Exploring RNS for Isogeny-based Cryptography

David Jacquemin, Ahmet Can Mert, Sujoy Sinha Roy
Institute of Applied Information Processing and Communications, Graz University of Technology, Austria
{david.jacquemin,ahmet.mert,sujoy.sinharoy}@iaik.tugraz.at

***Abstract*—Isogeny-based cryptography suffers from a long-running time due to its requirement of a great amount of large integer arithmetic. The Residue Number System (RNS) can compensate for that drawback by making computation more efficient via parallelism. However, performing a modular reduction by a large prime which is not part of the RNS base is very expensive. In this paper, we propose a new fast and efficient modular reduction algorithm using RNS. Our method reduces the number of required multiplications by 40% compared to RNS Montgomery modular reduction algorithm. Also, we evaluate our modular reduction method by realizing a cryptoprocessor for isogeny-based SIDH key exchange. On a Xilinx Ultrascale+ FPGA, the proposed cryptoprocessor consumes 151,009 LUTs, 143,171 FFs and 1,056 DSPs. It achieves 250 MHz clock frequency and finishes the key exchange for SIDH in 3.8 and 4.9 ms.**

***Index Terms*—Post-quantum cryptography, Isogeny, Residue Number System**

I. Introduction

Post-quantum Cryptography (PQC) focuses on developing new cryptographic schemes that are resistant to attacks from quantum computers. Isogeny-based cryptography is a class of PQC algorithms that rely on the isogeny problem for security. Various cryptographic constructs use the isogeny problem in different ways, SIDH (Supersingular Isogeny Diffie-Hellman) [1] and its more efficient variant SIKE (Supersingular Isogeny Key Encapsulation) which is a 4th round candidate in the PKE/KEM category. SQISiqn [2] is a new isogeny-based signature algorithm. One of the main advantages of isogeny-based cryptography over other PQC schemes is that isogeny-based schemes tend to use much shorter keys than the other PQC schemes. Various works exist in the literature ([3], [4], [5], [6]) that present optimized implementations of isogeny cryptography in software, hardware and co-design. One of the main drawbacks of isogeny cryptographic schemes is that it is relatively slow compared to other PQC classes. In the literature, the Residue Number System (RNS) which is a number representation system, has been used to increase the performance of multiple cryptographic schemes such as Elliptic curves cryptography (ECC) [7] and RSA [8]. RNS is a hardware-friendly numeral system as it allows parallel computation of small numbers, which is very useful feature for hardware acceleration.

Our contributions: Our goal is to explore the potential of RNS to improve the performance of isogeny-based cryptography. We make the following contributions towards this goal.

1) RNS is not friendly to modular reduction by a modulus p different from the base. Such a modular reduction by a non-base modulus is very expensive to compute. We propose a new and more efficient method for performing such modular reductions in the RNS.
2) We develop all the finite field primitives for computing isogeny-based cryptography fully in the RNS.
3) To experimentally evaluate the potential of RNS in isogeny-cryptography, we design and construct an instruction-set architecture for computing isogeny-based key exchange protocol SIDH [1] as a case study. The architecture uses the optimised finite field primitives that perform computation in the RNS. To the best of our knowledge, our work is the first to propose an implementation of isogeny-based cryptography in the RNS. Furthermore, our architecture achieves the fastest SIDH in the literature.

The paper is organized as follows. Sec. II briefly describes the mathematical background. Sec. III explains the algorithmic improvement. Next, in Sec. IV, we presented our hardware architecture. Area and performance results are presented in Sec. V. Discussions on side-channel security and the conclusion are presented in Sec. VI.

II. Preliminaries

A. Notation

For an integer a, we will use $|a|_p$ to denote $a \mod p$. In our case, p is the prime modulus of the underlying prime field $\mathbb{F}_p$. The quadratic extension field $\mathbb{F}_{p^2}$ is constructed as $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 = 1$. Let, $Q = \prod_{i=1}^{n} q_i$ be the product of n co-primes q_i. The RNS base with respect to the modulus Q consists of all the n co-primes q_i and it is denoted as $\mathcal{B}$. An integer $a \mod Q$ in the RNS base $\mathcal{B}$ is the vector $\overline{A} = \{a_1, \ldots, a_{n-1}\}$ where $a_i = a \mod q_i$ for $i \in [1, n]$. We also introduce the notations $\tilde{Q}_i = \frac{Q}{q_i}$ for all $i \in [1, n]$. In this paper, all q_i are co-primes of the same bit length s and with the special structure $q_i = 2^s - c_i$ with $c_i < 2^{s/2}$. Note that q_i need not be a prime. The prime p of the base finite field is of length e bits. For two integers b and c, we will use $\|b - c\|$ as the absolute difference between b and c.

B. Basics of Isogeny-based Cryptography

Let E_a and E_b be two elliptic curves on $\mathbb{F}_{p^2}$. We define an isogeny $\phi : E_a \rightarrow E_b$ as a non-constant rational map between two elliptic curves that preserves the identity $\mathcal{O}$. Two elliptic curves are isogenous if their order (number of points) is identical. The search version of the isogeny problem refers

This work was supported in part by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

to the following: from two isogenous curves, E_a and E_b, compute an isogeny ϕ between the two curves. The problem is presumed to be computationally infeasible. Key-exchange protocol like SIDH [1] uses this problem and performs a random walk in the isogeny graph. Other PQC schemes, such as the public key exchange CSIDH [9] or the signature scheme SQISign [2] use the isogeny problem in other ways for their scheme constructions. In July 2022, Castryck *et al.* [10] proposed an attack that broke SIDH and SIKE in polynomial time. The attack is SIKE/SIDH specific and uses the three additional elements exchanged in the SIDH/SIKE protocol to perform a successful key recovery. Other isogeny-based schemes e.g., CSIDH [9], SQISign [2], etc., remain secure from the above attack.

C. Residue Number System (RNS)

The RNS is a numeral system that uses a set of co-primes moduli to represent a number. The base of the RNS consists of all the moduli and is $\mathcal{B} = \{q_1, q_2, \dots, q_{n-1}, q_n\}$. An integer a is represented in the RNS base $\mathcal{B}$ as the vector $\overline{A} = (|a|_{q_1}, |a|_{q_2}, \dots, |a|_{q_{n-1}}, |a|_{q_n})$. From a given RNS representation $\overline{A}$, we can obtain the integer a by applying the Chinese Remainder Theorem (CRT) as $a = \big|\sum_{i=1}^{n} |a_i * \tilde{Q}_i^{-1}|_{q_i} * \tilde{Q}_i\big|_Q$. With the application of RNS, a long integer computation on $a \mod Q$ gets mapped into several small integer computations on $a_i \mod q_i$. On parallel platforms these small integer operations can be performed in parallel.

When RNS is used, Montgomery reduction is the common method to perform a modular reduction in RNS. Transposing the classic Montgomery reduction algorithm into RNS representation was first proposed in [11] and later improved in [12], [13], [14]. The RNS variant of Montgomery reduction requires base extension. Interested readers may follow the original paper [11] for detailed description of the reduction algorithm. There are several RNS variants of the Montgomery reduction and the best of them requires $2 * n^2 + n$ unit multiplications where n is the number of moduli in the RNS base [15].

III. Proposed Optimization Techniques

In isogeny-based cryptography, we need to perform modular arithmetic in the prime field $\mathbb{F}_p$ where p is a several hundred bits long prime. To perform the finite field operations in RNS, we need a sufficiently large RNS base of a composite Q such that $Q > p^2$. Such a Q will ensure that, for elements a, b $\in \mathbb{F}_p$, when their equivalent RNS representations $\bar{A}$ and $\bar{B}$ respectively are multiplied, the result is never larger than Q thus avoiding a true reduction by Q.

A. Modular addition and subtraction in RNS

To do finite field arithmetic over $\mathbb{F}_p$ using the RNS representation which work modulo Q, we need to perform a reduction modulo p. When performing long integer representation addition or subtraction of a and b, the modular reduction becomes an inequality test $(a + b > p)$ or $(a - b < 0)$ followed by an conditional addition or subtraction of p, in order to put the result in the range $[0, p-1]$. However as we mentioned earlier, testing these inequalities in RNS involves computing the CRT which is extremely expensive. We avoid this problem simply by not doing a reduction following an addition or a subtraction and let the data leave the normal finite field range $[0, p-1]$. Such an overflow does not cause any harm as Q is much larger than p. We perform modular reduction by p only after multiplication because the result becomes double in the size. This step is very costly and usually involves using a special algorithm (RNS-based Montgomery reduction here). But, this algorithm involves using multiple RNS bases. Hence why we chose to design our own novel approach for a reduction in RNS using features from hardware implementation in Sec. III-B.

B. Novel RNS Modular Reduction for Multiplication Result

Let $\overline{A} = (a_1, \dots, a_n)$ be the RNS representation of a which is the result of a multiplication modulo Q. Our goal here is to compute $\overline{A} \pmod p$. The RNS version of the Montgomery Reduction algorithm [11] is able to compute this result from $\overline{A}$ using $2*n^2+n$ unit multiplications. We propose a new and more efficient way for reduction. Starting from $\overline{A}$, we obtain a using CRT as follows:

$$a = |\textstyle\sum_{i=1}^{n} |a_i * \tilde{Q}_i^{-1}|_{q_i} * \tilde{Q}_i|_Q = \sum_{i=1}^{n} |a_i * \tilde{Q}_i^{-1}|_{q_i} * \tilde{Q}_i - k * Q \quad (1)$$

with $k = \lfloor (\sum_{i=1}^{n} |a_i * \tilde{Q}_i^{-1}|_{q_i} * \tilde{Q}_i)/Q \rfloor$. Eq. 1 uses very expensive multi-precision arithmetic for processing the long integer operations such as a multiplication by $\tilde{Q}_i$ and $k * Q$ which we want to avoid at all cost. Our aim here is to break down these large integer operations into smaller ones. We will use $v_i = |a_i * \tilde{Q}_i^{-1}|_{q_i}$. So, now we take Eq. 1 mod p:

$$|a|_p = (\sum_{i=1}^{n} v_i * \tilde{Q}_i - k * Q) \bmod p \quad (2)$$

$$= (\sum_{i=1}^{n} v_i * |\tilde{Q}_i|_p - |k * Q)|_p) \bmod p \quad (3)$$

We define $M_{sum} = \sum_{i=1}^{n} |a_i * \tilde{Q}_i^{-1}|_{q_i} * |\tilde{Q}_i|_p - |k * Q|_p$ and then replace the reduction with a subtraction by $r * p$ where r is defined as $r = \lfloor M_{sum}/p \rfloor$. Hence:

$$|a|_p = \sum_{i=1}^{n} v_i * |\tilde{Q}_i|_p - |k * Q|_p - r * p \quad (4)$$

We use a quotient r here to complete our reduction after Eq. 3. Here, $size(r) = s + size(n)$ is very close to the size of an RNS moduli. Next, we take the modulo p reduced result from Eq. 4 and then perform reductions mod q_i for all the moduli of the RNS base to bring the reduced result back into the RNS domain (i.e., modulo Q).

$$|a|_p \pmod{q_i} = (M_{sum} - r \cdot p) \mod q_i \quad (5)$$

$$= \Big|\sum_{i=1}^{n} v_i \cdot ||\tilde{Q}_i|_p|_{q_i} - ||k \cdot Q|_p|_{q_i} - |r \cdot p|_{q_i}\Big|_{q_i}$$

That transforms all the long integer operations in Eq. 1 into small s bit operations in the RNS. In Eq. 5, we have expressed $a \mod p$ as a function of $\overline{A}$ (which is present in v_i) and others parameters. The constants ($\tilde{Q}_i^{-1}$, $||\tilde{Q}_i|_p|_{q_i}$, Q and p) can be pre-computed, while the two variables k and r must be computed during the reduction. In [14], the author mentions an effective way to compute the first reduction factor k, by approximating it with $\overline{k}$:

$$\overline{k} = \sum_{i=1}^{n} trunc(v_i, l)/2^s + \alpha \tag{6}$$

The $trunc$ function takes in a bit length l, an integer v_i, and returns the l most significant bits of the input integer. The parameters α is a float that represents an error correction, s is the bit length of the RNS moduli. The paper [14] also explained the requirement for the parameters to guarantee that $\overline{k}$ will be equal to k.

To find r, our approach is to approximate the values of M_{sum} and $k * Q$ to compute the approximation $\overline{r}$:

$$\overline{r} = \frac{1}{p} * \sum_{i=1}^{n} v_i * \overline{|\tilde{Q}_i|_p} - \overline{|k * Q|_p} \tag{7}$$

with $\overline{|\tilde{Q}_i|_p} = trunc(|\tilde{Q}_i|_p, u)$ and $\overline{|k * Q|_p} = trunc(|k * Q|_p, u)$, u being our approximation accuracy. This method works well because the size of r is very close to s; thus we can select u to be either s or $2 * s$ turning most operations into s bits operations where s is the bit length of RNS moduli q_i. Therefore, our method effectively eliminates long integer arithmetic. In a later part of this subsection, we will discuss the advantages of our optimization over the Montgomery method of modular reduction.

Probability of incomplete reduction: Our approach introduces approximation, which means that the reduction algorithm will incompletely reduce an input value. There are two sources of approximation: the value of $\overline{k}$ (Eq. 6) and $\overline{r}$ (Eq. 7). We have carefully chosen our parameters to always get the correct output for $\overline{k}$. For the approximation $\overline{r}$ of r, the main source of approximation error comes from the fact that we only took into account the u most significant bits of $|\tilde{Q}_i|_p$ and $|k * Q|_p$, leading to the risk of missing a carry propagation from the lower bits to the higher bits that would increase the true value of r by one. We will now estimate the impact of the approximation of r by calculating the distance $\delta(r) = \|r - \overline{r}\|$ using Eq. 7:

$$\delta(r) = \|\frac{1}{p} * (\sum_{i=1}^{n} v_i * (|\tilde{Q}_i|_p - \overline{|\tilde{Q}_i|_p}) - (|k * Q|_p - \overline{|k * Q|_p}))\|$$

$$\delta(r) \leq \frac{1}{p} * (\sum_{i=1}^{n} v_i * \delta(|\tilde{Q}_i|_p) + \delta(|k * Q|_p)).$$

$\overline{|\tilde{Q}_i|_p}$ and $\overline{|k * Q|_p}$ are an approximation of $|\tilde{Q}_i|_p$ and $|k * Q|_p$ respectively, by taking theirs most significant u bits. Since both $|\tilde{Q}_i|_p$ and $|k * Q|_p$ are e bits long, only their least significant $e - u$ bits will be different. Therefore, $\left\|\overline{|k * Q|_p} - |k * Q|_p\right\| \leq 2^{e-u}$ and $\left\|\overline{|\tilde{Q}_i|_p} - |\tilde{Q}_i|_p\right\| \leq 2^{e-u}$. Note that the prime p is e bits long and therefore 2^{e-1} is smaller than p. Also note that $\forall i \in [1, n]$ $v_i \leq 2^s$ since q_i are s bits long moduli.

Algorithm 1 Proposed Reduction Algorithm for RNS

```
Input: $\overline{A}$ in RNS base $\mathcal{B}$
Input: $\tilde{Q}^{-1}$ in $\mathcal{B}$, $\forall i, j$ $\left||\tilde{Q}_i|_p\right|_{q_j}$, $\overline{\tilde{Q}_p} = ([\tilde{Q}_i]_p)_{0<i\leq n}$
Output: $\overline{B}$ in $\mathcal{B}$, $CRT(\overline{B}) = |CRT(\overline{A})|_p + u * p$, $u \in \{0, 1\}$
 1: $\overline{V} \leftarrow \overline{A} * \overline{\tilde{Q}^{-1}}$
 2: $\overline{k} \leftarrow int(\alpha * 2^l)$, $\overline{r} \leftarrow 0$, $\overline{Y} \leftarrow [0, 0, ..., 0]$
 3: for ($i = 1$; $i \leq n$; $i++$) do
 4:     $\overline{k} \leftarrow \overline{k} + (\overline{V}[i] \gg (s - l))$
 5:     $\overline{r} \leftarrow \overline{r} + \overline{V}[i] * (\overline{\tilde{Q}_p}[i] \gg (e - u))$
 6:     for ($j = 1$; $i \leq n$; $i++$) do
 7:         $\overline{Z}[i] = \overline{Z}[i] + \overline{V}[i] * \left||\tilde{Q}_i|_p\right|_{q_j}$
 8:     end for
 9: end for
10: $\overline{k} \leftarrow (\overline{k} \gg l)$
11: $\overline{r} \leftarrow (\overline{r} - |\overline{k} * Q|_p) * \lfloor 2^{e+t}/p \rfloor$    ▷ $t$ arbitrary chosen $> 0$
12: $\overline{r} \leftarrow (\overline{r} \gg t + u)$
13: for ($i = 1$; $i \leq n$; $i++$) do
14:     $\overline{B}[i] = \overline{Z}[i] - ||\overline{k} * Q|_p|_{q_i} - \left|\overline{r} * |p|_{q_i}\right|_{q_i}$
15: end for
16: return $\overline{B}$
```

$$\begin{aligned}\delta(r) \leq &\quad \tfrac{1}{p} * (\textstyle\sum_{i=1}^{n} v_i * 2^{e-u} + 2^{e-u}) \\ \leq &\quad \tfrac{2^{e-u}}{2^{e-1}} * (\textstyle\sum_{i=1}^{n} v_i + 1) = 2^{1-u} * (\textstyle\sum_{i=1}^{n} v_i + 1) \\ \leq &\quad 2^{1-u} * (\textstyle\sum_{i=1}^{n} 2^s + 1) = 2^{1-u} * (n * 2^s + 1)\end{aligned}$$

So supposing a uniform distribution of values, our approach has a $2^{1-u} * (n * 2^s + 1)$ chance of not outputting the correct value of r. This is not problematic at all, as $\overline{r}$ can only take two values, either r or $r - 1$. When $\overline{r} = r - 1$, an incomplete reduction takes place, meaning the output will be $|d|_p + p$ instead of $|d|_p$ for some d modulo p. Such an incomplete reduction is harmless, as the RNS can hold much bigger integers – and the impact of incomplete reduction gets compensated during the next reduction operation. Algorithm 1 describes our approach for the reduction.

Example: For our use-case, we have selected the SIKE prime `SIKEp503` with the prime of size $e = 503$. We work with following the RNS parameters $s = 48$ (size of q_i), $u = 96 = 2 * s$ and $n = 22$. We chose this value of s, because it is very friendly for hardware multiplication via the use of DSPs. Similarly, we have also selected $\alpha = 0.5$ and $l = 18$, for the approximation of k. With the chosen parameters, the probability of an incomplete reduction becomes

$$\delta(r) \leq 2^{1-u} * (n * 2^s + 1) = 2^{-95} * (22 * 2^{48} + 1) \leq 2^{-42}.$$

Here, our approach has less than 2^{-42} chance of generating an incomplete approximation (which is harmless) of r.

Advantages of our method over RNS Montgomery: In Alg. 1, Line 1 has one RNS vector multiplication requiring n unit multiplications. Line 5 has $\lfloor u/s \rfloor$ multiplications in a for-loop and the total number becomes $\lfloor u/s \rfloor * n$ unit multiplications. Line 7 has a multiplication in a double for-loop and the total number becomes n^2 unit multiplications. Line 14 has a multiplication $\overline{r} * |p|_{q_i}$ in a for loop, thereby requiring total n unit multiplications. Summing all, our method of reduction has a cost of $n^2 + (2 + \lfloor u/s \rfloor) * n$ unit multiplications which is nearly half the cost for the RNS Montgomery reduction. It also uses mostly s bit arithmetic, except in the computation $\overline{r}$ where we need one u bits subtraction and $\lfloor u/s - 1 \rfloor$ addition. Both this method and the RNS Montgomery reduction have a small chance of computing a partial reduction, meaning that the output will not always be within $[0, p-1]$. For our example with the 503 bit prime, we need to compute $22^2 + (2 + \lfloor 96/48 \rfloor) * 22 = 572$ unit multiplications, while using the fastest RNS Montgomery reduction would cost $2 * 22^2 + 22 = 990$ unit multiplications. Therefore, we obtain a 43% decrease in the computation cost. For larger primes, the gain will be even larger.

C. Handling of Negative Numbers

In the RNS representation, any arithmetic operation between two integers implicitly perform a modular reduction by Q. This reduction is actually problematic for negative numbers. E.g., let us consider the RNS base $\mathcal{B} = [31, 32, 33]$ for $Q = 32,736$. Let two integers be $a = 1,052$ and $b = 18,976$ with the RNS representations $\overline{A} = [29, 28, 29]$ and $\overline{B} = [4, 0, 1]$ respectively in $\mathcal{B}$. When we perform the normal subtraction directly on the long integers a and b then we first obtain $a - b = -17,924$. Next, the result is reduced modulo p. On the other hand, when the subtraction $a - b$ is performed in the RNS, the result is $\overline{[25, 28, 28]}$. Combining them using the CRT gives $\text{CRT}(\overline{A} - \overline{B}) = 14,812 = -17,924 + Q$. Note that our goal is to correctly compute $a - b \pmod p$. However, the automatic reduction of a negative result modulo Q in the RNS will add the term $Q \mod p$ to the result. To solve this problem, we chose to represent integers in the central domain. We change the range of our field arithmetic from $[0, p-1]$ to $[-(p-1)/2, (p-1)/2]$, and then translated that into the RNS changing our effective range to $[0, (p-1)/2]$ for positive integers and $[Q - (p-1)/2, Q-1]$ for negatives integers.

IV. Hardware Architecture

In this section, we present a highly parallel hardware architecture for implementing isogeny-based cryptography in the RNS. As a case study, we optimize the hardware architecture for SIDH, which enables fair comparisons since there are several hardware implementations of SIDH in the literature. We would like to remark that the proposed algorithmic optimization techniques are applicable to any cryptographic scheme that uses long integer prime field arithmetic, e.g., isogeny-based signature schemes and CSIDH.

For the overall cryptoprocessor, we choose an instruction-set architecture (ISA) framework. In this framework, the $\mathbb{F}_{p^2}$ operations are the 'instructions'. The high-level block diagram of the cryptoprocessor architecture is shown in Fig. 1. The arithmetic core compute the $\mathbb{F}_{p^2}$ and $\mathbb{F}_p$ arithmetic. Note that any arithmetic in $\mathbb{F}_{p^2}$ essentially gets transformed into several arithmetic operations in $\mathbb{F}_p$. The memory unit stores all the

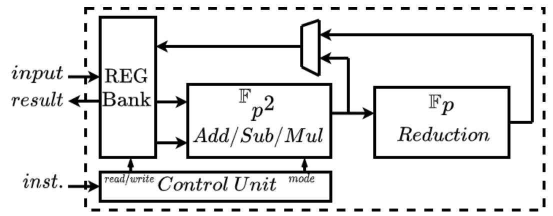

Fig. 1. Block diagram of cryptoprocessor architecture.

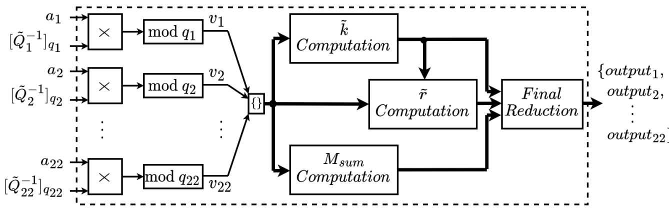

Fig. 2. High-level architecture diagram of the reduction unit. Block with {} symbol represents concatenation operation.

data during the protocol computation (e.g., SIDH). It consists of registers and multiplexers. The control unit generates the control signals during the protocol for performing the arithmetic operations and accessing the register. In the following part of this section, we describe internal architectures of the main arithmetic core. We use 503-bit prime `SIKEp503` to describe the design decisions for the architecture.

A. $\mathbb{F}_{p^2}$ Addition/Subtraction/Multiplication Unit

Size of the coprimes in the RNS base is an important design parameter. We choose the size $s = 48$ bits for each coprime so that the DSP units in the FPGA can be utilized optimally. For the $e = 503$ bit prime `SIKEp503`, the composite modulus Q has to be larger than $(e+5) * 2$ bits. Therefore, the RNS basis of Q consists of 22 coprimes in this case. Each coprime $q_i = 2^s - c$ has a sparse 'Mersenne prime' like structure so that a modular reduction by q_i can be performed following cheap additions and subtractions. In RNS, standard arithmetic operations such as addition (ADD), subtraction (SUB) and multiplication (MUL) are done independently for every RNS moduli. We translate that in our architecture as doing 22 operations in parallel. As there is no data dependency, we choose to instantiate 22 parallel 48 bits adder/subtractor circuits for computing ADD and SUB in one cycle. Similarly, 22 parallel 48 bit multipliers are used to perform any multiplication in the RNS base. One 48-bit multiplier is composed of six DSP units in the Xilinx FPGA. As each q_i has a reduction friendly sparse structure, reducing the result of an integer multiplication modulo q_i takes only two cycles.

B. $\mathbb{F}_p$ Reduction Unit

This section describes the design decisions we made for implementing our RNS reduction unit. We will consider $\overline{A}$, the RNS representation of an integer a as our unit input. We use the method described in Sect. III-B and Fig. 2 shows a high-level description of the reduction unit. The first step is to compute the multiplication $\overline{V} \leftarrow \overline{A} * \overline{\tilde{Q}^{-1}}$ (Line 1 of Alg. 1). As shown in Fig. 2, we use 22 RNS modular multipliers

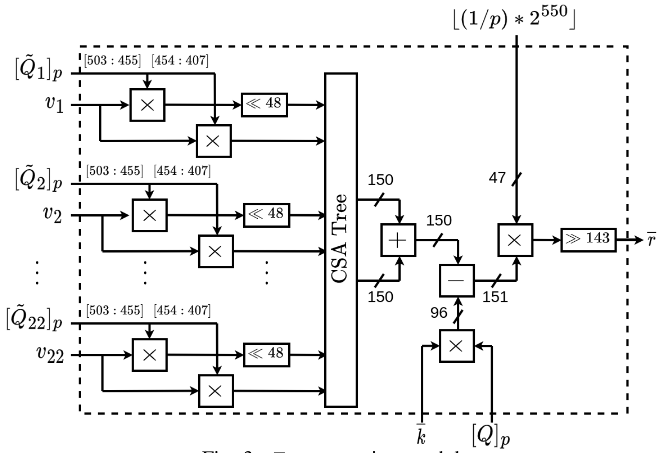

Fig. 3. $\overline{r}$ computation module.

in parallel for computing $\overline{A} * \overline{\tilde{Q}^{-1}}$. We split the remaining computations into four parts, two blocks to compute the two variables $\overline{k}$ and $\overline{r}$, one block to calculate M_{sum}, and the last one combining all results for the final reduction.

Computation of $\overline{k}$: The computation of $\overline{k}$ is based upon the approximation method proposed by [14] and given in Eq. 6 for the following parameters: $\alpha = 0.5$ and $l = 18$. We select those parameters in the example part of Sec. III-B to guarantee the correctness of the approximation. We start by right-shifting the $s - l = 30$ bits of the input. As established in Eq. 6, the main operation of this step is to compute a large sum. We use a carry-save adder tree to perform the large sum in one cycle. We finish this step by another 18 bits right shift of the sum to get $\overline{k}$. Right-shifting the bits before and after the CSA tree replaces the trunc( ) function and the division by 2^{48}. It also allows us to avoid using floating point arithmetic as we convert them into 18 bits integers.

Computation of $\overline{r}$: We have expressed how we compute $\overline{r}$ in Eq. 6 and Fig. 3 shows the high level diagram of our block. The first step here is the multiplication of $v_i * \overline{|\tilde{Q}_i|_p}$ $\forall i \in [1, 22]$. $\overline{|\tilde{Q}_i|_p}$ is a 96-bit integer, so we split it up into a high and low bits part to change the large multiplication into two 48 bits ones. We use 44 multipliers to compute this step. We then use a large CSA tree to add all the terms together. The next step is to subtract the previous accumulation by trunc($\overline{k} * Q \pmod p, 96$). We compute $\overline{k}$ parallel to this whole block in Fig. 2, its value becomes available before the trunc($\overline{k} * Q \pmod p, 96$) computation. As $\overline{k}$ can only take a value between $[0, 22]$, we use a small table (ROM) to store all the possible combination of trunc($\overline{k} * Q \pmod p, 96$) and select the correct value accordingly. We use a 151-bit subtractor circuit to calculate that 'large' subtraction. The fourth step is the division by p. We change this operation to multiplication by the inverse of p. We pre-computed the value of $\lfloor 1/p * 2^{503+t} \rfloor$ with $t = 47$ which is a 47-bit integer. We use a constant multiplier circuit with the constant value being the 47-bit inverse of p on to the output of our subtraction. The result is then right shifted by $t + 96 = 143$ bits to get $\overline{r}$, see Fig. 3. The choice for t is arbitrary, but we recommend it to be near 48 to fit with RNS arithmetic.

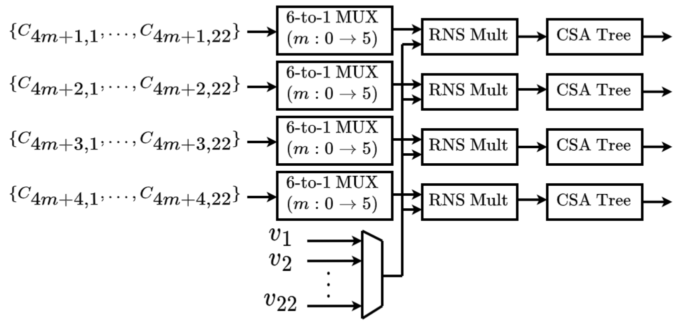
$\{c_{4m+1,1}, \cdots, c_{4m+1,22}\}$
$\{c_{4m+2,1}, \cdots, c_{4m+2,22}\}$
$\{c_{4m+3,1}, \cdots, c_{4m+3,22}\}$
$\{c_{4m+4,1}, \cdots, c_{4m+4,22}\}$
6-to-1 MUX
$(m : 0 \to 5)$
RNS Mult
CSA Tree
v_1
v_2
v_{22}

Fig. 4. M_{sum} computation module. Each input/output signal is 48-bit long.

Computation of M_{sum} and final reduction: This module computes M_{sum} (Eq. 4) on every RNS moduli. This block is the most expensive part of our reduction module (we perform n^2 unit multiplications). Fig. 4 shows our architecture for this part. To compute M_{sum} for one moduli, we use an RNS multiplication followed by a CSA tree to add all the 22 integers out of the RNS multiplication. Lastly, we perform a modulo reduction by q_i. We instantiate four of these in parallel (see Fig. 4) to calculate four results per clock cycle. This choice is a compromise between performance and area consumption, as adding more parallel multiplication and CSA tree will decrease the latency of this block. But it would also significantly increases the area consumption, i.e., adding an RNS multiplication increases the DSP count of design by 132.

The final reduction is the block which computes the RNS modular reduction. It takes $\overline{r}$ and $\overline{k}$ as inputs from the other blocks. It then uses those two values to compute $\forall i \in [1, n]$ $\overline{k} * Q \pmod{q_i}$ and $\overline{r} * p \pmod{q_i}$. As $\overline{k}$ can only take 23 values, we use 22 (one for each moduli) small tables (ROM) to store all the possible combinations of $\overline{k} * Q \pmod{q_i}$ and select the correct value accordingly. For $\overline{r} * p \pmod{q_i}$, we use 22 constant multiplier circuits to compute $\overline{r} * (p \mod q_i) \mod q_i$ (one circuit for one moduli). Then, we use one RNS addition unit to add $\overline{k} * Q \pmod{q_i}$ and $\overline{r} * p \pmod{q_i}$ together. The last step is to subtract, using an RNS subtraction unit, our previous result and the output of the M_{sum} block.

V. Experimental Results

In this section, we present experimental results of the first RNS-based hardware implementation of isogeny-based cryptography. As a case study, we implemented the full SIDH protocol in the Zynq Ultrascale+ ZCU102 FPGA with a performance-optimized implementation strategy in the Vivado 2019.1 tool suite. For the 503-bit prime `SIKEp503`, the implementation achieves 250 MHz clock frequency and consumes 151,009 LUTs (55%), 143,171 DFFs (26%), 1,056 DSPs (41%) in the FPGA. We computed the latency for `SIKEp503` through simulation. Alice's public-key generation takes 478,318 cycles (1.913 ms), and the shared key generation takes 467,695 cycles (1.87 ms) only. Bob's public-key generation takes 657,650 cycles (2.631 ms), and the shared key generation takes 547,793 cycles (2.191 ms) only.

TABLE I
PERFORMANCE COMPARISONS FOR THE MODULAR MULTIPLICATION

Work	**Freq.** *(MHz)*	**Lat. for 751-bit** (in cc/μs)	**Lat. for 503-bit** (in cc/μs)
[16]	109	176 / 2.147	90 / 0.532
[3]	193	101 / 0.523	-
[4]	167.4	69 / 0.412	-
[5]	294	90 / 0.306	-
[6]	182.3	54 / 0.296	-
Our Work	250	28 / 0.112	26 / 0.104

TABLE II
COMPARISON TO OTHER SIDH IMPLEMENTATIONS FOR `SIKEp503`

Work	**Freq.** *(MHz)*	**Total** (in μs)	**Area** (in LUTs/FFs/DSPs/BRAMS)
[16]	109	49.9	21,321 / 13,756 / 162 / 39
[3]	207	14.2	45,615 / 33,969 / 384 / 40
[17]	177.1	33.7	30,031 / 24,499 / 192 / 27
[4]	165.9	14.1	27,609 / 23,746 / 264 / 33.5
Our Work	250	8.6	151,009 / 143,171 / 1,056 / 0

Modular multiplication is the most time consuming finite field primitive in isogeny-based cryptography. The proposed RNS modular reduction unit for the prime `SIKEp503` uses 86,129 LUTs, 51,629 DFFs and 924 DSPs. Table I shows the timing comparisons between modular multipliers proposed in our work and previous works for the two primes `SIKEp751` and `SIKEp503`. We can see that our proposed modular multiplication is by far the fastest due to the use of both RNS and our new modular reduction technique. However, such a significant speed up comes at the cost of a higher resource usage due to parallel RNS arithmetic. In this work, we prioritized speed over resource consumption to make isogeny cryptography faster targeting high-end platforms e.g., servers.

In Table II, we compare our SIDH implementation for `SIKEp503` with other works in the literature. Our work is faster than other works in the literature, but we use more resources. In the RNS there is no data dependency between the coprime moduli. Hence, to fully benefit from this property, we instantiate every moduli in parallel in the hardware to achieve maximum parallel processing. The other key point is that most works in non-RNS representation use an architecture based on the Radix Montgomery multiplication approach for large integer modular reduction [3], [4], [16], [17]. This approach results in an architecture with a high performance to area ratio. However, for very-fast implementations, this approach becomes less efficient due to the data dependencies within the algorithm becoming the performance bottleneck. Meaning adding more multipliers will decrease the latency but at a diminishing return.

VI. DISCUSSION AND CONCLUSION

Several works exist in the literature that propose attacks SIKE implementations using side-channels [18], [19]. RNS arithmetic can be considered as a countermeasure against certain types of power analysis attacks because it behaves similarly to how arithmetic shares work in masking. However, side-channel attacks are still possible on RNS, particularly during the computation of the three-point ladder. In both isogeny cryptography and ECC, the biggest side-channel vulnerability is the three-point ladder algorithm, $P + [sk] * Q$ operation. [18] analysis leakage from this operation for its attack on SIKE/SIDH. The discussed countermeasure is adding randomness to the algorithm by using a special point of the starting elliptic curve, the infinite point $\mathcal{O}$. This countermeasure slightly increases the latency of the implementation.

In this paper, we propose a new modular reduction technique using RNS for large finite field arithmetic. We also present its high-performance hardware architecture and utilize it for implementing a hardware architecture for full SIDH protocol as a case study. The experimental results show that our proposed RNS modular reduction method outperforms the existing RNS Montgomery reduction methods. Also, its hardware implementation achieves very low latency. Similarly, the SIDH implementation presents the fastest performance in the literature. We hope that our work contributes to the use of RNS for large finite field arithmetic by tackling one of its main challenges, modular reduction. Exploring side-channel security of RNS-based implementations is left as future work.

REFERENCES

[1] D. Jao and L. D. Feo, “Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies,” *IACR Cryptol. ePrint Arch.*, p. 506, 2011. [Online]. Available: http://eprint.iacr.org/2011/506

[2] L. D. Feo, D. Kohel, A. Leroux, C. Petit, and B. Wesolowski, “Sqisign: Compact post-quantum signatures from quaternions and isogenies,” in *ASIACRYPT 2020, Daejeon, South Korea, December 7-11, 2020*. Springer, 2020, pp. 64–93.

[3] B. Koziel, R. Azarderakhsh, and M. M. Kermani, “A high-performance and scalable hardware architecture for isogeny-based cryptography,” *IEEE Trans. Computers*, vol. 67, no. 11, pp. 1594–1609, 2018.

[4] B. Koziel, A. E. Ackie, R. E. Khatib, R. Azarderakhsh, and M. M. Kermani, “Sike’d up: Fast hardware architectures for supersingular isogeny key encapsulation,” *IEEE Trans. Circuits Syst.*, vol. 67-I, no. 12, pp. 4842–4854, 2020.

[5] R. Elkhatib, R. Azarderakhsh, and M. M. Kermani, “Highly optimized montgomery multiplier for SIKE primes on FPGA,” in *ARITH 2020, Portland, OR, USA, June 7-10, 2020*. IEEE, 2020, pp. 64–71.

[6] M. H. Farzam, S. B. Sarmadi, H. Mosanaei-Boorani, and A. Alivand, “Hardware architecture for supersingular isogeny diffie-hellman and key encapsulation using a fast montgomery multiplier,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 68, no. 5, pp. 2042–2050, 2021.

[7] S. Asif, M. S. Hossain, Y. Kong, and W. Abdul, “A fully RNS based ECC processor,” *Integr.*, vol. 61, pp. 138–149, 2018.

[8] J. Bajard and L. Imbert, “A full RNS implementation of RSA,” *IEEE Trans. Computers*, vol. 53, no. 6, pp. 769–774, 2004.

[9] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, “CSIDH: an efficient post-quantum commutative group action,” in *ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6, 2018*. Springer, 2018, pp. 395–427.

[10] W. Castryck and T. Decru, “An efficient key recovery attack on sidh (preliminary version),” Cryptology ePrint Archive, Paper 2022/975, 2022.

[11] K. C. Posch and R. Posch, “Modulo reduction in residue number systems,” *IEEE Tran. Paral. Dist. Sys.*, vol. 6, no. 5, pp. 449–454, 1995.

[12] J. Bajard, L. Didier, and P. Kornerup, “An RNS montgomery modular multiplication algorithm,” *IEEE Trans. Computers*, vol. 47, no. 7, pp. 766–776, 1998.

[13] F. Gandino, F. Lamberti, G. Paravati, J. Bajard, and P. Montuschi, “An algorithmic and architectural study on montgomery exponentiation in RNS,” *IEEE Trans. Computers*, vol. 61, no. 8, pp. 1071–1083, 2012.

[14] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, “Cox-rower architecture for fast parallel montgomery multiplication,” in *EUROCRYPT 2000, Bruges, Belgium, May 14-18, 2000*. Springer, 2000, pp. 523–538.

[15] S. Kawamura, Y. Komano, H. Shimizu, and T. Yonemura, “RNS montgomery reduction algorithms using quadratic residuosity,” *J. Cryptogr. Eng.*, vol. 9, no. 4, pp. 313–331, 2019.

[16] P. M. C. Massolino, P. Longa, J. Renes, and L. Batina, “A compact and scalable hardware/software co-design of SIKE,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 245–271, 2020.

[17] B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao, “Post-quantum cryptography on FPGA based on isogenies on elliptic curves,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 64-I, no. 1, pp. 86–99, 2017.

[18] A. Genêt, N. L. de Guertechin, and N. Kaluerović, “Full key recovery side-channel attack against ephemeral sike on the cortex-m4,” Cryptology ePrint Archive, Paper 2021/858, 2021.

[19] B. Koziel, R. Azarderakhsh, and D. Jao, “Side-channel attacks on quantum-resistant supersingular isogeny diffie-hellman,” in *Selected Areas in Cryptography (SAC) 2017, Ottawa, ON, Canada, August 16-18, 2017*. Springer, 2017, pp. 64–81.