Faster Pairing Coprocessor Architecture

Gavin Xiaoxu Yao[1], Junfeng Fan[2], Ray C.C. Cheung[1], and Ingrid Verbauwhede[2]

[1] Department of Electronic Engineering
City University of Hong Kong, Hong Kong SAR
`Gavin.Yao@student.cityu.edu.hk`, `R.Cheung@cityu.edu.hk`
[2] KU Leuven, ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{`Junfeng.Fan,Ingrid.Verbauwhede`}`@esat.kuleuven.be`

Abstract. In this paper, we present a high-speed pairing coprocessor using Residue Number System (RNS) which is intrinsically suitable for parallel computation. This work improves the design of Cheung *et al.* [11] using a carefully selected RNS base and an optimized pipeline design of the modular multiplier. As a result, the cycle count for a modular reduction has been halved. When combining with the lazy reduction, Karatsuba-like formulas and optimal pipeline scheduling, a 128-bit optimal ate pairing computation can be completed in less than 100,000 cycles. We prototype the design on a Xilinx Virtex-6 FPGA using 5237 slices and 64 DSPs; a 128-bit pairing is computed in 0.358 ms running at 230MHz. To the best of our knowledge, this implementation outperforms all reported hardware and software designs.

Keywords: Optimal pairing, Residue Number System (RNS), Field Programmable Gate Array (FPGA).

1 Introduction

Pairing-Based Cryptography (PBC) has been applied to provide efficient solutions to several long-standing problems in cryptography, such as three-way key exchanges [22], identity-based encryptions [9], identity-based signatures [10], and non-interactive zero-knowledge proof systems [19]. As cryptographic schemes based on pairings are introduced and investigated, the performance of pairing computations also receives increasing interest [1,2,7,11,13,14,16,18,20,23,30].

The pairing computation is relatively complex and slow compared with other popular public-key primitives such as Rivest-Shamir-Adleman (RSA) [34] or Elliptic Curve Cryptography (ECC) [25,27]. For pairings over ordinary curves defined over prime fields $\mathbb{F}_p$, the computation can be broken down into modular multiplications and additions in the underlying fields. For example, an optimal ate pairing with 128-bit security consists of around ten thousand modular multiplications [2]. Thus, a faster pairing coprocessor is essential, and an efficient modular multiplier is the key component to make this happen.

Due to the suitability for parallel implementation and the low cost for multiplications [31], Residue Number Systems (RNSs) have been introduced and studied for long integer modular multiplications [4, 24, 33, 36]. With area complexity of $O(n)$, the time complexity is $O(1)$ for a multiplication and $O(n)$ for a modular reduction, where n is the number of machine-words to represent the modulus p. Recently, [11] has proposed a novel parameter selection method to ensure further complexity decrease for RNS modular reduction.

Moreover, lazy reduction and Karatsuba-like formulas are introduced to the computation. These techniques were first deployed for pairing by Scott [35] and then generalised by Aranha *et al.* [2]. In short, lazy reduction performs one reduction for multiple multiplications, which is possible for expressions like $\sum AB$ in $\mathbb{F}_p$; Karatsuba-like formulas save multiplications in extension fields. As such, the number of modular reductions and multiplications decreases.

In this paper, we improve the design of Cheung *et al.* [11] with a higher throughput. We show that reducing the number of moduli in the RNS basis leads to a faster RNS reduction. Although the size of multipliers in each channel is much larger than that of [11], the maximum frequency is only 8% lower due to the increase of pipeline stages. We maximally explore parallelisms in RNS arithmetic and the pairing algorithm to remove pipeline bubbles. As a result, our implementation of 126-bit optimal ate pairing uses only 78×10^3 cycles, 45% less than that of the design in [11]. Our pairing processor, implemented on a Xilinx Virtex-6 FPGA, requires 0.338 ms to finish one 126-bit optimal ate pairing and 0.358 ms for a 128-bit one. To the best of our knowledge, this implementation outperforms all previous hardware and software designs.

The rest of the paper is organized as follows: Section 2 provides a recap on mathematical background. Section 3 emphasizes on the optimal parameter selection. We illustrate the architectural design and the scheduling on the proposed architecture in detail in Section 4 and 5, respectively. Section 6 gives the FPGA implementation results of the proposed architecture, and compares them with recent results from the literatures. Finally, Section 7 concludes this paper.

2 Background

2.1 Bilinear Pairing

A bilinear pairing is a non-degenerate map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups and $\mathbb{G}_T$ is a subgroup of a multiplicative group. The core property of map e is linearity in both components, which enables the construction of novel cryptographic protocols. Popular pairings such as Tate pairing [5], ate pairing [21], R-ate pairing [26], optimal pairing [37] choose $\mathbb{G}_1$ and $\mathbb{G}_2$ to be specific cyclic subgroups of $E(\mathbb{F}_{p^k})$, and $\mathbb{G}_T$ to be a subgroup of $\mathbb{F}^*_{p^k}$.

The selection of parameters has an essential impact on the security and the performance of a pairing computation, and not all elliptic curves are suitable. We refer to Freeman *et al.* [15] for a summary of known pairing-friendly curves. Among them, Barreto and Naehrig (BN) described a parameterized family of elliptic curves [6], and it is well-suited for computing asymmetric pairings. BN-curves are defined with $E : y^2 = x^3 + b, b \neq 0$ over $\mathbb{F}_p$, where $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and g, the order of E, is $36u^4 + 36u^3 + 18u^2 + 6u + 1$. Note that any $u \in \mathbb{Z}$ that generates prime p and g will suffice. BN-curves have embedding degree $k = 12$. Because of the limited space, we mainly focus on the discussion of the optimal ate pairing on BN-curves.

Let $E' : y^2 = x^3 + b/\zeta$ be a sextic twist of E with ζ not a cube nor a square in $\mathbb{F}_{p^2}$, and $E[g]$ be the subgroup of g-torsion points of E, then the optimal ate pairing is defined as [2,30]:

$$a_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$$

$$(Q, P) \to (f_{r,Q}(P) \cdot l_{[r]Q,\pi_p(Q)}(P) \cdot l_{[r]Q+\pi_p(Q),-\pi_p^2(Q)}(P))^{\frac{p^{12}-1}{g}}$$

where $r = 6u + 2$. The group $\mathbb{G}_1 = E[g] \bigcap \mathrm{Ker}(\pi_p - [1]) = E(\mathbb{F}_p)[g]$ and $\mathbb{G}_2$ is the preimage $E'(\mathbb{F}_{p^2})[g]$ of $E[g] \bigcap \mathrm{Ker}(\pi_p - [p]) \subseteq E(\mathbb{F}_{p^{12}})[g]$ under the twisting isomorphism $\psi : E' \to E$. The group $\mathbb{G}_T$ is the subgroup of g-th roots of unity $\mu_g \subset \mathbb{F}^*_{p^{12}}$. The map $\pi_p : E \to E$ is the Frobenius endomorphism $\pi_p(x, y) = (x^p, y^p)$, and $f_{r,Q}(P)$ is a normalized function with divisor $(f_{r,Q}) = r(Q) - ([r]Q) - (r-1)(\mathcal{O})$. The line function, $l_{Q_1,Q_2}(P)$, is the line arising in the addition of Q_1 and Q_2 evaluated at point P.

Miller [28] proposed an algorithm that constructs $f_{r,Q}$ in stages by using double-and-add method. When u is selected as a negative integer, the corresponding Miller algorithm is shown in Algorithm 1 [2].

Algorithm 1. Optimal ate pairing on BN curves for $u < 0$ [2]

```
Require: $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, r = |6u + 2| = \sum_{i=0}^{\lfloor \log_2 r \rfloor} r_i 2^i$, where $u < 0$
Ensure: $a_{opt}(Q, P)$
 1: $T \leftarrow Q, f \leftarrow 1$
 2: for $i = \lfloor \log_2 r \rfloor - 1$ downto 0 do
 3:     $f \leftarrow f^2 \cdot l_{T,T}(P), T \leftarrow 2T$
 4:     if $r_i = 1$ then
 5:         $f \leftarrow f \cdot l_{T,Q}(P), T \leftarrow T + Q$
 6:     end if
 7: end for
 8: $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p^2(Q)$
 9: $T \leftarrow -T, f \leftarrow f^{p^6}$
10: $f \leftarrow f \cdot l_{T,Q_1}(P), T \leftarrow T + Q_1$
11: $f \leftarrow f \cdot l_{T,-Q_2}(P), T \leftarrow T - Q_2$
12: $f \leftarrow f^{(p^6-1)(p^2+1)(p^4-p^2+1)/g}$
13: return $f$
```

2.2 Residue Number System

A Residue Number System (RNS) uses a set of smaller integers to represent a large integer. An RNS is defined by a set of n coprime integer constants, $\mathfrak{B} = \{b_1, b_2, \ldots, b_n\}$. The set $\mathfrak{B}$ is also known as a *base*, and the element b_i, $1 \leqslant i \leqslant n$, is called *RNS modulus*, and each modulus forms an RNS *channel.* Let $M_{\mathfrak{B}} = \prod_{i=1}^{n} b_i$. Let $|a|_b$ be a modulo b, then any integer X, $0 \leqslant X < M_{\mathfrak{B}}$, can be uniquely represented as a set of smaller integers: $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$, where $x_i = |X|_{b_i}$, $1 \leqslant i \leqslant n$. Similar to the radix-$2^w$ representation, we also call x_i a *digit* of X. The original value of X can be restored from $\{X\}_{\mathfrak{B}}$ using the Chinese Remainder Theorem (CRT):

$$X = \left| \sum_{i=1}^{n} \left| x_i \cdot B_i^{-1} \right|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}}, \text{ where } B_i = \frac{M_{\mathfrak{B}}}{b_i} = \prod_{j=1, j \neq i}^{n} b_j, \ 1 \leqslant i \leqslant n. \tag{1}$$

Using RNS, arithmetic operations in $\mathbb{Z}/M_{\mathfrak{B}}\mathbb{Z}$ can be efficiently performed. Consider two integers X, Y and their RNS representations $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$ and $\{Y\}_{\mathfrak{B}} = \{y_1, y_2, \ldots, y_n\}$, then

$$\{|X \odot Y|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}} = \{|x_1 \odot y_1|_{b_1}, \ldots, |x_n \odot y_n|_{b_n}\}.$$

for $\odot \in \{+, -, \times, /\}$. The division is available only if Y is coprime with $M_{\mathfrak{B}}$, i.e. the multiplicative inverse of Y exists and is calculated in $\mathfrak{B}$.

Note that for all the basic operations $(+, -, \times, /)$, computations between x_i and y_i have no dependency on other digits, which largely simplifies the parallelization of the operations. Besides, the complexity of a multiplication in RNS is $O(n)$, while it is $O(n^2)$ using textbook arithmetics.

For every operation, there is an implicit channel reduction to bring the result in the range $[0, b_i)$. In order to accelerate the channel reduction, pseudo-Mersenne numbers of the form $b_i = 2^w - d_i$, where $d_i < 2^{\lfloor \frac{w}{2} \rfloor}$, are commonly chosen as moduli. To compute $x < 2^{2w}$ modulo b_i, one first performs the following step twice:

$$x \leftarrow (x \bmod 2^w) + d_i \cdot (x \text{ div } 2^w) \tag{2}$$

Then, x will be in the range of $[0, 2^{w+1})$, and after a conditional subtraction, one finishes the residue calculation. If the Hamming weight of d_i is small, multiplications by d_i can also be replaced by a few additions.

2.3 RNS Montgomery Algorithm and Faster Base Extension

Most cryptographic applications, such as pairings, require the operations modulo a prime, which prevents a direct utilization of RNS. This problem can be avoided by combining RNS representation and the Montgomery reduction algorithm [29].

Algorithm 2 shows the Montgomery modular multiplication algorithm without conditional subtraction in RNS context. A new base, $\mathfrak{C} = \{c_1, c_2, \ldots, c_n\}$, where $M_{\mathfrak{C}} = \prod_{i=1}^{n} c_i$ is coprime with $M_{\mathfrak{B}}$, is introduced to perform the division, and

```
Algorithm 2. RNS Montgomery Modular Multiplication [24]
Require: RNS bases $\mathfrak{B}$ and $\mathfrak{C}$ with $M_{\mathfrak{B}}, M_{\mathfrak{C}} > 2P$
Require: $P$, $M_{\mathfrak{B}}$, $M_{\mathfrak{C}}$ are pairwise coprime
Require: $\{X\}_{\mathfrak{B}}$, $\{X\}_{\mathfrak{C}}$, $\{Y\}_{\mathfrak{B}}$, $\{Y\}_{\mathfrak{C}}$ with $X, Y < 2P$
  Precompute: $\{P'\}_{\mathfrak{B}} \leftarrow \{|-P^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$
  Precompute: $\{M'\}_{\mathfrak{C}} \leftarrow \{|M_{\mathfrak{B}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$ and $\{P\}_{\mathfrak{C}}$
Ensure: $\{U\}_{\mathfrak{B}}$, $\{U\}_{\mathfrak{C}}$ s.t. $|U|_P = |XYM_{\mathfrak{B}}^{-1}|_P$, $U < 2P$

                    in $\mathfrak{B}$                 in $\mathfrak{C}$
 1: $\{T\}_{\mathfrak{B}} \leftarrow \{X\}_{\mathfrak{B}} \times \{Y\}_{\mathfrak{B}}$, $\{T\}_{\mathfrak{C}} \leftarrow \{X\}_{\mathfrak{C}} \times \{Y\}_{\mathfrak{C}}$
 2: $\{Q\}_{\mathfrak{B}} \leftarrow \{T\}_{\mathfrak{B}} \times \{P'\}_{\mathfrak{B}}$
 3:        $\{Q\}_{\mathfrak{B}} \xrightarrow{Base\ Extension\ 1} \{Q\}_{\mathfrak{C}}$
 4:        $\{U\}_{\mathfrak{C}} \leftarrow \big(\{T\}_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \times \{P\}_{\mathfrak{C}}\big) \times \{M'\}_{\mathfrak{C}}$
 5:        $\{U\}_{\mathfrak{B}} \xleftarrow{Base\ Extension\ 2} \{U\}_{\mathfrak{C}}$
 6:           return $\{U\}_{\mathfrak{B}}$ and $\{U\}_{\mathfrak{C}}$
```

all the moduli from both $\mathfrak{B}$ and $\mathfrak{C}$ are pairwise coprime as $M_{\mathfrak{B}}$ and $M_{\mathfrak{C}}$ are coprime. The overhead is two Base Extensions (BEs) required in Algorithm 2.

BE is to compute $\{X\}_{\mathfrak{C}} = \{x'_1, x'_2, \ldots, x'_n\}$ given $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$. We choose CRT method, specifically, the parallelizable Posch-Posch Method [24,32]. Given $\{X\}_{\mathfrak{B}}$, for (1), there must exist a certain integer $\lambda < n$ such that:

$$X = \left| \sum_{i=1}^{n} \left| x_i \cdot B_i^{-1} \right|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}} = \left| \sum_{i=1}^{n} \xi_i \cdot B_i \right|_{M_{\mathfrak{B}}} = \sum_{i=1}^{n} \xi_i \cdot B_i - \lambda \cdot M_{\mathfrak{B}} \tag{3}$$

where $\xi_i = \left| x_i \cdot B_i^{-1} \right|_{b_i}$, $1 \leqslant i \leqslant n$, and λ can be calculated by:

$$\lambda = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i \cdot B_i}{M_{\mathfrak{B}}} \right\rfloor = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i}{b_i} \right\rfloor \tag{4}$$

In [24], ξ_i/b_i is further approximated by $\xi_i/2^w$ as b_i is of the form $2^w - d_i, d_i > 0$. Once λ is obtained, $\{X\}_{\mathfrak{C}}$ can be computed by a matrix multiplication and channel reductions:

$$\left(x''_1, \ldots, x''_n\right)^T := \begin{pmatrix} |B_1|_{c_1} & \cdots & |B_n|_{c_1} \\ \vdots & \ddots & \vdots \\ |B_1|_{c_n} & \cdots & |B_n|_{c_n} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_n \end{pmatrix} - \lambda \begin{pmatrix} |M_{\mathfrak{B}}|_{c_1} \\ \vdots \\ |M_{\mathfrak{B}}|_{c_n} \end{pmatrix} \tag{5}$$

$$\{X\}_{\mathfrak{C}} \equiv \{x'_1, \ldots, x'_n\} := \{|x''_1|_{c_1}, \ldots, |x''_n|_{c_n}\} \tag{6}$$

Note that the elements in the matrix, $|B_i|_{c_j}$, $1 \leqslant i, j \leqslant n$, are constants and are determined as $|B_i|_{c_j} = \left| \prod_{k=1, k\neq i}^{n} b_k \right|_{c_j} = \left| \prod_{k=1, k\neq i}^{n} (b_k - c_j) \right|_{c_j}$. In [11], it shows that the complexity of base extension can be reduced if the moduli in the two bases are close to each other. Define $\tilde{B}_{i,j} := \prod_{k=1,k\neq i}^{n}(b_k - c_j)$. $|B_i|_{c_j}$ or $\tilde{B}_{i,j}$ makes no difference to the results but $\tilde{B}_{i,j}$ can be much smaller when $b_k - c_j$ and n are small. We denote $v < w$ as the maximal bitlength of $\tilde{B}_{i,j}$. Now the $w \times w$ multiplications are substituted by the $v \times w$ multiplications.

Clearly, we need n $w\times w$-bit multiplications to calculate ξ_i and n^2 $v\times w$-bit multiplications for all $\sum_{i=1}^{n} \xi_i \cdot |B_i|_{c_j}, 1 \leqslant j \leqslant n$. In total, each reduction uses $4n$ digit multiplications and $2n^2$ $v\times w$-bit multiplications. Even though faster BE reduces the complexity, RNS reductions cost more than multiplications. Recall that a multiplication only takes $2n$ digit multiplications in Algorithm 2. Fortunately, lazy reduction is commonly used to reduce the number of the expensive modular reductions.

3 Parameter Selection

3.1 Pairing Parameter Selection

As stated in Section 2, we choose optimal ate pairing and BN-curve. Specifically, in order to achieve 128-bit security level, we choose $u = -(2^{63}+2^{22}+2^{18}+2^{7}+1)$ (p is 258-bit) as that in [11]. Also for comparison, we consider $u = -(2^{62}+2^{55}+1)$ (p is 254-bit) which is used in [2, 11] and achieve 126-bit security level. We also deploy the same tower extension field as in [2]:

- $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, where $\beta = -1$;
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[W]/(W^6 - \zeta)$, where $\zeta = 1 + i$.

3.2 RNS Parameter Selection

As p is 258-bit, $n \cdot w$ should be greater than 258 to provide sufficient operating range. In [11], the authors have shown that it takes two cycles for a multiplication and $2 \cdot n/(\lceil w/v \rceil) + 4$ cycles for a modular reduction, where v is basically determined by n. The authors choose $n = 8$ in [11], so that $w = 33, v = 25$. As a result, a reduction takes 12 cycles, and 25×18-bit multipliers are in demand. In this section, we show that if $n = 4$, $w = 67$, v will be < 18, hence a reduction will only take six cycles; 18×18 multipliers are already competent.

The selected bases are chosen as follows: ($w = 67$)

$$\mathfrak{B} = \{2^w - 1, 2^w - 7, 2^w - 9, 2^w - 15\},$$

$$\mathfrak{C} = \{2^w - 0, 2^w - 3, 2^w - 5, 2^w - 31\}.$$

We use $\mathbb{L}_{\tilde{B}}$ and $\mathbb{L}_{\tilde{C}}$ to show the bit-length of all $\tilde{B}_{i,j}$ and $\tilde{C}_{i,j}$ in the matrix form, respectively.

$$\mathbb{L}_{\tilde{B}} = \begin{pmatrix} 10 & 8 & 7 & 6 \\ 9 & 8 & 7 & 6 \\ 7 & 8 & 7 & 6 \\ 14 & 14 & 14 & 14 \end{pmatrix}, \ \mathbb{L}_{\tilde{C}} = \begin{pmatrix} 8 & 7 & 6 & 4 \\ 8 & 9 & 10 & 6 \\ 10 & 10 & 11 & 8 \\ 11 & 12 & 12 & 11 \end{pmatrix}.$$

The selected bases have the following features:

- All $\tilde{B}_{i,j}$ and $\tilde{C}_{i,j}$ are less than 18-bit. The largest element in $\tilde{B}_{i,j}$ and $\tilde{C}_{i,j}$ is 14-bit excluding the sign bit.
- The Hamming weights of all elements are less than 3 in non-adjacent form (NAF). Therefore, the channel reduction can be performed efficiently.
- b_i and c_i have the same NAF representation. For instance, b_3, $2^w - 15$, and c_3, $2^w - 31$ have the representation $2^w - 2^k + 1$, where $k = 4, 5$ respectively.
- All elements are equal to or less than 2^{67}, hence, after an addition or subtraction, the absolute value of the operand is less than 2^{68}, and can be represented by 69 bits including the sign bit.

The advantages of using the above features will be elaborated in the next section.

4 Architectural Design and Finite Field Arithmetics

4.1 The Controller and the Cox-Rower Architecture

The top level architecture is depicted by Fig. 1(a). The coprocessor can be divided into two major parts: the controller and the Arithmetic Logic Unit (ALU). We use an efficient and flexible micro-coded sequencer as the controller. By doing so, the controller maintains relatively small area, the ability to control accurately, and the flexibility for different curve and pairing operation. In this paper, we perform an optimal pairing computation on BN curves, however the proposed hardware architecture is capable of performing other pairings when the characteristic of the underlying field is less than 260 bits.

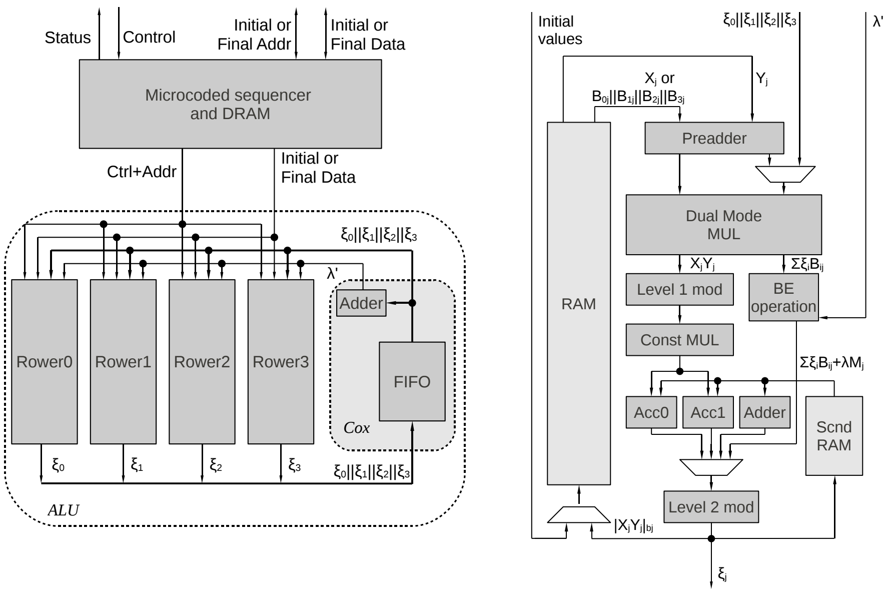
The ALU design is modified from that in [24]. We still call it the Cox-Rower architecture. Each rower performs operations in one channel of $\mathfrak{B}$ and one of $\mathfrak{C}$. Therefore, we have four rowers as $n = 4$. As described in Section 2, most operations are handled inside the channels independently, and the only step which requires communication between rowers is the ξ distribution in BE. We use a shared memory to redistribute the ξ values. Different from the original design [24], all ξ values for one BE are used at the same cycle, and hence, the ξ registers turn into a FIFO. The adder in the cox computes the value of λ for BE operation (4).

4.2 The Rower Design and Finite Field Arithmetics

We adjust the rower design to perform pairing computation more efficiently. The j-th rower architecture is shown in Fig. 1(b). For simplicity, we do not depict the control signal for multiplexers in the figures.

Dual Mode Multiplier

Fig. 2(a) depicts the dual mode multiplier in detail. It contains four 69×18-bit signed multipliers, and two different addition logics. One addition logic is for the 69×69-bit product, and the other is for the summation of four 69×18 products.



(a) Top level architecture with Cox-Rower ALU (b) The j-th rower design in the Cox-Rower architecture

Fig. 1. The Cox-Rower architecture

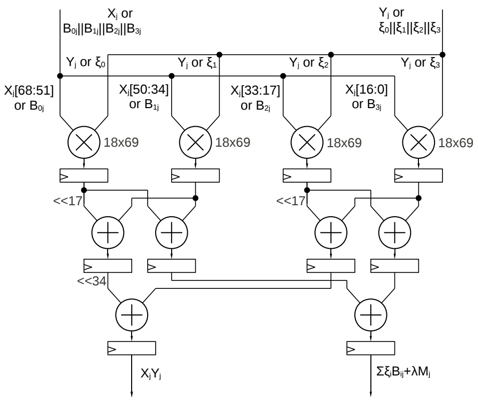
The sum of four 69×18 products is to perform one row of the matrix multiplication (5), since ξ_i is at most 69-bit and $\tilde{B}_{i,j}$ or $\tilde{C}_{i,j}$ is at most 18-bit. Therefore, a matrix multiplication only takes one cycle. Apart from the matrix multiplication, all the other multiplications are using full-length 69×69-bit signed representation. The four partial products are added up after the corresponding shift.

Preadders and Accumulators

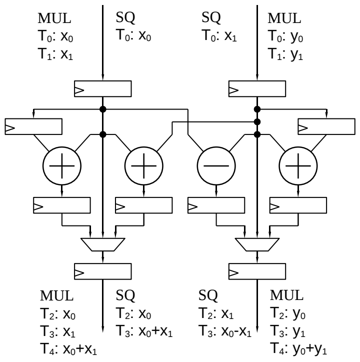
Since the operation in $\mathbb{F}_{p^{12}}$ can be broken down to $\mathbb{F}_{p^2}$, the operation in $\mathbb{F}_{p^2}$ is the fundamental arithmetic for pairing. The preadders and the accumulators together provide fast $\mathbb{F}_{p^2}$ operation with Karatsuba-like formulas embedded. The squaring and multiplication in $\mathbb{F}_{p^2}$ can be written as follows:

$$\begin{aligned} z_0 + z_1 i \leftarrow & \ (x_0 + x_1 i)^2 \\ = & \ x_0^2 - x_1^2 + 2x_0x_1 i \\ = & \ (x_0 + x_1)(x_0 - x_1) + 2x_0x_1 i \end{aligned} \tag{7}$$

$$\begin{aligned} z_0 + z_1 i \leftarrow & \ (x_0 + x_1 i)(y_0 + y_1 i) \\ = & \ x_0y_0 - x_1y_1 + (x_0y_1 + x_1y_0)i \\ = & \ x_0y_0 - x_1y_1 + ((x_0 + x_1)(y_0 + y_1) - x_0y_0 - x_1y_1)i \end{aligned} \tag{8}$$



(a) Dual-mode multiplier design



(b) Preadder design in each rower

Fig. 2. Architectural design of the components in a rower

When performing multiplication or squaring in $\mathbb{F}_{p^{12}}$, there are also operations as follows involved:

$$\begin{aligned} z_0 + z_1 i &\leftarrow (x_0 + x_1 i)^2 \zeta \\ &= x_0^2 - x_1^2 - 2x_0x_1 + (x_0^2 - x_1^2 + 2x_0x_1)i \\ &= (x_0 + x_1)(x_0 - x_1) - 2x_0x_1 + ((x_0 + x_1)(x_0 - x_1) + 2x_0x_1)i \end{aligned} \tag{9}$$

$$\begin{aligned} z_0 + z_1 i &\leftarrow (x_0 + x_1 i)(y_0 + y_1 i)\zeta \\ &= x_0y_0 - x_1y_1 - x_0y_1 - x_1y_0 + (x_0y_0 - x_1y_1 + x_0y_1 + x_1y_0)i \\ &= 2x_0y_0 - (x_0 + x_1)(y_0 + y_1) + ((x_0 + x_1)(y_0 + y_1) - 2x_1y_1)i \end{aligned} \tag{10}$$

The additions before multiplications are performed by the preadders and the ones after are done by accumulators. Fig. 2(b) shows the pipelined design of the preadders. There are only 2 patterns for $\mathbb{F}_{p^2}$ preaddition operations: squaring (7)/(9), and multiplication (8)/(10). The input and output sequences for squaring and multiplication are also provided in Fig. 2(b). We employ 2 accumulators in Fig. 1(b) and compute both z_0 and z_1 at the same time, because the same products are used for both z_0 and z_1 in (8), (9) and (10). As there are small constant multiplications in the algorithm, we also integrate a constant multiplier in the pipeline.

Channel Reducer

When performing the channel reduction, the multiplication by b_i or c_i in (2) is achieved by shifts and additions, as b_i and c_i are of small Hamming weight. Also as b_i and c_i are of the same NAF representations, the 2 channel reducers can be integrated into one with a signal to control the shift. Using Rower 3 as an example, since $d_3 = 2^k - 1, k = 4, 5$, (2) is performed by one subtraction and one addition. To reduce modulo 2^w and $2^w - 1$, the arithmetic is even simpler and the design occupies less area.

After the first execution of (2), the bit-length of x will be less than or equal to $\lceil \log_2(d_i + 1) \rceil + w$, and it is only a few bits more than w as the selected d_i is very small. We choose to put the accumulators here, because compared to accumulating immediately after multiplication, the bit-length of the accumulators is shortened by almost a factor of 2. (Another choice is to put the accumulators after the second execution of (2), which needs another channel reduction for the accumulated results.) As long as the accumulated result x is smaller than $2^{2w - \lceil \log_2(d_i+1) \rceil}$ (which is always the case), the second execution of (2) can bring x less than 2^{w+1}. Therefore, the channel reduction is divided into Level 1, which performs the first execution, and Level 2, which performs the second reduction and result correction.

Other Components
Each rower also has a 3-port RAM, one write port and two read ports, so that the RAM can provide two operands at the same cycle. There is an adder and a secondary RAM (sncd RAM) involved. The secondary RAM stores the values for addition and the initial values for accumulation. For instance, the second RAM stores the value of $\{T\}_{\mathfrak{C}}$ generated in Step 1 of Algorithm 2, and this value is sent to accumulators as initial value in Step 4. Therefore, the addition and the multiplication are performed in parallel. The adder takes the operation which cannot be integrated in accumulation efficiently. The BE operation module computes $\lambda \cdot M_j$ and adds it to the sum generated by the dual mode multiplier.

4.3 Cycle Count of Finite Field Operations

Since all the other operations (namely, preaddition, accumulation, channel reduction) are hidden in the pipeline, the cycle count for each operation is the cycle used in the dual mode multiplier. For one modular reduction, there are one multiplication in $\mathfrak{B}$, three multiplications in $\mathfrak{C}$ and two matrix multiplications. As one matrix multiplication only takes 1 cycle, it takes 6 cycles to perform one reduction. Essentially, at least k reductions are required for a multiplication in $\mathbb{F}_{p^k}$, as the result has k coefficients. Excluding the reduction, one multiplication or squaring in $\mathbb{F}_p$ takes 2 cycles, one squaring and one multiplication in $\mathbb{F}_{p^2}$ take 4 and 6 cycles, respectively. One squaring in $\mathbb{F}_{p^{12}}$ is equivalent to 6 $\mathbb{F}_{p^2}$ squarings, 15 $\mathbb{F}_{p^2}$ multiplications and 12 reductions, while one normal multiplication in $\mathbb{F}_{p^{12}}$ is equivalent to 36 multiplications and 12 reductions. The cycle count provided in this section is under the condition that all the pipeline stages are filled. For the latency and the pipeline bubbles, readers can refer to Section 5 for more information.

5 Operation Scheduling

5.1 Optimal Pipeline Scheduling for RNS Montgomery Algorithm

While the pairing algorithm is more complex than ECC or RSA, it also contains more parallelisms to be exploited. Utilizing these parallelisms, pipeline structure is a popular technique to improve throughput with negligible latency overhead. Typically, the pipeline depth is equal to or less than the level of parallelism, otherwise, there will be pipeline bubbles introduced. On the other hand, adding more pipeline stages can achieve a higher frequency. Hence, the throughput of the implementation might still be higher even if new pipeline bubbles are introduced.

Before implementing Algorithm 2 using the proposed ALU, we first apply the following optimizations on the algorithm:

- The multiplication by $\{M'\}_{\mathfrak{C}}$ is distributed to $\{T\}_{\mathfrak{C}}$ and $\{QP\}_{\mathfrak{C}}$ in Step 4.
- We directly compute $\{\xi\}_{\mathfrak{C}}$ for BE2, and then compute $\{U\}_{\mathfrak{C}}$ by $\{\xi\}_{\mathfrak{C}} \times \{C\}_{\mathfrak{C}}$.
- We pre-compute the products of the constant multiplicands, i.e. $\{P'B^{-1}\}_{\mathfrak{B}}$, $\{M'C^{-1}\}_{\mathfrak{C}}$ and $\{PM'C^{-1}\}_{\mathfrak{C}}$, which reduces the number of operations.
- We use the computation of $\{T\}_{\mathfrak{C}}$, $\{R\}_{\mathfrak{C}} = \{T\}_{\mathfrak{C}} \times \{M'C^{-1}\}_{\mathfrak{C}}$, and $\{U\}_{\mathfrak{C}}$ to fill in the idle state.

Let the pipeline depth of the rower be τ, and assume that there are ρ modular multiplications which can be pipelined. Also, as the ξ delivery is a shared operation between rowers, let it take ϵ cycles. We generalize η, the minimal number of cycles to execute these ρ modular multiplications, as follows without detailed elaboration due to paper length:

$$\eta = \begin{cases} 8\rho, & \tau \leqslant \rho \\ 2\tau + 6\rho, & \rho < \tau \leqslant 2\rho - \epsilon \\ 4\tau + 2\rho + 2\epsilon, & 2\rho - \epsilon < \tau \leqslant 2\rho \\ 5\tau + 2\epsilon, & \tau > 2\rho \end{cases} \tag{11}$$

As each modular multiplication takes 8 cycles, the pipeline occupation rate is given by $\frac{8\rho}{\eta} \times 100\%$, and the number of pipeline bubbles is $\eta - 8\rho$. Typically, ϵ is a very small number. Therefore, the pipeline occupation rate will be over 80% if $\tau \leqslant 2\rho$. In other words, if there are ρ concurrent reductions can be performed, the number of pipeline stages can be up to 2ρ without introducing a lot of idle cycles. Furthermore, if there is no dependency, one can use the multiplications in the next pipeline round to fill in the idle states in the current round.

5.2 Scheduling of the Miller Loop

We examine the data dependency of Miller algorithm for the optimal ate pairing on BN curve. The explict formulas, the pipeline grouping, and the cycle count are provided by Table 1. The number of pipelined inputs, ρ, is the number of reductions at the same pipeline group. In fact, $\tau = 18$ for our FPGA prototype. Using the scheduling shown in Table 1 for a single pairing, it guarantees $\tau < 2\rho$,

Table 1. Pipeline scheduling and operation counts of Miller loop

Cond-ition	Step	Pipeline group and Formulas	ρ for Single Pairing	Cycle Count: 3 Pair-ings/3	Cycle Count: Single Pairing
$r_i = 0$	1	$A = y_1^2$, $B = 3b'z_1^2$, $C = 2x_1y_1$ $D = 3x_1^2$, $E = 2y_1z_1, f_{0,1,2} = (f^2)_{0,1,2}$	16	180	180
	2	$x_3 = (A - 3B)C$, $y_3 = A^2 + 6AB - 3B^2$, $z_3 = 4AE$ $l_3 = A - B$, $l_1 = \overline{x_P}D$, $l_0 = y_PE, f_{3,4,5} = (f^2)_{3,4,5}$	16	186	190
	3	$f = f \cdot l$ $x_1 = x_3, y_1 = y_3, z_1 = z_3$	12	180	180
$r_i = 1$	1	$A = y_1^2$, $B = 3b'z_1^2$, $C = 2x_1y_1$ $D = 3x_1^2$, $E = 2y_1z_1, f_2 = (f^2)_2$	12	116	116
	2	$x_3 = (A - 3B)C$, $y_3 = A^2 + 6AB - 3B^2$, $z_3 = 4AE$ $l_3 = A - B$, $l_1 = \overline{x_P}D$, $l_0 = y_PE, f_{3,4,5} = (f^2)_{3,4,5}$ $x_1 = x_3, y_1 = y_3, z_1 = z_3$	16	186	190
	3	$A = y_1 - y_Qz_1$, $B = x_1 - x_Qz_1$ $f = f \cdot l$	16	224	228
	4	$C = A^2$, $D = B^2$ $l_3 = y_QB - x_QA$, $l_1 = x_PA$, $l_0 = \overline{y_P}B$	10	88	104
	5	$E = BD$, $C = BD + z_1C - 2x_1D$, $D = BD + z_1C - 2x_1D - x_1D$, $f = f \cdot l$	18	240	240
	6	$x_3 = BC$, $y_3 = -AD - y_1E$, $z_3 = z_1E$ $x_1 = x_3, y_1 = y_3, z_1 = z_3$, $f_{0,1} = (f^2)_{0,1}$	10	122	122

and hence the pipeline occupation rate is over 80%. Moreover, we use multiplications in the next group to fill in the idle state in the current, so that there is only 6 idle cycle when $r_i = 0$ (only 1% to the total cycles). For $r_i = 1$, there are 26 idle cycles. However, as the r is chosen with small Hamming weight, $r_i = 1$ rarely happens, and the pipeline occupation rate is over 98% for Miller loop.

To get rid of pipeline bubbles, one possible way is to perform multiple concurrent pairings, which happens to certain protocols [12] or the operation on a server side. We use Block RAM on FPGA to serve as the Rower memory, and even with the minimal configuration, it can store the intermediate values for 3 pairing computations with the same public parameter set. Therefore, our design is naturally suitable for 3 pairing computations at the same time.

5.3 Scheduling of the Final Steps

The final steps of an optimal pairing include 2 final additions (Step 10, 11 in Algorithm 1) and a final exponentiation (Step 12 in Algorithm 1). The operation count is given in Table 2. The formulas for final additions are the same with the point addition formulas in the Miller loop. The Frobenius endomorphism of Q_1, Q_2 computation is also included in the cycle count.

For the final exponentiation, the power $(p^{12} - 1/g)$ is factored into three small exponents: $(p^6 - 1)$, $(p^2 + 1)$, and $(p^4 - p^2 + 1)/g$. To compute $(p^6 - 1)$, it requires an inversion. The formulas from [35] is utilized to transform the inversion in $\mathbb{F}_{p^{12}}$

Table 2. Cycle counts of final steps for the pairing computation

Step	3 pair-ings/3	Occu. Rate	Single Pair.	Occu. Rate	Step	3 pair-ings/3	Occu. Rate	Single Pair.	Occu. Rate
1st final add	502	99.2%	582	87.6%	p^2+1	356	100%	398	89.4%
2nd final add	284	98.6%	394	74.1%	mul in hardexp	288	100%	300	96.0%
p^6-1, before inv	237.6	86.9%	431	49.6%	sq in hardexp	138	100%	150	92.0%
single sq in inv	28.3	42.4%	79	15.2%	exp to p, $3p$	90	100%	106	84.9%
p^6-1, after inv	265.3	94.2%	374	69.0%	exp to $2p$	68	100%	98	69.4%

Table 3. Logic Utilizations of Virtex-6 XC6VLX240T-1

Logic Utilization	# DSPs	# LUTs	# Reg.	# Occupied Slices	# 18Kb BRAMs
Used	64	18,794	21,505	5,237	41
Available	768	150,720	301,440	37,680	832
Utilization	8%	12%	7%	13%	5%

to an inversion in $\mathbb{F}_p$. To perform this inversion in $\mathbb{F}_p$, we employ Fermat's little theorem, i.e. $d^{-1} = d^{p-2} \bmod p$ if p is prime, and exponentiation with LSB-first as [11]. The problem with this exponentiation in $\mathbb{F}_p$ is that the computation cannot be pipelined. Indeed, only one stage is taken out of all pipeline stages, which causes low pipeline occupation rate and a huge waste. We use the same formulas as [11] for the exponentiation to $(p^4 - p^2 + 1)/g$, also known as *hard exponentiation.* We also use cyclotomic subgroup structure to accelerate squaring $\mathbb{F}_{p^{12}}$ [2,11]. Finally, in one hard exponentiation, 3 exponentiations to u take most of the time. Besides, there are 6 exponentiations to p or $3p$, 1 exponentiation to $2p$, 13 multiplications, and 4 squarings. We also consider that potentially 3 pairings are computed at the same time.

6 Implementation and Comparisons

6.1 Implementation

The pairing coprocessor is implemented on a Xilinx Virtex-6 XC6VLX240T-1 FPGA, which embeds 25×18 DSP slices (in fact, 18×18 multipliers are sufficient for our design). As there are 4 rowers and each rower contains 16 DSP slices, the total number of used DSP is 64. Data RAMs used in each Rower are implemented with 4 block RAMs. The block RAMs (BRAMs) also serve as the microcode sequencer. We have designed an 18-depth carefully tuned pipeline structure, which ensures the coprocessor to operate at 230MHz (with timing constrains of 210MHz, the coprocessor only occupies 3879 slices). Note that the logic utilization of DSPs and Slices is quite balanced (all around 10%), i.e. given a bounded area, we use all the available logic resources efficiently.

Table 4 gives number of cycles used in optimal ate pairing for our design. As expected, it provides better throughput if multiple pairings are computed at

Table 4. The number of cycles for one pairing

	Security [bit]	Miller loops	Final additions	Exp to p^6-1	Exp to p^2+1	Hard exp	Total # Cycles	Time [ns]
single pairing	126	37,000	976	20,792	398	34,934	94,100	409.1
3 pairings/3	126	36,664	786	7,663	356	32,300	77,769	338.1
single pairing	128	39,350	976	21,108	398	37,184	99,016	430.5
3 pairings/3	128	38,930	786	7,776	356	34,442	82,290	357.8

the same time. On the other hand, fine-pipelined architecture is affordable for pairing computation, as RNS and pairing algorithm provide enough parallelism. The increased frequency of the design could compensate the overhead induced by the pipeline bubbles.

6.2 Comparison

Table 5 provides the area and timing information of recent reported pairing implementations on both software and hardware platforms. The fastest FPGA implementation of pairing on BN curve in literature is Design II [11]. Compared

Table 5. Performance comparison of software and hardware implementations of pairings at around 128-bit security

Design	**Pairing**	**Security** [bit]	**Platform**	**Algorithm**	**Area**	**Freq.** [MHz]	**Cycle** [$\times 10^3$]	**Delay** [ms]
Ours	optimal ate	126	Xilinx FPGA (Virtex-6)	RNS (Parallel)	5237 slices 64 DSPs	210	78	0.338
		128					82	0.358
[11] Design II	optimal ate	126	Xilinx FPGA (Virtex-6)	RNS (Parallel)	7032 slices 32 DSPs	250	143	0.573
[11] Design I	optimal ate	126	Altera FPGA (Stratix III)	RNS (Parallel)	4233 ALMs 72 DSPs	165	176	1.07
[14]	ate	128	Xilinx FPGA (Virtex-6)	HMM (Parallel)	4014 slices 42 DSPs	210	336	1.60
	optimal ate						245	1.17
[16]	Tate	128	Xilinx FPGA (Virtex-4)	Blakley	52k Slices	50	1,730	34.6
	ate						1,207	24.2
	optimal ate						821	16.4
[23]	Tate	128	ASIC (130 nm)	Montgomery	97 kGates	338	11,627*	34.4
	ate						7,706*	22.8
	optimal ate						5,340*	15.8
[17]	η_T over $\mathbb{F}_{2^{1223}}$	128	Xilinx FPGA (Virtex-6)	-	15167 Slices	250	47.6	0.19
[13]	Tate over $\mathbb{F}_{3^{5\cdot 97}}$	128	Xilinx FPGA (Virtex-4)	-	4755 Slices 7 BRAMs	192	429	2.23
[1]	optimal Eta over $\mathbb{F}_{2^{367}}$	128	Xilinx Virtex-4	-	4518 Slices	220	774*	3.52
[20]	ate	128	64-bit Core2	Montgomery	-	2400	15,000	6.25
	optimal ate						10,000	4.17
[18]	ate	128	64-bit Core2	Montgomery		2400	14,429	6.01
[30]	optimal ate	128	Core2 Quad	Hybrid Mult.	-	2394	4,470	1.86
[7]	optimal ate	126	Core i7	Montgomery	-	2800	2,330	0.83
[2]	optimal ate	126	Phenom II	Montgomery	-	3000	1,562	0.52
[3]	η_T over $\mathbb{F}_{2^{1223}}$	128	Xeon (8 cores)	-	-	2000	3,020	1.51
[8]	η_T over $\mathbb{F}_{3^{509}}$	128	Core i7 (8 cores)	-	-	2900	5,423	1.87
[1]	opt. Eta $\mathbb{F}_{2^{367}}$	128	Core i5	-	-	2530	2,440	0.96

* Estimated by the authors.

with this work, our prototype provides 41.0% speed-up, and with 25.5% less FPGA area. Also, the underlying multiplier used in our design is 18×18, which is popular among the off-the-shelf products, while Design II [11] requires 25×18 multipliers, which are only available on high-end FPGAs. [17] provides almost twice faster implementations on binary field, but with around 3 times of FPGA area as ours. Also, our work is the first hardware design which outperforms software implementations for pairing on BN curves.

7 Conclusions

In this paper, we present an efficient pairing coprocessor using RNS and lazy reduction techniques. We introduce a set of RNS moduli that are suitable for 258-bit modular multiplications in the pairing computation. The proposed architecture is prototyped on a Xilinx Virtex-6 FPGA, which utilizes 5237 slices and 64 DSPs, and can run at 230 MHz. The coprocessor computes one optimal pairing in 0.358 ms. To the best of our knowledge, this is a speed record for hardware implementations for pairings on BN-curve that achieves 128-bit security. For the future work, we plan to implement other curves and different types of pairings on this architecture. Furthermore, we will provide an optimal parameter set and pairing implementations for higher security level including 192-bit or 256-bit security.

Acknowledgements. We would like to thank Nicolas Guillermin for the helpful discussion on improving the scheduling of the RNS Montgomery algorithm. We also thank the anonymous referees for the helpful comments and improving the quality of this work. This work is partly supported by the City University of Hong Kong Start-up Grant 7200179, and partly by the Research Council KU Leuven: GOA TENSE (GOA/11/007) and by the European Commission under contract number ICT-2007-216676 ECRYPT-II NoE. In addition, this work is supported in part by the Flemish Government, FWO G.0550.12N and by the Hercules Foundation AKUL/11/19.

References

1. Aranha, D., Beuchat, J.L., Detrey, J., Estibals, N.: Optimal eta pairing on supersingular genus-2 binary hyperelliptic curves. Cryptology ePrint Archive, Report 2010/559 (2010), `http://eprint.iacr.org/`
2. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (2011)
3. Aranha, D.F., López, J., Hankerson, D.: High-Speed Parallel Software Implementation of the η_T Pairing. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 89–105. Springer, Heidelberg (2010)
4. Bajard, J.C., Kaihara, M., Plantard, T.: Selected RNS bases for modular multiplication. In: ARITH 2009: Proceedings of the 2009 19th IEEE Symposium on Computer Arithmetic, pp. 25–32. IEEE Computer Society, Washington, DC (2009)

5. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient Algorithms for Pairing-Based Cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–369. Springer, Heidelberg (2002)
6. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
7. Beuchat, J.-L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 21–39. Springer, Heidelberg (2010)
8. Beuchat, J.-L., López-Trejo, E., Martínez-Ramos, L., Mitsunari, S., Rodríguez-Henríquez, F.: Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 413–432. Springer, Heidelberg (2009)
9. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
10. Cha, J.C., Cheon, J.H.: An Identity-Based Signature from Gap Diffie-Hellman Groups. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 18–30. Springer, Heidelberg (2003)
11. Cheung, R.C.C., Duquesne, S., Fan, J., Guillermin, N., Verbauwhede, I., Yao, G.X.: FPGA Implementation of Pairings Using Residue Number System and Lazy Reduction. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 421–441. Springer, Heidelberg (2011)
12. Dutta, R., Barua, R., Sarkar, P.: Pairing-based cryptographic protocols: A survey. Cryptology ePrint Archive, Report 2004/064 (2004)
13. Estibals, N.: Compact Hardware for Computing the Tate Pairing over 128-Bit-Security Supersingular Curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 397–416. Springer, Heidelberg (2010)
14. Fan, J., Vercauteren, F., Verbauwhede, I.: Efficient hardware implementation of $\mathbb{F}_p$-arithmetic for pairing-friendly curves. IEEE Transactions on Computers 61(5), 676–685 (2012)
15. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. Journal of Cryptology 23, 224–280 (2010)
16. Ghosh, S., Mukhopadhyay, D., Roychowdhury, D.: High Speed Flexible Pairing Cryptoprocessor on FPGA Platform. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 450–466. Springer, Heidelberg (2010)
17. Ghosh, S., Roychowdhury, D., Das, A.: High Speed Cryptoprocessor for η_T Pairing on 128-bit Secure Supersingular Elliptic Curves over Characteristic Two Fields. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 442–458. Springer, Heidelberg (2011)
18. Grabher, P., Großschädl, J., Page, D.: On Software Parallel Implementation of Cryptographic Pairings. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 35–50. Springer, Heidelberg (2009)
19. Groth, J., Sahai, A.: Efficient Non-interactive Proof Systems for Bilinear Groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
20. Hankerson, D., Menezes, A., Scott, M.: Software Implementation of Pairings. Cryptology and Information Security Series, vol. 2, pp. 188–206. IOS Press (2009)
21. Hess, F., Smart, N., Vercauteren, F.: The Eta pairing revisited. IEEE Transactions on Information Theory 52(10), 4595–4602 (2006)

22. Joux, A.: A one round protocol for tripartite Diffie-Hellman. Journal of Cryptology 17, 263–276 (2004)
23. Kammler, D., Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg, M., Auras, D., Ascheid, G., Mathar, R.: Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 254–271. Springer, Heidelberg (2009)
24. Kawamura, S.-I., Koike, M., Sano, F., Shimbo, A.: Cox-Rower Architecture for Fast Parallel Montgomery Multiplication. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 523–538. Springer, Heidelberg (2000)
25. Koblitz, N.: Elliptic Curve Cryptosystem. Mathematics of Computation 48, 203–209 (1987)
26. Lee, E., Lee, H.S., Park, C.M.: Efficient and generalized pairing computation on abelian varieties. IEEE Transactions on Information Theory 55(4), 1793–1803 (2009)
27. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
28. Miller, V.S.: The Weil pairing, and its efficient calculation. Journal of Cryptology 17, 235–261 (2004)
29. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
30. Naehrig, M., Niederhagen, R., Schwabe, P.: New Software Speed Records for Cryptographic Pairings. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 109–123. Springer, Heidelberg (2010)
31. Parhami, B.: Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press (2000)
32. Posch, K., Posch, R.: Base extension using a convolution sum in residue number systems. Computing 50, 93–104 (1993)
33. Posch, K., Posch, R.: Modulo reduction in residue number systems. IEEE Transactions on Parallel and Distributed Systems 6(5), 449–454 (1995)
34. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
35. Scott, M.: Implementing cryptographic pairings. In: Pairing-Based Cryptography - Pairing 2007. LNCS, vol. 4575, pp. 117–196. Springer (2007)
36. Szerwinski, R., Güneysu, T.: Exploiting the Power of GPUs for Asymmetric Cryptography. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 79–99. Springer, Heidelberg (2008)
37. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory 56(1), 455–461 (2010)