A FPGA pairing implementation using the Residue Number System

Sylvain Duquesne, Nicolas Guillermin

IRMAR, UMR CNRS 6625
Université Rennes 1
Campus de Beaulieu
35042 Rennes cedex, France
sylvain.duquesne@univ-rennes1.fr, nicolas.guillermin@m4x.org

Abstract. Recently, a lot of progresses have been made in software implementations of pairings at the 128-bit security level in large characteristic. In this work, we obtain analogous progresses for hardware implementations. For this, we use the RNS representation of numbers which is especially well suited for pairing computation in a hardware context. A FPGA implementation is proposed, based on an adaptation of Guillermin's architecture which computes a pairing in 1.07 ms. It is 2 times faster than all previous hardware implementations (including ASIC and small characteristic implementations) and almost as fast as best software implementations.

Keywords: Elliptic curves, optimal pairing, Barreto-Naerigh curves, Residue Number System (RNS), efficient hardware implementation.

1 Introduction

Bilinear pairings on elliptic curves have been introduced in cryptography in the middle of 90's for cryptanalysis. Indeed, they allow to transfer the discrete logarithm on an elliptic curve to a discrete logarithm in the multiplicative group of a finite field, where subexponential algorithms are available [16, 24]. In 2000, Joux introduces the first constructive use of pairings with a tripartite key exchange protocol [20]. Since, it has been shown that pairings can be used to construct new protocols like identity based cryptography [9] or short signature [10]. As a consequence, pairings became very popular in asymmetric cryptography and computing them as fast as possible is very important. In this work, we are interested by hardware implementation of pairing in the 128-bit security level in large characteristic. Indeed, the recent introduction of Barreto-Naehrig curves [7] and optimal pairings [30] leads to very efficient implementations, especially in software. We combine all the recent algorithmical improvements of pairing in large characteristic and a FPGA architecture based on the Residue Number System to obtain the fastest hardware implementation of pairing at the 128-bit security level. This paper is organized as follows. Section 2 gives an overview of optimal Ate pairing computation over Barreto-Naehrig (BN) curves. The Residue Number System (RNS) and the field arithmetic optimisations used for pairing computations are presented in Section 3. In Section 4, we explain our choice of curve and detail the algorithmic optimisations specific to these curves. Section 5 describes our pipeline architecture and Section 6 summarizes cycle count and provides comparison of results with the previous implementations in the literature. Section 7 concludes the paper.

2 Pairing on elliptic curves and their computation

Pairings on elliptic curves are bilinear maps from the curve to the multiplicative group of a finite field $\mathbb{F}_{p^k}$ where k is called the embedding degree. It is usually very large so that computing in $\mathbb{F}_{p^k}$ is not reasonable. This is reassuring regarding the destructive use of pairings but annoying for pairing based cryptosystems. Small embedding degrees can be easily obtained by using supersingular curves. However, it is too small ($k \leq 2$) if large characteristic base fields are used as in this work. Thus, we use ordinary curves with prescribed embedding degrees constructed via the complex multiplication method as surveyed in [15]. We focused on the most popular one to date, namely the Barreto-Naehrig curves having embedding degree equal to 12 [7]. The reason of their popularity it that if p is a 256-bit prime number, such curves ensure a 128-bit security level both on the curve and on the finite field $\mathbb{F}_{p^{12}}$, assuming NIST recommendations [27].

2.1 Pairings on Barreto-Naehrig curves

Let $u \in \mathbb{Z}$, $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ a prime number and E an elliptic curve defined over $\mathbb{F}_p$ by an equation

$$y^2 = x^3 + a_6, \tag{1}$$

such that E has prime cardinality $\ell = 36u^4 + 36u^3 + 18u^2 + 6u + 1$. Such curves were introduced in [7] and have 12 as an embedding degree. This means that ℓ divides $p^{12} - 1$ but not $p^k - 1$ for $0 \leq k < 12$ and that the full ℓ-torsion of the curve is defined on the field $\mathbb{F}_{p^{12}}$.

Let m be an integer and Q a point on E. If Q_m denotes the point mQ, $f_{(m,Q)}$ is the function on the curve whose divisor is

$$\mathrm{div}(f_{(m,Q)}) = m\,Q - Q_m - (m-1)\mathcal{O}.$$

This function is the core of all known pairings and is computed thanks to an adaptation of classical scalar multiplication algorithm [25]. In this algorithm $g_{(A,B)}$ is the equation of the line passing through

Algorithm 1: `Miller`(m, Q, P)

Data: $m \in \mathbb{Z}$ with binary representation $(m_{s-1}, \cdots, m_0)_2$, P and Q in $E(\mathbb{F}_{p^{12}})$.
Result: $f_{(m,Q)}(P) \in \mathbb{F}_{p^{12}}$.
begin
  $f \leftarrow 1$ and $T \leftarrow Q$
  for *i from* $s-2$ *downto 0* **do**
    $f \leftarrow f^2 . \frac{g_{(T,T)}(P)}{v_{2T}(P)}$ and $T \leftarrow 2T$
    if $m_i = 1$ **then**
      $f \leftarrow f . \frac{g_{(T,Q)}(P)}{v_{T+Q}(P)}$ and $T \leftarrow T + Q$
    end if
  end for
  return f
end

the points A and B (or tangent to E in A if $A = B$) and v_A is the equation of the vertical line passing

by A, so that $\frac{g_{(A,B)}}{v_{A+B}}$ is the function on E involved in the addition of A and B.
In this paper, we are interested in the optimal Ate pairing [26] because it is the most efficient to date for BN curves but the same work can be easily done with other pairings. If $r = 6u + 2$, it is given by

$$\begin{aligned} e_o : E(\mathbb{F}_{p^{12}}) \cap \mathrm{Ker}(\pi - p) \times E(\mathbb{F}_p)[\ell] \ &\rightarrow \ \mathbb{F}^*_{p^{12}} / \left(\mathbb{F}^*_{p^{12}}\right)^{\ell} \\ (Q, P) \ &\mapsto \ \left(f_{(r,Q)}(P) . g_{(rQ,\pi(Q))}(P) . g_{(rQ+\pi(Q),-\pi^2(Q)}(P)\right)^{\frac{p^{12}-1}{\ell}} \end{aligned}$$

where π is the Frobenius map on the curve $(\pi : (x, y) \mapsto (x^p, y^p))$.

2.2 Efficient implementation of pairings

An other advantage of BN curves is their degree 6 twist. This means E is isomorphic over $\mathbb{F}_{p^{12}}$ to a curve $\tilde{E}$ defined by $y^2 = x^3 + \frac{a_6}{\nu}$, where ν is an element in $\mathbb{F}_{p^2}$ which is not a cube nor a square. Then we can defined twisted versions of pairings on $\tilde{E}(\mathbb{F}_{p^2}) \times E(\mathbb{F}_p)[\ell]$. This means that the coordinates of Q can be written as $(x_Q \nu^{\frac{1}{3}}, y_Q \nu^{\frac{1}{2}})$ where x_Q and y_Q are in $\mathbb{F}_{p^2}$. There are three important consequences on the Miller loop.

- Computing $g, v, 2T$ and $T + Q$ requires only $\mathbb{F}_{p^2}$ arithmetic (but the result remains in $\mathbb{F}_{p^{12}}$).
- The denominators (v_{2T} and v_{T+Q}), and more generally all the factors of f lying in a proper subfield of $\mathbb{F}_{p^{12}}$, are wiped out by the final exponentiation. This is the famous denominator elimination introduced in [6].
- The numerators ($g_{T,T}$ and $g_{T,Q}$) have the particular form $g_0 + g_2 \nu^{\frac{1}{2}} + g_3 \nu^{\frac{1}{3}}$ where $g_i \in \mathbb{F}_{p^2}$ which contains only 6 coefficients instead of 12. Hence the multiplication by such an element during the writing up of f is cheaper than a complete multiplication in $\mathbb{F}_{p^{12}}$.

Finally, Koblitz and Menezes show in [23] that the cost of the final exponentiation can be reduced thanks to the integer factorization

$$\frac{p^{12} - 1}{\ell} = \left(p^6 - 1\right)\left(p^2 + 1\right)\left(\frac{p^4 - p^2 + 1}{\ell}\right)$$

Then the final exponentiation can be split in two steps

- Powering to the $p^6 - 1$ and to the $p^2 + 1$. This is easily obtained via cheap Frobenius computations and an inversion.
- Powering to the $\frac{p^4 - p^2 + 1}{\ell}$ which is called the hard part of the final exponentiation. As explained in [17], there are efficient ways to compute it for BN curves and the cost is around three-quarters of the cost of an exponentiation with an exponent having the same size than p. Moreover, as f has been raised to the $\left(p^6 - 1\right)\left(p^2 + 1\right)$, it has order $p^4 - p^2 + 1$. Then, as noticed in [17], it is in $G_{\Phi_6}(\mathbb{F}_{p^2})$ and squaring such an element (which is the most used operation in the hard part of the exponentiation) is less expensive than a classical squaring in $\mathbb{F}_{p^{12}}$.

As noticed in [12], choosing a RNS based architecture would be of great interest for hardware implementation of pairings. This is due to the massive use of lazy reduction in extension fields. Let us now recall this system for representing numbers and describe our architecture.

3 RNS

3.1 Notation

In all the paper, for $a, b \in \mathbb{N}$, we denote by $|a|_b$ the result of a modulo b.

3.2 Introduction to Residue Number System

Let $B = \{m_1, \cdots, m_n\}$ be a set of co-prime natural integers, and $M = \prod_{i=1}^{n} m_i$. Let $0 \leq X < M$. The residue number system (RNS) representation $\{X\}_B$ is the set of positive integers $\{x_1, \cdots, x_n\}$ with $x_i = |X|_{m_i}$. $\{x_1, \cdots, x_n\}$ are called the RNS digits of X. They are unique for each X, and X can be recomputed from them using the following fomula :

$$X = \left| \sum_{i=1}^{n} |x_i \times M_i^{-1}|_{m_i} \times M_i \right|_M \text{ where } M_i = \frac{M}{m_i}. \tag{2}$$

The strongest advantage of a such system is that it distributes large integer operations on the small residue values. The operations are performed independently on the residues. In particular, there is no carry propagation over addition, and multiplication complexity is linear with n. M is called the module of B.

3.3 RNS Montgomery reduction algorithm :

Using RNS representation in B ensures very efficient computation in $\mathbb{Z}/M\mathbb{Z}$. To efficiently compute pairing over $\mathbb{F}_p$ we need to introduce a reduction by p algorithm.

The papers [5, 22] propose an adaptation of the well known Montgomery algorithm in the context of RNS representation. We refer the reader to these papers for more details on RNS reduction algorithm. Here we recall the useful results for pairing computation.

We define B and $\tilde{B}$ two RNS bases of the same size n, with their coprime modules M and $\tilde{M}$. We choose B and $\tilde{B}$ such that $M > \alpha p$ and $\tilde{M} > 3p$ (α being a parameter defining the maximal value we are able to reduce thanks to this algorithm. See section 5 for the definition in our specific case). For all input $A < \alpha p^2$ given in B and $\tilde{B}$, the RNS Montgomery algorithm computes S in B and $\tilde{B}$ such that $S < 3p$ and $|S|_p = |a \times M^{-1}|_p$. This algorithm calls 2 times the $B_{ext}(X, B_1, B_2)$ subroutine which computes $\{|X|_{M_1}\}_{B_2}$ from $\{X\}_{B_1}$.

Algorithm 2: $\texttt{Red}(X, p, B, \tilde{B})$

Data: B and $\tilde{B}$ RNS bases with module $M > \alpha p$ and $\tilde{M} > 3p$,
p co-prime with M and $\tilde{M}$,
$\{X\}_B$ and $\{X\}_{\tilde{B}}$ RNS representation of $X < \alpha p^2$ in B and $\tilde{B}$,
precalculations : $\{|-p^{-1}|_M\}_B$,$\{|M^{-1}|_{\tilde{M}}\}_{\tilde{B}}$ and $\{p\}_{\tilde{B}}$,
algorithm $B_{ext}(A, B_1, B_2)$ computing $\{|A|_{M_2}\}_{B_2}$ from $\{A\}_{B_1}$
Result: $\{S\}_B$ and $\{S\}_{\tilde{B}}$ such that $|S|_p = |XM^{-1}|_p$ and $S < 3p$
begin
1 $Q \leftarrow X \times |-p^{-1}|_M$ in B
2 $\tilde{Q} \leftarrow B_{ext}(Q, B, \tilde{B})$
3 $\tilde{R} \leftarrow X + \tilde{Q} \times p$ in $\tilde{B}$
4 $\tilde{S} \leftarrow \tilde{R} \times M^{-1}$ in $\tilde{B}$
5 $S \leftarrow B_{ext}(S, \tilde{B}, B)$
6 **return** S and $\tilde{S}$
end

There are many ways to realize the B_{ext} algorithm. The asymptotically cheapest way being given in [4] with the overall complexity $\frac{7}{5}n^2 + \frac{8}{5}n$ RNS digit-products. Nevertheless this version is hardly parallelizable and gives poor results while using it for cryptographic size over a hardware design.

We therefore will use the version of Kawamura *et al.* [22]. This algorithm has overall complexity of $2n^2 + 3n$, but at each time every calculation can be made on n RNS digits in a row.

3.4 The Cox-Rower architecture

The Cox-Rower architecture was first proposed by Kawamura et al. in [22], for the purpose of RSA multiplication. It was enhanced by Guillermin in [18], with support of full arithmetic operations in $\mathbb{F}_p$, and fast elliptic curve scalar multiplication. The main feature of Guillermin's design is the use of n parallel Rower achieving one RNS digit multiplication and accumulation per cycle. B and $\tilde{B}$ are chosen such that $\forall m \in \{B, \tilde{B}\}$, $2^w(1 - 2^{-\frac{w}{2}}) < m \leq 2^w$ where w is the architecture word size, typically 18 or 36 on a FPGA (which embeds very efficient 18×18 multipliers). n is chosen such that $2^{wn} > p$. Each Rower is able to compute at every clock cycle $|a \times b|_{m_i}$ or $|a \times b|_{\tilde{m}_i}$. It can also add it simultaneously with the previous result. The entire design can therefore execute a full length multiplication in 2 cycles (one cycle over B and one over $\tilde{B}$), an addition in 4 (using 2 accumulated multiplications by 1), a subtraction in 6 (an addition with a multiple of p to keep the result positive is mandatory) and a whole reduction in $2n + 3$ cycles.

An adaptation of Guillermin's architecture is necessary to provide pairing support. As the number of local variables and precalculations is much larger for pairing computation than for simple elliptic curve scalar multiplication, we used a single triple port RAM of 256 words instead of the ROM and the bunch of 16 registers to store precalculations and temporary results.

These RAMs are generated with the Altera Megawizard tool, and consume only 2 M9k on Stratix III per Rower. Our design for 256-bit curves only requires 8% of the EP3SE50F484, the smallest FPGA of the Stratix III series. We made this choice for the sake of simplicity, but it can be improved at the price of classical register allocation techniques (for ASIC implementations, among other).

On figure 3 in appendix we present the adaptation of Guillermin's architecture for pairing computation (with FPGA triple port RAM).

3.5 Finite field arithmetic

The most consuming operation in RNS is the reduction, while multiplication and addition over $\mathbb{F}_p$ have a the same weak complexity. This leads to completely different choices compared to classical multiprecision representation.

Lazy reduction : Let us consider any $E = \sum_{i=0}^{l} A_i B_i$ with $A_i, B_i \in \mathbb{F}_p$. We call lazy reduction the fact that this kind of expression only needs one reduction after the l multiplications. Whereas lazy reduction can be used with classical multiprecision representation, as in [3], it is even more interesting in RNS. Indeed the complexity of the arithmetic is concentrated on the reduction step. For example, a Cox Rower with n Rowers (n being the size of the RNS bases B and $\tilde{B}$) needs $2l$ cycles to compute the unreduced representation of E (l cycles over B and l over $\tilde{B}$), and eventually $2n + 3$ cycles to reduce it. The only thing we need to take notice of is the α parameter : M must be up to $\max(E)/p$ with E positive, unless the result of the `Red` algorithm is wrong. We observe that the most consuming part of pairings are multiplications and squaring in extension fields of $\mathbb{F}_p$. If the field extension representations are well chosen, multiplication in these extensions are candidate to a massive use of lazy reduction. The reduction part of any product in $\mathbb{F}_{p^k}$ costs only k reductions, then $2kn + 3k$ cycles.

Extension field optimizations : For considered field extensions ($k \leq 12$), reductions are the most consuming part. Nevertheless extension field optimizations may accelerate the overall pairing. In [12], the author studied the complexities of both a RNS and a classical implementation of pairings assuming that interpolation methods are used, such as Karatsuba or Toom-Cook. It spares a multiplication compared to schoolbook at the price of 4 additions or subtractions. These techniques are only useful when the relative cost of a multiplication is high compared to addition. In RNS multiplication has a comparable price with addition, interpolation is therefore less attractive. We do not define in this work any method to find the best way to implement tower field arithmetic for BN curves, but propose specific optimisations for our chosen curves in section 4.1. We justify our choice in section 5.

4 Optimal pairing for BN curves

4.1 Choice of the curve

It is explain in [28] how to generate BN curves having nice properties. For this work we chose two curves both with $a_6 = 2$. The first one, called BN_{126} is defined by $u = -(2^{62}+2^{55}+1)$ and has already been used in [28, 3]. It ensures only 126 bits of security but allows to use lazy reduction techniques without requiring an extra word in 32 or 64-bit architecture. Thanks to the size of the FPGA DSP blocks (18×18), the architecture proposed here can have w up to 36 bits words so that we do not have this restriction and we can consider a curve, called BN_{128} really ensuring 128 bits of security without artificial extra cost. In both cases, our architecture requires $n = 8$ Rower and the word size $w = 34$ ensure sufficient margin for lazy reductions considering the algorithmic choices we made (see section 5 for details). This curve is defined by $u = -(2^{63} + 2^{22} + 2^{18} + 2^7 + 1)$. In both cases, $\mathbb{F}_{p^{12}}$ is defined by the following tower of extensions

$$\begin{array}{rl} \mathbb{F}_{p^2} = \mathbb{F}_p[X]/(X^2+1) & = \mathbb{F}_p[\mathbf{i}] \\ \mathbb{F}_{p^6} = \mathbb{F}_{p^2}[Y]/(Y^3-(1+\mathbf{i})) & = \mathbb{F}_{p^2}[\boldsymbol{\beta}] \\ \mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[Z]/(Z^2-\boldsymbol{\beta}) & = \mathbb{F}_{p^6}[\boldsymbol{\gamma}]\ (= \mathbb{F}_{p^2}[\boldsymbol{\gamma}]). \end{array}$$

This tower of extensions has many advantages, the most important being an efficient multiplication algorithm for the canonical polynomial base. The major difficulty is the apparition of the term $1+\mathbf{i}$ for the definition of $\mathbb{F}_{p^6}$. We used then some adaptations of the Guillermin's architecture [18]. We refer the reader to the section 5 for more details. The optimal Ate pairing on these curves can then be computed thanks to the algorithm 3 where `dbl, add` and `hard-part` are given in appendix. Note that the line 4 is due to $u < 0$ and is potentially expensive (inversion) but thanks to the final exponentiation, f^{-1} can be replaced by f^{p^6} [3] which is almost for free (conjugation). Moreover, since $r = 6u+2$ has Hamming weight 5 (resp. 9) for BN_{126} (resp. BN_{128}), `add` is used only 6 (resp. 10) times.

4.2 Formulas for Miller loop

Jacobian coordinates are usually used for pairing computations [8, 19, 26] but projective coordinates are more interesting in our case [3, 11]. Let $P = (x_P, y_P) \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^{12}})$. Using the degree 6 twist on the curve, Q can be written $(x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3)$ with x_Q and $y_Q \in \mathbb{F}_{p^2}$. The "doubling" step of the Miller loop for optimal pairing is consisting in two stages:

- The doubling of a temporary projective point T in $E(\mathbb{F}_{p^{12}})$. As T is a multiple of Q, it has the form $(X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T)$ with X_T, Y_T and $Z_T \in \mathbb{F}_{p^2}$.

Algorithm 3: optimal(P, Q)

Data: $r = |6u + 2|$, $P = (x_P, y_P) \in E(\mathbb{F}_p)[\ell]$,
$Q = (x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3) \in E(\mathbb{F}_{p^{12}}) \cap \mathrm{Ker}(\pi - p)$ with x_Q and $y_Q \in \mathbb{F}_{p^2}$.
Result: $e_o(P, Q) \in \mathbb{F}_{p^{12}}$.
begin
1 $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T) \leftarrow (x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3, 1)$ and $f \leftarrow 1$
for *i from* $\lfloor \log_2(r) \rfloor - 2$ *downto 0* **do**
2 $T, g \leftarrow \mathtt{dbl}(T, P)$ and $f \leftarrow f^2.g$
if $m_i = 1$ **then**
3 $T, g \leftarrow \mathtt{add}(T, Q, P)$ and $f \leftarrow f.g$
end if
end for
4 $T \leftarrow -T$, $f \leftarrow f^{-1}$
5 $Q_1 \leftarrow \pi(Q)$
6 $T, g \leftarrow \mathtt{add}(T, Q_1, P)$ and $f \leftarrow f.g$
7 $Q_2 \leftarrow -\pi(Q_1)$
8 $T, g \leftarrow \mathtt{add}(T, Q_2, P)$ and $f \leftarrow f.g$
9 $f \leftarrow \left(f^{p^6-1}\right)^{p^2+1}$
10 $f \leftarrow \mathtt{hard\text{-}part}(f, |u|)$.
11 **return** f
end
.

– The evaluation in P of the tangent line in T to the curve. As recalled in Section 2.2, we do not need to take into account the multiplicative factors lying in a proper subfield of $\mathbb{F}_{p^k}$ (they are cancelled by the final exponentiation).

It is given by algorithm 4 in appendix. The classical formulae are rearranged to highlight the reductions (every temporary results need a reduction except F, G, H and t_3), and the inherent parallelism in the local variables (Each line can be implemented in a random order). This is important for avoiding idle state in Cox-Rower (see section 5 for more details). The total cost of this step is 4 multiplications, 5 squaring, 8 reductions in $\mathbb{F}_{p^2}$ and 2 modular multiplications of an element of $\mathbb{F}_{p^2}$ by an element of $\mathbb{F}_p$. Note that multiplications like $2X_TY_T$ can be transformed into squaring but at the cost of some extra additions [3, 11] so that it is not interesting for our design.

In the same way, the addition step is consisting in the sum of T and Q followed by the evaluation in P of the line passing through T and Q and is given by algorithm 5. The total cost of this step is 11 multiplications, 2 squaring, 10 reductions in $\mathbb{F}_{p^2}$ and 2 modular multiplications of an element of $\mathbb{F}_{p^2}$ by an element of $\mathbb{F}_p$.

4.3 Final exponentiation

The line 9 requires a conjugation, an inversion in $\mathbb{F}_{p^{12}}$, a Frobenius computation and a multiplication in $\mathbb{F}_{p^{12}}$. In this "easy" part of the final exponentiation, the inversion is in fact very expensive. It can be done with only one inversion, 97 multiplications and 35 reductions in $\mathbb{F}_p$ [12, 19], but the remaining inversion in $\mathbb{F}_p$ is done via an exponentiation to the $p - 2$ on our design. For this operation, we use a simple square and multiply algorithm with least significant bit first. It is more memory consuming,

but presents more independence between the local variables. Due to pipeline idle steps, this operation only needs $\lfloor log(p) - 1 \rfloor$ multiplications for the calculation of X^{2^i} (with X the value to inverse), the other operations are executed during the necessary pipeline idle states. Then, there is no need to find more efficient ways to execute the exponentiation here. Hence this step will be very expensive compared to the others or compared to a software implementation. On the contrary, the Frobenius only costs 5 multiplications in $\mathbb{F}_{p^2}$ and 10 reductions, thanks to the definition of the tower of extensions.

To date, the fastest way for computing the hard part (line 10 of algorithm 3) of the final exponentiation involved a multi-addition chain [29]. The corresponding algorithm `hard-part` is given in Appendix in a way using less temporary variables than in [29]. Moreover, as noticed in [17], f is in $G_{\Phi_6}(\mathbb{F}_{p^2})$ and squaring such an element is less expensive than a classical squaring in $\mathbb{F}_{p^{12}}$. In our context, if $a = \sum_{i=0}^{5} a_i \boldsymbol{\gamma}^i$ with $a_i \in \mathbb{F}_{p^2}$, the coefficient of $A = a^2$ are given by

$$\begin{array}{ll} A_0 = 3a_0^2 + 3(1+\mathbf{i})a_3^2 - 2a_0 & A_1 = 6(1+\mathbf{i})a_2a_5 + 2a_1 \\ A_2 = 3a_1^2 + 3(1+\mathbf{i})a_4^2 - 2a_2 & A_3 = 6a_0a_3 + 2a_3 \\ A_4 = 3a_2^2 + 3(1+\mathbf{i})a_5^2 - 2a_4 & A_5 = 6a_1a_4 + 2a_5 \end{array}$$

More recently, Karabina introduced a compressed form for elements in $\mathbb{F}_{p^{12}}$ which requires less operations to be squared [3, 21]. Unfortunately, this method involves extra inversions so that it is unsuited for our design.

5 Pipeline architecture and extension field arithmetic

In this section we start from the definition of Guillermin's pipeline [18], and show how to construct efficient hardware for the chosen curve in 4.1. Subsection 5.1 describes constraints we followed during the pipeline architecture process. In subsection 5.2 and 5.3, we justify these optimizations regarding operations in pairing and evaluate their cost in Stratix II ALM. Eventually in subsection 5.4, we evaluate the overall gain in clock cycle brought by these optimizations. Figure 1 recalls the one used by [18] and compares it to ours.

5.1 Pipeline depth

Our goal is to keep the maximal frequency already available in [18]. Additive hardware must be carefully introduced, to keep the critical path under control. On the other hand, we can easily raise the pipeline depth without a lot of cycle loss. Indeed pairing computation provides less dependency between local variables than classical elliptic curve scalar multiplication over $\mathbb{F}_p$. We can use this parallelism to avoid idle states due to the pipeline effect. Algorithm `dbl` and `add` show this inherent parallelism during the Miller loop. On each line, every value are independent with one another. The order in the line is then not important. Operations in $\mathbb{F}_{p^{12}}$ also present such inherent parallelism.

If π_d is the pipeline depth, a result of an operation is available $\pi_d + 1$ cycles after the computation began, and so π_d independent instructions must be inserted before reusing the result. In [22], $\pi_d = 2$, while in [18], $\pi_d = 5$. During our implementation we found that π_d could be up to 8 and still avoid idle states in the whole pairing, except for $\mathbb{F}_p$ inversion (see subsection 4.3).

We note i_1, i_2 the two input of the ALU of figure 3 and $\epsilon_i = 2^r - m_i$, the modulus of B or $\tilde{B}$.

5.2 Additive hardware for multiplication and squaring in $\mathbb{F}_{p^2}$

Using Guillermin architecture, a $\mathbb{F}_{p^2}$ multiplication over the chosen extension costs 12 cycles using naive schoolbook multiplication. Among them, 4 cycles are lost to compute the subtraction due to $\mathbf{i}^2 = -1$. If we add the ability to get the opposite of the result before accumulating it in a pipeline stage, we spare 4 cycles for every computation. As i_1 and i_2 are in fact less than $3p$ (they are reduced values), the opposite hardware module computes $|9p^2 - x|_{m_i}$ from x the output of the pipeline (the result of the operation needs to remain positive). This module only costs 55 ALM per Rower, and has a critical path suitable with the rest of the design.

Karatsuba optimization for $\mathbb{F}_{p^2}/\mathbb{F}_p$ is here not efficient. Using Guillermin's pipeline, no less than 18 cycles is needed. The cost of an adder is around 42 ALM. No way has been found to add hardware to make multiplication fall under 8 cycles while preserving correct pipeline depth and critical path.

The opposite hardware module also spares 4 cycles per $\mathbb{F}_{p^2}$ squaring. Another important optimization for squaring is the capacity to compute $|2i_1 \times i_2|_{m_i}$ in one cycle. Squaring in $\mathbb{F}_{p^2}$ falls from 8 cycles to 6. This optimization only costs 2 muxes per Rower (see figure 1). A side effect of this optimization is the need of the precomputed value $18p^2$ instead of $9p^2$ in the opposite module.

5.3 Squaring and Multiplication over $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^2}$:

Schoolbook multiplication and squaring : Let $A = \{a_1, \cdots, a_{12}\}$, $B = \{b_1, \cdots, b_{12}\}$ and $C = AB \in \mathbb{F}_{p^{12}}$. To optimize schoolbook multiplication and squaring with a Cox-Rower, we aim to execute only once $i_1 = a_i$ and $i_2 = b_j$, for all $\{i, j\} \in \{1, \cdots, 12\}^2$. As $\mathbf{i}^2 = -1$ and $\gamma^6 = (1+\mathbf{i})$, the result $|a_i \times b_j|$ is either added or subtracted on at most two components of C. Two selectable accumulators combined with opposite calculations are then of great interest. These hardware modules (a combination of an opposite module and an accumulator) cost 150 ALM per Rower. 4 different types of multiplications and squaring are needed during pairing (cycle counts are given excluding the reduction step, which is exactly the same).

- Squaring in the doubling part of the Miller loop : using schoolbook squaring together with the proposed extra hardware costs 156 cycles. In Guillermin's pipeline, the same operation is computed in 210 cycles.
- Multiplication by the line in the Miller loop : half the values of B are equal to 0. Using our extra hardware, this costs 144 cycles instead of 177 cycles.
- Squaring in $G_{\Phi_6}(\mathbb{F}_{p^2})$. We use the formulae of subsection 4.3. With our extra hardware, we need 84 cycles instead of 160.
- Whole multiplications in the final exponentiation : no specific optimization is proposed. We spend here 288 cycles instead of 382 in Guillermin's architecture.

Interpolation techniques : As interpolation needs a lot of additions, it may be interesting to put an addition capability in our pipeline. But we need to evaluate if they are useful. We start from Karatsuba for arithmetic in $\mathbb{F}_{p^{12}}$ over $\mathbb{F}_{p^6}$, which is the best case for these techniques (interpolation at other levels $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}/\mathbb{F}_{p^2}$ or $\mathbb{F}_{p^4}/\mathbb{F}_{p^2}$ would give poorer results). Once again we consider 4 cases seen above :

- multiplication of f by the line g at line 2 and 3 of algorithm 3 (g having half its term set to 0): the schoolbook multiplication costs 144 cycles, with no need of addition while Karatsuba costs 180 cycles with additional hardware.
- squaring in the miller loop : the schoolbook squaring costs 156 cycles, but could be replaced through : $(A + \gamma B)^2 = \big((A + B)(A + \gamma^2 B) - (1 + \gamma^2)AB\big) + \gamma(2AB)$. This time with a total cost of 180 cycles.

– squaring in the cyclotomic subgroup : here we use algorithm of subsection 4.3. The total cost of it is 84 cycles not considering the reduction step. Karatsuba is not of any help here.
– general multiplication in $\mathbb{F}_{p^{12}}$. Here Karatsuba could be of some help, replacing the 288 cycles schoolbook multiplication by 264 cycles if we can use an additive instruction (there are 48 such addition instructions, an addition hardware module is then mandatory to make Karatsuba efficient). Nevertheless only 20 such general multiplications are needed for pairing over BN_{126}, and only 26 for BN_{128} , leading to a spare of respectively 480 and 624 cycles (less than 0.4% of the total number of instruction).

Finally, adding an addition module to our architecture for interpolation techniques seems to be globally not interesting. Moreover the addition has to be integrated in the pipeline by using muxes and additional registers, and may degrade the maximal frequency. We decided not to implement it.

5.4 final pipeline and cycle spare count

The optimized pipeline definition is given in figure 1. It compares it with original Guillermin's pipeline. Our different proposed optimizations allows us to spare 25500 cycles for a whole pairing ($\sim 15\%$). A schematic is given in appendix (figure 7). The final pipeline depth π_d is 8. Because of our chosen arithmetic, the value α defined in subsection 3.2 reaches 378. The worst case is reached during the schoolbook multiplication. Therefore the word length w is set to 34 in the Cox-Rower.

Fig. 1. Pipeline description and comparison with [18]

	Guillermin [18]	Ours
stage 1	$MSB_1 \| \| LSB_1 = i_1 \times i_2$	$MSB_1 \| \| LSB_1 = i_1 \times i_2$
stage 2	$MSB_2 \| \| LSB_2 = MSB_1 \times \epsilon_i$ and $(SR2 = \|LSB1 + 0\|_{m_i}$ or $SR2 = \|LSB1 + ROM_{cox}\|_{m_i})$	$(MSB_1 \| \| LSB_1 = i_1 \times i_2$ and $(SR2 = \|LSB1 + 0\|_{m_i}$ or $SR2 = \|LSB1 + ROM_{cox}\|_{m_i}))$ or $(MSB_2 \| \| LSB_2 = MSB_1 \times 2\epsilon_i$ and $SR2 = \|LSB1 + LSB1\|_{m_i})$
stage 3	$LSB3 = MSB_2 \times \epsilon_i$ and $SR3 = \|LSB2 + SR2\|_{m_i}$	$LSB3 = MSB_2 \times \epsilon_i$ and $SR3 = \|LSB2 + SR2\|_{m_i}$
stage 4	$(acc_1 = \|acc_1 + LSB3\|_{m_i}$ or $acc_1 = LSB3)$ and $(acc_2 = \|acc_2 + SR3\|_{m_i}$ or $acc_1 = SR3)$	$SR4 = \|SR3 + LSB3\|_{m_i}$
stage 5	$o = \|acc_1 + acc_2\|_{m_i}$	$(SR5_1 = \|18p^2 - SR4\|_{m_i}$ or $SR5_1 = SR4)$ and $(SR5_2 = \|18p^2 - SR4\|_{m_i}$ or $SR5_2 = SR4)$
stage 6		$(acc_1 = \|acc_1 + SR5_1\|_{m_i}$ or $acc_1 = SR5_1)$ and $(acc_2 = \|acc_2 - SR5_2\|_{m_i}$ or $acc2_2 = SR5_2)$
stage 7		$o = acc_1$ or $o = acc2$

6 Results and comparison

In this section we give our results and compare it to the state of art designs and software implementations.

6.1 Intermediary operations cycle count

At this point the fastest known hardware implementation of pairing on BN curves is given by Fan *et al.* in [14]. In this paper they used the Hybrid Modular Multiplication technique to improve $\mathbb{F}_p$ arithmetic, and compute a full R-Ate pairing over a BN curve of security level 128. This work divides the number of necessary cycles to compute a full pairing by a factor of around 3. Here we give the operation details for curves BN_{126} and BN_{128}. f^2 stands for squaring in $\mathbb{F}_{p^{12}}$ in the doubling part of the Miller loop, and $f.g$ stands for the line multiplication (including the reduction step).

Op	`dbl`	`add`	f^2	$f.g$	Miller	$f^{\frac{p^k-1}{\ell}}$	e_O
[14]	996	1260	1541	1239	311418	281558	592976
BN_{126}	507	581	384	372	86530	89581	176111
BN_{128}	507	581	384	372	92480	94101	192502

We see that we spare cycles at each step of the calculation, mostly thanks to RNS. Other implementation also exist, but we cannot compare their cycle count with us since they use small characteristic [13, 2] or high speed processors ([3, 8, 26]).

6.2 Overall results and comparison

We synthesized our design on 3 target technologies : Altera Cyclone III, Stratix II and Stratix III. The first one, the EP2C35, is a low cost FPGA used for bigger series. Its public price is less than $100 for 1 chip on [1], for the one we used. The second one is proposed for a comparison point with [13, 2] which used a comparable technology with same technological node (Xilinx Virtex IV). We point out the fact that size comparison is not fair, since we do not take in account DSP blocks, and they use none. Our DSP block consumption is important (2 36×36 DSP blocks and 1 9×9 per Rower) and force us to use the second smallest size Stratix II device (the EP2S30). For the Stratix III generation ($65nm$ node), Altera has made a substantial effort to increase the number of DSP blocks. This lets us synthesize the design in the smallest Stratix III of the series (the EPS3SE50). Other point of comparison are given in array 2, for high speed implementations targeting 128-bit security, but on other technologies : ASIC [14], and $x86$ CPU [26, 3]. We only give the results for BN_{126}. Results for BN_{128} can be deduced from it, since hardware remains the same (only precomputed values and microcode in the sequencer change).

6.3 Beyond 128 bits

In this section we propose the first high speed implementations of pairing with 192-bit security, according to the NIST recommendations [27]. We use BN curves, even it is not clear that this is the best choice (we are indeed handicapped by the too small embedding degree of BN, and therefore forced to

Fig. 2. Overall results and comparison

	curve	device	chip	freq	size	time(ms)
Ours	BN_{126}	Cyclone II	EP2C35	91	14274 LC	1.94
	BN_{126}	Stratix II	EP2S30	154	4227 A	1.14
	BN_{126}	Stratix III	EP3S50	165	4233 A	1.07
[13]	supersingular over $\mathbb{F}_3$	Virtex IV	xc4vlx25	192	4755 sl.	2.22
[2]	genus 2 supersingular over $\mathbb{F}_2$	Virtex IV	xc4vlx25	220	4646 sl.	3.5
[14]	Barreto-Naehrig	ASIC	130 nm	204	113 kG	2.91
[26]	Barreto-Naehrig	Core 2 duo	-	2400	-	1.87
[3]	Barreto-Naehrig	Core 2 duo	-	2400	-	0.94

use too large curves). We propose here one curve, named BN_{192}, whose $\mathbb{F}_{p^{12}}$ relying tower field is constructed the same way as BN_{126} or BN_{128}. Its parametrization is given by $u = -(2^{160} + 2^{74} + 2^{12} + 1)$. Here are the implementation results :

curve	FPGA	Rowers	area	cycle count	time
BN_{192}	EP3SE50	19	9910 ALM	790010	6.02 ms

7 Conclusion

In this work we demonstrated that RNS is really competitive for pairing computations over FPGA. Thanks to the presence of high speed DSP blocks, we can compute our pairing in almost half the time of the best known FPGA implementations in small characteristic (2 or 3). If we cannot reach the speed of the best software implementations over actual CPU, we have bridged a huge part of the gap between the two approaches. The loss of speed can be advantageously exchanged with a gain in security and power consumption.

Appendix

Algorithm 4: `dbl`, doubling step

Data: $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T) \in E(\mathbb{F}_{p^{12}})$ with X_T, Y_T and $Z_T \in \mathbb{F}_{p^2}$, $P = (x_P, y_P) \in E(\mathbb{F}_p)$.
Result: The point $2T$ and the evaluation in P of the equation of the tangent line in T to the curve up to multiplicative factors in $\mathbb{F}_{p^2}$.
begin

$B = Y_T^2, \quad C = 3Z_T^2, \quad D = 2X_TY_T$
$F = \mathbf{i}B - 3C, \quad G = \mathbf{i}B + 3C, \quad H = 3C, \quad t_3 = B + 4\mathbf{i}C, \quad A = X_T^2, \quad E = 2Y_TZ_Y$
$X_{2T} = DF, \quad Y_{2T} = -\mathbf{i}G^2 + 2HC, \quad Z_{2T} = 4BE, \quad t_0 = Fy_P, \quad t_1 = -3Ax_P$
return $(X_{2T}\boldsymbol{\gamma}^2, Y_{2T}\boldsymbol{\gamma}^3, Z_{2T}), t_0 + t_1\boldsymbol{\gamma} + t_3\boldsymbol{\gamma}^3$

end

Algorithm 5: `add`, addition step

Data: $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T) \in E(\mathbb{F}_{p^{12}})$ with X_T, Y_T and $Z_T \in \mathbb{F}_{p^2}$,
$Q = (x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3) \in E(\mathbb{F}_{p^{12}})$, $P = (x_P, y_P) \in E(\mathbb{F}_p)$.
Result: The point $T + Q$ and the evaluation in P of the equation of the line passing through T and Q up to multiplicative factors in $\mathbb{F}_{p^2}$.
begin

$E = x_QZ_T - X_T, \quad F = y_QZ_T - Y_T$
$E_2 = E^2, \quad F_2 = F^2$
$A = F_2Z_T - 2X_TE_2 - EE_2, \quad B = X_TE_2, \quad E_3 = EE_2$
$X_{T+Q} = AE, \quad Z_{T+Q} = Z_TE_3, \quad t_3 = Fx_Q - Ey_Q$
$Y_{T+Q} = F(B - X_{T+Q}) - y_QE_3, \quad t_0 = Ey_P, \quad t_1 = -Fx_P$
return $(X_{T+Q}\boldsymbol{\gamma}^2, Y_{T+Q}\boldsymbol{\gamma}^3, Z_{T+Q}), t_0 + t_1\boldsymbol{\gamma} + t_3\boldsymbol{\gamma}^3$

end

Algorithm 6: `hard-part`, hard part of the final exponentiation according [29]

Data: $f \in \mathbb{F}_{p^{12}}$ of order $p^4 - p^2 + 1$, $x = |u|$
Result: $f^{(p^4-p^2+1)/\ell}$ with p and ℓ as in 2.1.
begin

computation of the y_i
$y_0 \leftarrow f^p f^{p^2} f^{p^3}$, $y_1 \leftarrow f^x$, $y_3 \leftarrow y_1^x$, $y_5 \leftarrow y_3^x$, $y_4 \leftarrow y_5^p$, $y_6 \leftarrow y_4y_5 \left(= f^{x^3}\left(f^{x^3}\right)^p\right)$
$y_5 \leftarrow y_3^p$, $y_2 \leftarrow y_5^{-1}$, $y_4 \leftarrow y_1y_2 \left(= f^x/\left(f^{x^2}\right)^p\right)$
$y_2 \leftarrow y_5^p \left(= \left(f^{x^2}\right)^{p^2}\right)$, $y_5 \leftarrow y_3^{-1} \left(= 1/f^{x^2}\right)$, $y_3 \leftarrow y_1^p\,((m^x)^p)$, $y_1 \leftarrow f^{-1}$
multi-addition chain for computing $y_0.y_1^2.y_2^6.y_3^{12}.y_4^{18}.y_5^{30}.y_6^{36}$
$t_0 \leftarrow y_6^2$, $t_0 \leftarrow t_0y_4$, $t_0 \leftarrow t_0y_5$, $t_1 \leftarrow y_3y_5$, $t_1 \leftarrow t_1t_0$, $t_0 \leftarrow t_0y_2$, $t_1 \leftarrow t_1^2$
$t_1 \leftarrow t_1t_0$, $t_1 \leftarrow t_1^2$, $t_0 \leftarrow t_1y_1$, $t_1 \leftarrow t_1y_0$, $t_0 \leftarrow t_0^2$, $t_0 \leftarrow t_0t_1$
return t_0

end

Fig. 3. General architecture of a Cox Rower with the triple port RAM

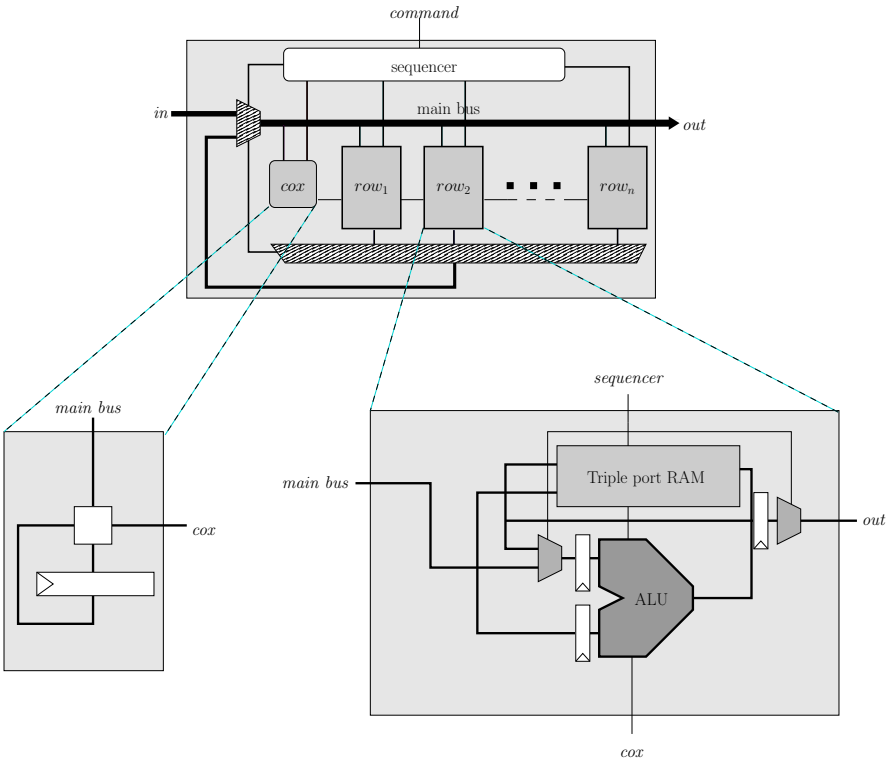
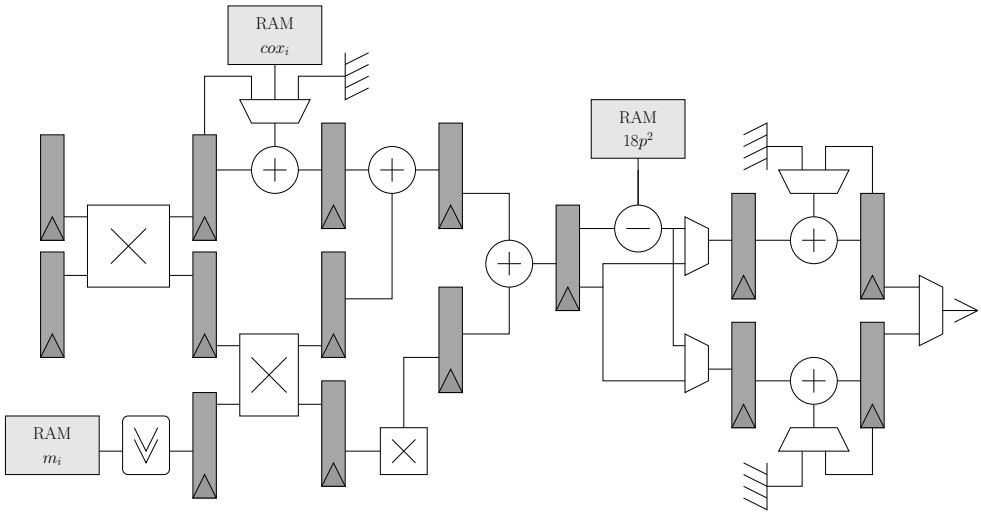


Fig. 4. design of the pipeline



References

1. Altera web site. http://www.altera.com.
2. Diego F. Aranha, Jean-Luc Beuchat, Jérémie Detrey, and Nicolas Estibals. Optimal eta pairing on supersingular genus-2 binary hyperelliptic curves. Cryptology ePrint Archive, Report 2010/559, 2010. http://eprint.iacr.org/.
3. Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio Lpez. Faster explicit formulas for computing pairings over ordinary curves. Cryptology ePrint Archive, Report 2010/526, 2010. http://eprint.iacr.org/.
4. J.C. Bajard, S. Duquesne, and M. Ercegovac. Combining leak–resistant arithmetic for elliptic curves defined over $\mathbb{F}_p$ and rns representation. Cryptology ePrint Archive, Report 2010/311, 2010. http://eprint.iacr.org/.
5. Jean-Claude Bajard, Laurent-Stéphane Didier, and Peter Kornerup. An rns montgomery modular multiplication algorithm. *IEEE Transactions on Computers*, 47(7):766–776, 1998.
6. P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. volume 2442 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2002.
7. P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected areas in cryptography–SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
8. J.L. Beuchat, J.E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves.
9. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology–CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
10. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
11. C. Costello, T. Lange, and M. Naehrig. Faster pairing computations on curves with high-degree twists. In *Public Key Cryptography–PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 224–242. Springer, 2010.
12. S. Duquesne. Rns arithmetic in $\mathcal{U}_{p^k}$ and application to fast pairing computation. Cryptology ePrint Archive, Report 2010/555, 2010. http://eprint.iacr.org/, to appear in Journal of Mathematical Cryptology.
13. Nicolas Estibals. Compact hardware for computing the tate pairing over 128-bit-security supersingular curves. In *Pairing*, volume 6487 of *Lecture Notes in Computer Science*, pages 397–416, 2010.
14. Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Faster-arithmetic for cryptographic pairings on barreto-naehrig curves. In *Cryptographic Hardware and Embedded Systems–CHES 2009*, pages 240–253, 2009.
15. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.
16. G. Frey and H.G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of computation*, 62(206):865–874, 1994.
17. R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. *Public Key Cryptography–PKC 2010*, 6056:209–223, 2010.
18. Nicolas Guillermin. A high speed coprocessor for elliptic curve scalar multiplications over $\{F\}_p$.
19. D. Hankerson, A. Menezes, and M. Scott. *Software implementation of pairings*, volume 2 of *Cryptology and Information Security Series*, pages 188–206. IOS Press, m. joye and g. neven edition, 2009.
20. A. Joux. A one round protocol for tripartite Diffie–Hellman. *Journal of Cryptology*, 17(4):263–276, 2004.
21. K. Karabina. Squaring in cyclotomic subgroups. 2010. http://eprint.iacr.org/.
22. Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 523–538. Springer Berlin / Heidelberg, 2000.
23. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. *Cryptography and coding*, 3796:13–36, 2005.
24. A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39(5):1639–1646, 1993.
25. V.S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
26. Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123, 2010.
27. National Institute of Standard and technology. Key management, 2007. http://csrc.nist.gov/groups/ST/toolkit/key_management.html.
28. G.C.C.F. Pereira, M.A.S. Jr, M. Naehrig, and P.S.L.M. Barreto. A Family of Implementation-Friendly BN Elliptic Curves.
29. M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez, and E. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. *Pairing-Based Cryptography–Pairing 2009*, 5671:78–88, 2009.
30. F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.