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. 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. SQISign [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 present 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 $\bar{Q}_i = \frac{Q}{q_i}$ 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 $O$. Two elliptic curves are isogenous if their order (number of points) is identical. The search version of the isogeny problem refers to the following: from two isogenous curves, $E_a$ and $E_b$, compute an isogeny $\phi$ 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, Costruck 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 \( B = \{ q_1, q_2, \ldots, q_{n-1}, q_n \} \). An integer \( a \) is represented in the RNS base \( B \) as the vector \( \vec{A} = ([a]_{q_1}, [a]_{q_2}, \ldots, [a]_{q_{n-1}}, [a]_{q_n}) \). From a given RNS representation \( \vec{A} \), we can obtain the integer \( a \) by applying the Chinese Remainder Theorem (CRT) as \( a = \sum_{i=1}^{n} a_i \cdot \tilde{Q}_i^{-1} \cdot \tilde{Q}_i \mod q_i \). With the application of RNS, a long integer computation \( a \mod Q \) 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.

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 \( \vec{A} \) and \( \vec{B} \) 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. 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 \( \vec{A} = (a_1, \ldots, a_n) \) be the RNS representation of \( a \) which is the result of a multiplication modulo \( Q \). Our goal here is to compute \( \vec{A} \mod p \). Starting from \( \vec{A} \), we obtain \( a \) using CRT as follows:

\[
\begin{align*}
    a &= \sum_{i=1}^{n} a_i \cdot \tilde{Q}_i^{-1} \cdot \tilde{Q}_i \mod q_i \\
    &= \sum_{i=1}^{n} v_i \cdot \tilde{Q}_i \mod q_i \tag{2}
\end{align*}
\]

We define \( M_{sum} = \sum_{i=1}^{n} a_i \cdot \tilde{Q}_i^{-1} \cdot \tilde{Q}_i \mod q_i \), then replace the reduction with a subtraction by \( r \cdot p \) where \( r \) is defined as \( r = \lfloor M_{sum}/p \rfloor \). The idea of replacing the \( \tilde{Q}_i \) by \( \tilde{Q}_i \mod p \) was first propose here [15] in order to compute a partial reduction for modular exponentiation. In this work we aim for a full modular reduction in RNS. Hence:

\[
\begin{align*}
    |a|_p &= \sum_{i=1}^{n} v_i \cdot \tilde{Q}_i \mod q_i \mod r \cdot p \tag{3}
\end{align*}
\]

We use a quotient \( r \) here to complete our reduction after Eq. 3. Here, \( \text{size}(r) = s + \text{size}(n) \) 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 \( Q \)).

\[
\begin{align*}
    |a|_p \mod q_i &= (M_{sum} - r \cdot p) \mod q_i \\
    &= \sum_{i=1}^{n} v_i \cdot \tilde{Q}_i \mod |q_i| - |k \cdot Q| \mod |q_i| - |r \cdot p| \mod |q_i|, q_i \tag{5}
\end{align*}
\]

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 \( \vec{A} \) (which is present in \( v_i \)) and others parameters. The constants \( (\tilde{Q}_i^{-1}, \tilde{Q}_i, Q, |q_i|, p) \) 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 $\bar{k}$:

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

The $\text{trunc}$ function takes in a bit length $l$, an integer $v_i$, and returns the $l$ most significant bits of the input integer. The parameters $\alpha$ 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 $\bar{k}$ will be equal to $k$.

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

$$\tau = \frac{1}{p} \sum_{i=1}^{n} v_i \cdot |\tilde{Q}_i|_p - |k \cdot Q|_p \quad (7)$$

with $|\tilde{Q}_i|_p = \text{trunc}(\tilde{Q}_i|_p, u)$ and $|k \cdot Q|_p = \text{trunc}(k \cdot Q|_p, u)$, $u$ 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 $2 \cdot s$ 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.

**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 $\bar{k}$ (Eq. 6) and $\tau$ (Eq. 7).

We have carefully chosen our parameters to always get the correct output for $\bar{k}$. For the approximation $\tau$ 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 $|k \cdot Q|_p$, 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 $\delta(r) = \|r - \tau\|$ using Eq. 7:

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

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

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

$$\delta(r) \leq \frac{1}{p} \sum_{i=1}^{n} v_i \cdot (2^{e-u} + 2^{e-u})$$

$$\leq 2^{e-u} \cdot \sum_{i=1}^{n} v_i + 1) = 2^{e-u} \cdot (\sum_{i=1}^{n} v_i + 1)$$

$$\leq 2^{e-u} \cdot (\sum_{i=1}^{n} 2^s + 1) = 2^{e-u} \cdot (n \cdot 2^s + 1)$$

So supposing a uniform distribution of values, our approach has a $2^{1-u} \cdot (n \cdot 2^s + 1)$ chance of not outputting the correct value of $r$. This is not problematic at all, as $\tau$ can only take two values, either $r$ or $r - 1$. When $\tau = r - 1$, 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 SIKe503 with the prime of size $e = 503$. We work with following the RNS parameters $s = 48$ (size of $q_i$), $u = 96 = 2 \cdot s$ 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

$$\delta(r) \leq 2^{1-u} \cdot (n \cdot 2^s + 1) \leq 2^{-42}.$$

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

In Alg. 1, Line 1 has one RNS vector multiplication requiring $n$ unit multiplications. Line 5 has $|u/s|$ multiplications in a for-loop and the total number becomes $|u/s| \cdot n$ 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 $\tau \cdot |p|_{q_i}$ in a for loop, thereby requiring total $n$ unit multiplications. Summing all, our method of reduction has a cost of $n^2 + (2 + |u/s|) \cdot n$ unit multiplications. It also uses mostly $s$ bit arithmetic, except in the computation $\tau$ where we need one $u$ bits subtraction and $|u/s - 1|$ addition. This method also has a small chance of computing a partial
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 $B = [31, 32, 33]$ 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 $B$. 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 $[25, 28, 28]$. 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 $Q \pmod{p}$ 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 high-level architecture, 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 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 $(e + 5) \times 2$ 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 $\overline{v} \leftarrow \overline{A} \ast \overline{Q}^{-1}$ (Line 1 of Alg. 1). As shown in Fig. 2, we use $22$ RNS modular multipliers in parallel for computing $\overline{A} \ast \overline{Q}^{-1}$. We split the remaining computations into four parts, two blocks to compute the two variables $\overline{k}$ and $\overline{\tau}$, 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 $\tau$:** We have expressed how we compute $\tau$ in Eq. 6 and Fig. 3 shows the high level diagram of our block. The first step here is the multiplication of $v_i \ast [\tilde{Q}_i]_p$ \(\forall i \in [1, 22]\). $[\tilde{Q}_i]_p$ 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($\tilde{k} \ast Q$ (mod $p$), 96). We compute $\tilde{k}$ parallel to this whole block in Fig. 2, its value becomes available before the trunc($\tilde{k} \ast Q$ (mod $p$), 96) computation. As $\tilde{k}$ can only take a value between [0, 22], we use a small table (ROM) to store all the possible combinations of trunc($\tilde{k} \ast Q$ (mod $p$), 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 use a 47-bit integer. We use a constant multiplier circuit with the constant tree which computes the RNS modular multiplication. It takes $\tau$ and $\tilde{k}$ as inputs from the other blocks. It then uses those two values to compute $\forall i \in [1, n] k \ast Q$ (mod $q_i$) and $\tau \ast p$ (mod $q_i$). As $\tilde{k}$ can only take 23 values, we use 22 (one for each moduli) small tables (ROM) to store all the possible combinations of $k \ast Q$ (mod $q_i$) and select the correct value accordingly. For $\tau \ast p$ (mod $q_i$), we use 22 constant multiplier circuits to compute $\tau \ast p$ (mod $q_i$) and select the correct value accordingly. For $\tau \ast p$ (mod $q_i$), we use one RNS addition unit to add $k \ast Q$ (mod $q_i$) and $\tau \ast p$ (mod $q_i$) 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.

**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
TABLE II

<table>
<thead>
<tr>
<th>Work</th>
<th>Freq. (MHz)</th>
<th>Total (in µs)</th>
<th>Area (in LUTs/FFs/DSPs/BRAMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[16]</td>
<td>109</td>
<td>49.9</td>
<td>21,321 / 13,756 / 162 / 39</td>
</tr>
<tr>
<td>[3]</td>
<td>207</td>
<td>14.2</td>
<td>45,615 / 33,969 / 384 / 40</td>
</tr>
<tr>
<td>[17]</td>
<td>177.1</td>
<td>33.7</td>
<td>30,031 / 24,499 / 192 / 27</td>
</tr>
<tr>
<td>Our Work</td>
<td>250</td>
<td>8.6</td>
<td>151,009 / 143,171 / 1,056 / 0</td>
</tr>
</tbody>
</table>

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. 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