute the final result, m_a+3 number of clock cycles will be required.

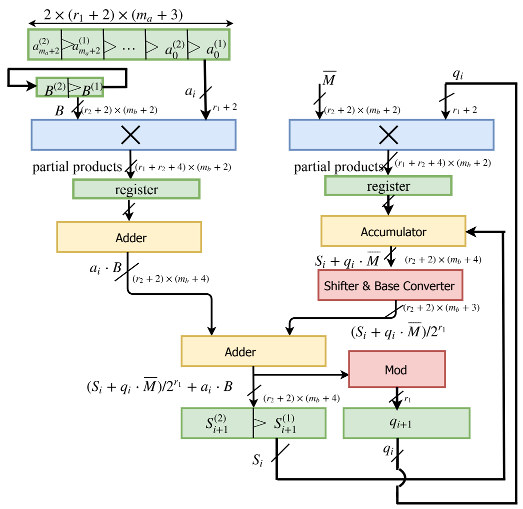
D. *Simultaneous Execution of Two Multiplications*

Each iteration of the for loop requires one clock cycle for its computation. The critical path of the design contains a multiplier followed by series of addition operation which is significant. The critical path of 256 bits Montgomery multiplier on a Virtex-7 FPGA is 10.726 ns which make the maximum frequency of operation less than 100 MHz. To address this issue, we have modified the proposed architecture so that it can support multiple modular multiplications simultaneously. The architecture for simultaneous execution of two modular multiplications is shown in Fig. 2. The critical path of the earlier design (Fig. 1) is partitioned into two different paths by placing registers between the multipliers and adders. This though reduces the critical path value, the clock cycle requirement become twice. The critical observation in this case is that when the multiplier layer is active, the addition layers are inactive. Similarly, when the addition layers are active, the multiplier layer is inactive. This allows us to execute two modular multiplications simultaneously in an interleaved manner. For 256 bit multiplications, the proposed architecture (Fig. 2) can compute two modular multiplications in 31 cycles. This can be extended to support simultaneous execution of 3 modular multiplication by placing another register layer between the addition modules.

Table I: Performance Analysis of Proposed Architecture

Size	Parallel Multiplications	DSP	LUT	Flip-Flops	Slices	Freq. (MHz)	Clock Cycles	Latency (ns)/ Multiplication
256	1	40	1635	736	696	93.23	15	160.89
	2	40	2159	2140	879 (1.26×)	153.4	31	101 (1.6×)
	3	40	2480	3565	937 (1.34×)	207.10	47	76.6 (2.1×)
504	1	72	2931	1248	1135	85.47	26	304.2
	2	72	3811	3712	1540 (1.23×)	149.4	53	177.36 (1.71×)
	3	72	4291	6194	1782 (1.42×)	209.2	80	127.47 (2.38×)

Figure 2: Simultaneous Execution of Two Modular Multiplications



V. Performance Analysis

Table I shows the area and timing performance of the proposed multiplier for prime field of size 256 bits and 504 bits on Virtex-7 FPGA. We propose three different variants of Montgomery multipliers capable of doing 1, 2 and 3 multiplications simultaneously. For both 504 and 256 bits, the architecture supporting 3 simultaneous multiplications is the most efficient one as it provides more than twice speed up. Previous proposal of Montgomery multiplier in redundant number system [8] was implemented on Virtex-II FPGAs. Hence a direct comparison is not possible. However, our proposed architecture is more efficient than [8] as it requires lesser number of clock cycles. The architecture of [8] was built using 18×18 unsigned multipliers, requiring 16 clock cycles to implement 256 bit Montgomery multiplication. Such 18×18 unsigned multipliers are not present in the modern FPGAs anymore. Nevertheless, our architecture consumes 15 clock cycles for one 256 bit Montgomery multiplication. For 504 bits, the architecture of [8] will consume 32 cycles, whereas we require 26 cycles only. DSP block requirement of [8] is less than the half of what we require. However, the reduction of DSP blocks usage is achieved by deploying block RAMs in the design. As block RAMs are used extensively in the implementation of ECC, we have stuck with using DSPs.

Among the various systolic array based implementations of Montgomery architecture, design of [13] was adopted by [22] for developing architecture of (SIKE). The original design of [13] requires 103 cycles for one 512 bit multiplication which was modified in [22] to support execution of two multiplications simultaneously. The first multiplication pair will take 99 clock cycles, and the next pair can be interleaved after 68 cycles. This reduces the latency to 34 cycles. Compared to that, our proposed architecture provides the result in 27 cycles, supporting simultaneous execution of 3 multiplications. The operating frequency of [22] and our design are similar (191.2 and 209.2 MHz respectively). This implies that our proposed architecture has the potential to improve the implementation of SIKE.

Modular multiplication in specialized prime form is carried out by executing normal multiplication followed by fast modular reduction using Solinas or pseudo-Mersenne prime. For example, in [3], the modular multiplications in Curve 25519 requires 55 cycles, with next multiplication can start after the interval of 17 cycles with clock frequency of 200 MHz. Compared to that, our design requires 15 cycles for each multiplications, supporting three multiplications simultaneously with the clock frequency of 207 MHz. This shows, that our proposed architecture is competitive even with modular multiplication in special primes.

VI. Conclusion

In this paper, we have proposed an efficient architecture for constant time Montgomery multiplication in the redundant number system using the asymmetric multipliers of Modern FPGAs. The proposed architecture consumes fewer number of clock cycles. Additionally, the proposed architecture can execute multiple modular multiplications simultaneously. For 256 and 504 bit multiplication, the architecture supporting the execution of 3 simultaneous modular multiplication is the most efficient one. This makes the proposed architecture of modular multiplication competitive even with modular multiplication using special primes. We have also shown that the proposed architecture perform better than the systolic array based Montgomery multiplier which is used in the implementation of SIKE.

References

[1] "Safecurves: Introduction," https://safecurves.cr.yp.to/.

[2] T. Güneysu, "Utilizing hard cores of modern fpga devices for high-performance cryptography," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 37–55, Apr 2011.

[3] P. Sasdrich and T. Güneysu, *Efficient Elliptic-Curve Cryptography Using Curve25519 on Reconfigurable Devices.* Cham: Springer International Publishing, 2014, pp. 25–36.

[4] V. S. Miller, "The weil pairing, and its efficient calculation," *Journal of Cryptology*, vol. 17, no. 4, pp. 235–261, 2004.

[5] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," in *Annual Cryptology Conference.* Springer, 2016, pp. 572–601.

[6] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques.* Springer, 1986, pp. 311–323.

[7] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, 1985.

[8] K. Shigemoto, K. Kawakami, and K. Nakano, "Accelerating montgomery modulo multiplication for redundant radix-64k number system on the fpga using dual-port block rams," in *EUC'08. IEEE/IFIP International Conference on*, vol. 1. IEEE, 2008, pp. 44–51.

[9] A. F. Tenca and C. K. Koç, "A scalable architecture for montgomery nultiplication," in *CHES-1999.* Springer, 1999.

[10] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An improved unified scalable radix-2 montgomery multiplier," in *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on.* IEEE, 2005, pp. 172–178.

[11] M. Huang, K. Gaj, and T. El-Ghazawi, "New hardware architectures for montgomery modular multiplication algorithm," *IEEE Transactions on computers*, vol. 60, no. 7, pp. 923–936, 2011.

[12] C. McIvor, M. McLoone, and J. V. McCanny, "Modified montgomery modular multiplication and rsa exponentiation techniques," *IEE Proceedings-Computers and Digital Techniques*, vol. 151, no. 6, pp. 402–408, 2004.

[13] ——, "High-radix systolic modular multiplication on reconfigurable hardware," in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on.* IEEE, 2005, pp. 13–18.

[14] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel montgomery multiplication," in *International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2000, pp. 523–538.

[15] J.-C. Bajard and N. Merkiche, "Double level montgomery cox-rower architecture, new bounds," in *International Conference on Smart Card Research and Advanced Applications.* Springer, 2014, pp. 139–153.

[16] P. M. Matutino, R. Chaves, and L. Sousa, "A compact and scalable rns architecture," in *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on.* IEEE, 2013, pp. 125–132.

[17] Y. Ma, Z. Liu, W. Pan, and J. Jing, "A high-speed elliptic curve cryptographic processor for generic curves over \mathrm {GF}(p) ," in *International Conference on Selected Areas in Cryptography.* Springer, 2013, pp. 421–437.

[18] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Computer Arithmetic, 1995., Proceedings of the 12th Symposium on.* IEEE, 1995, pp. 193–199.

[19] T. Blum and C. Paar, "High radix montgomery modular exponentiation on reconfigurable hardware for public-key cryptography,"," *IEEE Transactions on Computers. to appear.*

[20] S. Bhattacharya and D. Mukhopadhyay, "Who watches the watchmen?: Utilizing performance monitors for compromising keys of rsa on intel platforms," in *CHES-2015.* Springer, 2015.

[21] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics letters*, vol. 35, no. 21, pp. 1831–1832.

[22] B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Post-quantum cryptography on fpga based on isogenies on elliptic curves," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 1, pp. 86–99, 2017.