Faster $\mathbb{F}_p$-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves *

Junfeng Fan, Frederik Vercauteren**, and Ingrid Verbauwhede

Katholieke Universiteit Leuven, ESAT/SCD-COSIC,
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{jfan, fvercauteren, iverbauwhede}@esat.kuleuven.be

Abstract. This paper describes a new method to speed up $\mathbb{F}_p$-arithmetic for Barreto-Naehrig (BN) curves. We explore the characteristics of the modulus defined by BN curves and choose curve parameters such that $\mathbb{F}_p$ multiplication becomes more efficient. The proposed algorithm uses Montgomery reduction in a polynomial ring combined with a coefficient reduction phase using a pseudo-Mersenne number. With this algorithm, the performance of pairings on BN curves can be significantly improved, resulting in a factor 5.4 speed-up compared with the state-of-the-art hardware implementations. Using this algorithm, we implemented a pairing processor in hardware, which runs at 204 MHz and finishes one ate and R-ate pairing computation over a 256-bit BN curve in 4.22 ms and 2.91 ms, respectively.

Keywords: Pairings, BN curves, Modular reduction

1 Introduction

A bilinear pairing is a map $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are typically additive groups and $\mathbb{G}_T$ is a multiplicative group and the map is linear in each component. Many pairings used in cryptography such as the Tate pairing [1], ate pairing [11], and R-ate pairing [13] 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 a substantial impact on the security and performance of a pairing. For example, the underlying field, the type of curve, the order of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ should be carefully chosen such that it offers sufficient security, but still is efficient to compute. In this paper,

* This work was supported by research grants of Katholieke Universiteit Leuven (OT/06/40) and FWO projects (G.0300.07), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the EU IST FP6 projects (ECRYPT) and by the IBBT-QoE project of the IBBT.

** Postdoctoral Fellow of the Research Foundation - Flanders (FWO).

we focus on efficient implementation of pairings over BN curves [17]. BN curves are defined over $\mathbb{F}_p$ where $p = 36\bar{t}^4 + 36\bar{t}^3 + 24\bar{t}^2 + 6\bar{t} + 1$ for $\bar{t} \in \mathbb{Z}$ such that p is prime. In this paper, we propose a new modular multiplication algorithm for BN curves. We show that when choosing $\bar{t} = 2^m + s$, where s is a reasonably *small* number, the modular multiplication in $\mathbb{F}_p$ can be substantially improved. Existing techniques to speed up arithmetic in extension fields (see [7, 6] for fast operation in $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$) can be used on top of it. The proposed modular reduction algorithm and parameters for BN curves result in a significant improvement on the performance of ate and R-ate pairing.

The remainder of the paper is organized as follows. In Sect. 2 we review cryptographic pairings and their computation. In Sect. 3 we present a new modular multiplication algorithm and compare its complexity with known algorithms. The details of the hardware implementation and results are given in Sect. 4 and Sect. 5, respectively. We conclude the paper in Sect. 6.

2 Previous works

2.1 Bilinear Pairings

Let $\mathbb{F}_p$ be a finite field and let $E(\mathbb{F}_p)$ be an elliptic curve defined over $\mathbb{F}_p$. Let r be a large prime dividing $\#E(\mathbb{F}_p)$. Let k be the embedding degree of $E(\mathbb{F}_p)$ with respect to r, namely, the smallest positive integer k such that $r|p^k - 1$. We use $E(\mathbb{K})[r]$ to denote the $\mathbb{K}$-rational r-torsion group of the curve for any finite field $\mathbb{K}$. For $P \in E(\mathbb{K})$ and an integer s, let $f_{s,P}$ be a $\mathbb{K}$-rational function with divisor

$$(f_{s,P}) = s(P) - ([s]P) - (s-1)\mathcal{O},$$

where $\mathcal{O}$ is the point at infinity. This function is also known as Miller function [14, 15].

Let $\mathbb{G}_1 = E(\mathbb{F}_p)[r]$, $\mathbb{G}_2 = E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k})$ and $\mathbb{G}_3 = \mu_r \subset \mathbb{F}^*_{p^k}$ (the r-th roots of unity), then the reduced Tate pairing is a well-defined, non-degenerate, bilinear pairing. Let $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, then the reduced Tate pairing of P, Q is computed as

$$e(P, Q) = (f_{r,P}(Q))^{(p^k-1)/r}.$$

The ate pairing is similar but with different $\mathbb{G}_1$ and $\mathbb{G}_2$. Here we define $\mathbb{G}_1 = E(\mathbb{F}_p)[r]$ and $\mathbb{G}_2 = E(\mathbb{F}_{p^k})[r] \cap \mathrm{Ker}(\pi_p - [p])$, where π_p is the Frobenius endomorphism. Let $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and let t_r be the trace

Algorithm 1 Computing the R-ate pairing on BN curves [7]

Input: $P \in E(\mathbb{F}_p)[r]$, $Q \in E(\mathbb{F}_{p^k})[r] \bigcap Ker(\pi_p - [p])$ and $a = 6\bar{t} + 2$.
Output: $R_a(Q, P)$.
1: $a = \sum_{i=0}^{L-1} a_i 2^i$.
2: $T \leftarrow Q$, $f \leftarrow 1$.
3: **for** $i = L - 2$ downto 0 **do**
4: $T \leftarrow 2T$.
5: $f \leftarrow f^2 \cdot l_{T,T}(P)$.
6: **if** $a_i = 1$ **then**
7: $T \leftarrow T + Q$.
8: $f \leftarrow f \cdot l_{T,Q}(P)$.
9: **end if**
10: **end for**
11: $f \leftarrow (f \cdot (f \cdot l_{aQ,Q}(P))^p \cdot l_{\pi(aQ+Q),aQ}(P))^{(p^k-1)/r}$.
Return f.

of Frobenius of the curve, then the ate pairing is also well-defined, non-degenerate bilinear pairing, and can be computed as

$$a(Q, P) = (f_{t_r-1,Q}(P))^{(p^k-1)/r}.$$

The R-ate pairing is a generalization of the ate pairing. For the same choice of $\mathbb{G}_1$ and $\mathbb{G}_2$ as for the ate pairing, the R-ate pairing on BN curves is defined as

$$Ra(Q, P) = (f \cdot (f \cdot l_{aQ,Q}(P))^p \cdot l_{\pi(aQ+Q),aQ}(P))^{(p^k-1)/r},$$

where $a = 6\bar{t} + 2$, $f = f_{a,Q}(P)$ and $l_{A,B}$ denotes the line through point A and B.

Due to limited space, we only describe the algorithm to compute the R-ate pairing. The algorithms for Tate and ate pairings are similar, and can be found in [7].

2.2 Choice of Curve Parameters

The most important parameters for cryptographic pairings are the underlying finite field, the order of the curve, the embedding degree, and the order of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$. These parameters should be chosen such that the best exponential time algorithms to solve the discrete logarithm problem (DLP) in $\mathbb{G}_1$ and $\mathbb{G}_2$ and the sub-exponential time algorithms to solve the DLP in $\mathbb{G}_T$ take longer than a chosen security level. In this paper, we will use the 128-bit symmetric key security level.

Barreto and Naehrig [17] described a method to construct pairing-friendly elliptic curves over a prime field with embedding degree 12. The

finite field, trace of Frobenius and order of the curve are defined by the following polynomial families:

$$\begin{aligned} p(t) &= 36t^4 + 36t^3 + 24t^2 + 6t + 1, \\ t_r(t) &= 6t^2 + 1, \\ n(t) &= 36t^4 + 36t^3 + 18t^2 + 6t + 1. \end{aligned}$$

The curve is defined as $E : y^2 = x^3 + v$ for some $v \in \mathbb{F}_p$. The choice of $\bar{t}$ must meet the following requirements: both $p(\bar{t})$ and $n(\bar{t})$ must be prime and $\bar{t}$ must be large enough to guarantee a chosen security level. For the efficiency of pairing computation, $\bar{t}$, $p(\bar{t})$ and $t_r(\bar{t})$ should have small Hamming-weight.

For example, [7] suggested to use $\bar{t} =$ 0x6000000000001F2D, which is also used in [10] and [12]. With this parameter, pairings defined over $p(\bar{t})$ achieves 128-bit security.

Table 1. Selection of $\bar{t}$ for BN curves in [7]

$\bar{t}$	HW$(6\bar{t}+2)$	HW(t_r)	HW(p)	log$(2,p)$
0x6000000000001F2D	9	28	87	256

2.3 Multiplication in $\mathbb{F}_p$

We briefly recall the techniques for integer multiplication and reduction. Given a modulus $p < 2^n$ and an integer $c < 2^{2n}$, the following algorithms can be used to compute $c \bmod p$.

Barrett reduction The Barrett reduction algorithm [2] uses a precomputed value $\mu = \lfloor \frac{2^{2n}}{p} \rfloor$ to help estimate $\frac{c}{p}$, thus integer division is avoided. Dhem [8] proposed an improved Barrett modular multiplication algorithm which has a simplified final correction.

Montgomery reduction The Montgomery reduction method [16] precomputes $p' = -p^{-1} \bmod r$, where r is normally a power of two. Given c and p, it generates q such that $c + qp$ is a multiple of r. As a result, $(c + qp)/r$ is just a shift operation. Algorithm 2 shows both Barrett and Montgomery multiplication algorithms.

Chung-Hasan reduction In [4, 5], Chung and Hasan proposed an efficient reduction method for low-weight polynomial form moduli $p = f(\bar{t}) = \bar{t}^n + f_{n-1}\bar{t}^{n-1} + .. + f_1\bar{t} + f_0$, where $|f_i| \leq 1$. The modular multiplication is shown in Alg. 3.

Algorithm 2 Digit-serial modular multiplication algorithm.

Barrett [8]	**Montgomery [16]**
Input: $a = (a_{n-1}, .., a_0)_d$, $b = (b_{n-1}, .., b_0)_d$, $p = (p_{n-1}, .., p_0)_d$,$0 \le a, b < p$, $2^{(n-1)w} \le p < 2^{nw}$, $d = 2^w$. Precompute $\mu = \lfloor d^{n+3}/p \rfloor$. **Output:** $c = ab \bmod p$.	**Input:** $a = (a_{n-1}, .., a_0)_d$, $b = (b_{n-1}, .., b_0)_d$, $p = (p_{n-1}, .., p_0)_d$, $0 \le a, b < p$, $r = d^n$, Precompute $p' = -p^{-1} \bmod d$, $d = 2^w$. **Output:** $c = abr^{-1} \bmod p$.
1: $c \leftarrow 0$.	1: $c \leftarrow 0$.
2: **for** $i = n - 1$ downto 0 **do**	2: **for** $i = 0$ to $n - 1$ **do**
3: $c \leftarrow c \cdot d + a \cdot b_i$.	3: $c \leftarrow c + ab_i$.
4: $\hat{q} \leftarrow \lfloor (\lfloor c/d^{n-2} \rfloor \cdot \mu)/2^{w+5} \rfloor$.	4: $u \leftarrow c \bmod d$, $q \leftarrow (up') \bmod d$.
5: $c \leftarrow c - \hat{q} \cdot p$.	5: $c \leftarrow (c + qp)/d$.
6: **end for**	6: **end for**
7: **if** $c \ge p$ **then**	7: **if** $c \ge p$ **then**
8: $c \leftarrow c - p$.	8: $c \leftarrow c - p$.
9: **end if**	9: **end if**
Return c.	**Return** c.

Algorithm 3 Chung-Hasan multiplication algorithm [4].

Input: positive integers $a = \sum_{i=0}^{n-1} a_i t^i$, $b = \sum_{i=0}^{n-1} b_i t^i$, modulus $p = f(t) = t^n + f_{n-1}t^{n-1} + .. + f_1 t + f_0$.
Output: $c(t) = a(t)b(t) \bmod p$.

1: **Phase I: Polynomial Multiplication**
2: $c(t) \leftarrow a(t)b(t)$.
3: **Phase II: Polynomial Reduction**
4: **for** $i = 2n - 2$ down to n **do**
5: $c(t) \leftarrow c(t) - c_i f(t) t^{i-n}$.
6: **end for**
Phase III: Coefficient Reduction
7: $c_n \leftarrow \lfloor c_{n-1}/\bar{t} \rfloor$, $c_{n-1} \leftarrow c_{n-1}$- $c_n \bar{t}$.
8: $c(t) \leftarrow c(t) - c_n f(t) t$.
9: **for** $i = 0$ to $n - 1$ **do**
10: $q_i \leftarrow \lfloor c_i/\bar{t} \rfloor$, $r_i \leftarrow c_i$- $q_i \bar{t}$.
11: $c_{i+1} \leftarrow c_{i+1} + q_i$, $c_i \leftarrow r_i$.
12: **end for**
13: $c(t) \leftarrow c(t) - q_n f(t) t$.

Return $c(t)$.

The polynomial reduction phase is rather efficient since $f(t)$ is monic, making the polynomial long division (step 3) simple. Barrett reduction is used to perform divisions required in Phase III. The overall performance is more efficient than traditional Barrett or Montgomery reduction algorithm [4]. In [5], this algorithm is further extended to monic polynomials

with $|f_i| \leq s$ where $s \in (0, \bar{t})$ is a small number. Note that the polynomial reduction phase is efficient only when $f(t)$ is monic.

3 Fast modular reduction algorithm for BN curves

Instead of using a general modular reduction algorithm such as Montgomery or Barrett algorithm, we explore the special characteristics of the prime p. Note that the polynomial $p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1$ defined by BN is not monic, but has the following characteristics:

1. $p(t)$ has small coefficients.
2. $p^{(-1)}(t) = 1 \bmod t$.

The second condition implies via Hensel's lemma that $p^{(-1)}(t) \bmod t^n$ has integer coefficients. This suggests that multiplication and reduction with Montgomery's algorithm in the polynomial ring could be efficient. We first present a modular multiplication algorithm for polynomial form primes that satisfy $p^{(-1)}(t) = 1 \bmod t$ and then apply this method to BN curves.

3.1 Hybrid Modular Multiplication

Algorithm 4 describes a modular multiplication algorithm for polynomial form moduli. The algorithm is composed of three phases, i.e. polynomial multiplication (step 3), polynomial reduction (step 4-6), and coefficient reduction phase (step 9). Note that we present the algorithm in a digit-serial manner. The polynomial reduction uses the Montgomery reduction, while the coefficient reduction uses division. We call this algorithm Hybrid Modular Multiplication (HMM).

Note that algorithm 4 works for any irreducible polynomial $p(t)$ satisfying the condition $p^{(-1)}(t) = 1 \bmod t$ or equivalently, $p(t) = 1 \bmod t$. It can also be easily modified to support $p(t)$ satisfying $p^{(-1)}(t) = -1 \bmod t$.

Algorithm 4 requires division by $\bar{t}$ in both step 4 and step 9. Like Chung-Hasan's algorithm, division can be performed with the Barrett reduction algorithm [4]. However, the complexity of division can be reduced if $\bar{t}$ is a pseudo-Mersenne number. Algorithm 5 transfers division by $\bar{t}$ to multiplication by s for $\bar{t} = 2^m + s$ where s is small.

3.2 Modular Multiplication for BN Curves

In order to apply Alg. 4 and Alg. 5 to BN curves, we select $\bar{t} = 2^m + s$ where s is small. Note that any choice of $\bar{t}$ which makes p and n primes

Algorithm 4 Hybrid Modular Multiplication Algorithm

Input: $a(t) = \sum_{i=0}^{n-1} a_i t^i$, $b(t) = \sum_{i=0}^{n-1} b_i t^i$, and modulus $p(t) = \sum_{i=1}^{n-1} p_i t^i + 1$.
Output: $r(t) = a(t)b(t)t^{-n} \bmod p(t)$.

1: $c(t)(= \sum_{i=0}^{n-1} c_i t^i) \leftarrow 0$.
2: **for** $i = 0$ to $n-1$ **do**
3: $\quad c(t) \leftarrow c(t) + a(t)b_i$.
4: $\quad \mu \leftarrow c_0$ div $\bar{t}$, $\gamma \leftarrow c_0 \bmod \bar{t}$.
5: $\quad g(t) \leftarrow (p_{n-1}t^{n-1} + .. + p_1 t + 1)(-\gamma)$.
6: $\quad c(t) \leftarrow (c(t) + g(t))/t + \mu$.
7: **end for**
8: **for** $i = 0$ to $n-2$ **do**
9: $\quad c_{i+1} \leftarrow c_{i+1} + (c_i$ div $\bar{t})$, $c_i \leftarrow c_i \bmod \bar{t}$.
10: **end for**

Return $r(t) \leftarrow c(t)$.

Algorithm 5 Division by $\bar{t} = 2^m + s$

Input: a, $\bar{t} = 2^m + s$ with $0 < s < 2^{\lfloor k/2 \rfloor}$.
Output: μ and γ with $a = \mu t + \gamma$, $|\gamma| < \bar{t}$.

1: $\mu \leftarrow 0$, $\gamma \leftarrow a$.
2: **while** $|\gamma| \geq \bar{t}$ **do**
3: $\quad \rho \leftarrow \gamma$ div 2^m, $\gamma \leftarrow \gamma \bmod 2^m$.
4: $\quad \mu \leftarrow \mu + \rho$, $\gamma \leftarrow \gamma - s\rho$.
5: **end while**

Return μ, γ.

of the required size will suffice. As such we can choose $\bar{t} = 2^m + s$ where s is small; an example is shown in Table 2.

Table 2. Selection of $\bar{t} = 2^m + s$ for BN curves

$\bar{t}$	HW$(6\bar{t}+2)$	HW(t_r)	HW(p)	$\log(2,p)$
$2^{63} + 2^9 + 2^8 + 2^6 + 2^4 + 2^3 + 1$	6	20	68	257

With $\bar{t} = 2^m + s$ as shown in Table 2, Algorithm 6 describes a modular multiplication algorithm for BN curves which we call HMMB.

The following lemma provides bounds on the input value such that Algorithm 6 gives a bounded output. The proof is in the appendix.

Lemma 1. *Given $\bar{t} = 2^m + s$ and $\xi = (36s + 1) < 2^{m/2-7}$ (i.e. $m \geq 26$), if the input $a(t)$ and $b(t)$ satisfy*

$$0 \leq |a_i|, |b_i| < 2^{m/2}, \quad i = 4,$$
$$0 \leq |a_i|, |b_i| < 2^{m+1}, \quad 0 \leq i \leq 3,$$

Algorithm 6 Hybrid Modular Multiplication Algorithm for BN curves

Input: $a(t) = \sum_{i=0}^{4} a_i t^i$, $b(t) = \sum_{i=0}^{4} b_i t^i$. $p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1$, $p^{-1}(t) = 1 \bmod t$, $\bar{t} = 2^m + s$.
Output: $r(t) = a(t)b(t)t^{-5} \bmod p(t)$.

1: $c(t)(= \sum_{i=0}^{4} c_i t^i) \leftarrow 0$.
2: **for** $j = 0$ to 4 **do**
3: $\quad c(t) \leftarrow c(t) + a(t)b_j$.
4: $\quad \mu \leftarrow c_0$ div 2^m, $\gamma \leftarrow (c_0 \bmod 2^m) - s\mu$.
5: $\quad g(t) \leftarrow (36t^4 + 36t^3 + 24t^2 + 6t + 1)(-\gamma)$.
6: $\quad c(t) \leftarrow (c(t) + g(t))/t + \mu$.
7: **end for**
8: **for** $i = 0$ to 3 **do**
9: $\quad \mu \leftarrow c_i$ div 2^m, $\gamma \leftarrow (c_i \bmod 2^m) - s\mu$.
10: $\quad c_{i+1} \leftarrow c_{i+1} + \mu$, $c_i \leftarrow \gamma$.
11: **end for**
12: Repeat step 8-11.

Return $r(t) \leftarrow c(t)$.

then $r(t)$ calculated by Alg. 6 satisfies

$$\begin{array}{ll} 0 \le |r_i| < 2^{m/2}, & i = 4, \\ 0 \le |r_i| < 2^{m+1}, & 0 \le i \le 3. \end{array}$$

This algorithm is suitable for high performance implementation on multi-core systems. One can see that the first loop of HMMB algorithm can be easily parallelized. This is an intrinsic advantage of this algorithm, i.e. no carry propagation occurs during polynomial multiplication. The coefficient reduction phase can also be parallelized. We modify the last two loops of the HMMB algorithm and give a parallel version.

$$\begin{array}{lll} \textbf{loop1} & \mu_i \leftarrow c_i \text{ div } 2^m, \gamma_i \leftarrow (c_i \bmod 2^m) - s\mu_i & 0 \le i \le 3. \\ & c_i \leftarrow \gamma_i + \mu_{i-1} & 1 \le i \le 3. \\ & c_0 \leftarrow \gamma_0, c_4 \leftarrow \mu_3. & \\ \textbf{loop2} & \mu_i \leftarrow c_i \text{ div } 2^m, \gamma_i \leftarrow (c_i \bmod 2^m) - s\mu_i & 0 \le i \le 3. \\ & c_i \leftarrow \gamma_i + \mu_{i-1} & 1 \le i \le 3. \\ & c_0 \leftarrow \gamma_0, c_4 \leftarrow c_4 + \mu_3. & \end{array}$$

From the proof of Lemma 1 we know $c_i < 5 \cdot 2^{2m+3}$ for $0 \le i \le 4$. Thus, after **loop1**, we have

$$|c_i| = |\mu_{i-1}| + |\gamma_i| < 5 \cdot 2^{m+3} + 5 \cdot 2^{m+3}s + 2^m < s2^{m+6}.$$

Furthermore, after **loop2**, we have

$$|c_i| = |\mu_{i-1}| + |\gamma_i| < s2^6 + 2^6 s^2 + 2^m < 2^{m+1}.$$

3.3 Complexity of Algorithm 6

We compare the complexity of Alg. 6 with Montgomery's and Barrett's algorithm for 256-bit BN curves. We assume a digit-serial method is used with digit-size 64-bit. Note that Alg. 6 requires four 64-bit words together with one 32-bit word to represent a 256-bit integer.

For Montgomery multiplication, nine 64x64 multiplications are required in each iteration, resulting in 36 subword multiplications in total. Barrett multiplication has the same complexity as Montgomery algorithm.

For HMMB, in the first loop four 64x64 and one 32x64 multiplications are required in step 3, one $\lceil \log_2(s) \rceil \text{x} \lceil \log_2(\mu) \rceil$ multiplication is required in step 4. The last iteration takes four 32x64 and one 32x32 multiplications. In total, the first loop takes one 32x32, eight 32x64, sixteen 64x64, and five $\lceil \log_2(s) \rceil \text{x} \lceil \log_2(\mu) \rceil$ multiplications, where $\mu < 2^{k+6}$ as shown in the proof of Lemma 1. Note that $p(t)\gamma$ can be performed with addition and shift operation, e.g. $36\gamma = 2^5\gamma + 2^2\gamma$.

The coefficient reduction phase requires eight $\lceil \log_2(s) \rceil \text{x} \lceil \log_2(\mu) \rceil$ multiplications. From the proof of Lemma 1 we know that $c_i < 5 \cdot 2^{2k+3}$, thus $\mu < 2^{k+6}$ in the first **for** loop (step 8-10). In the second **for** loop (step 12), as shown in the end of section 3.2, we have $\mu < s2^6$.

Table 3 compares the number of multiplications required by the Barrett, Montgomery and the HMMB algorithm. Compared to Barrett and Montgomery reduction, HMMB has much lower complexity. One can see that $s\mu$ can be efficiently computed if s is small (see Table 2). Especially, if s is of low Hamming-weight, $s\mu$ can be performed with shift and addition operations.

Table 3. Complexity comparison of different modular multiplication algorithms

Algorithm	32x32	32x64	64x64	$\lceil \log_2(s) \rceil \text{x} \lceil \log_2(\mu) \rceil$
Barrett			36	
Montgomery			36	
HMMB	1	8	16	13

4 Hardware Design of Pairings

As a verification of the efficiency of the HMMB algorithm, we made a hardware implementation of the ate and R-ate pairing using this algo-

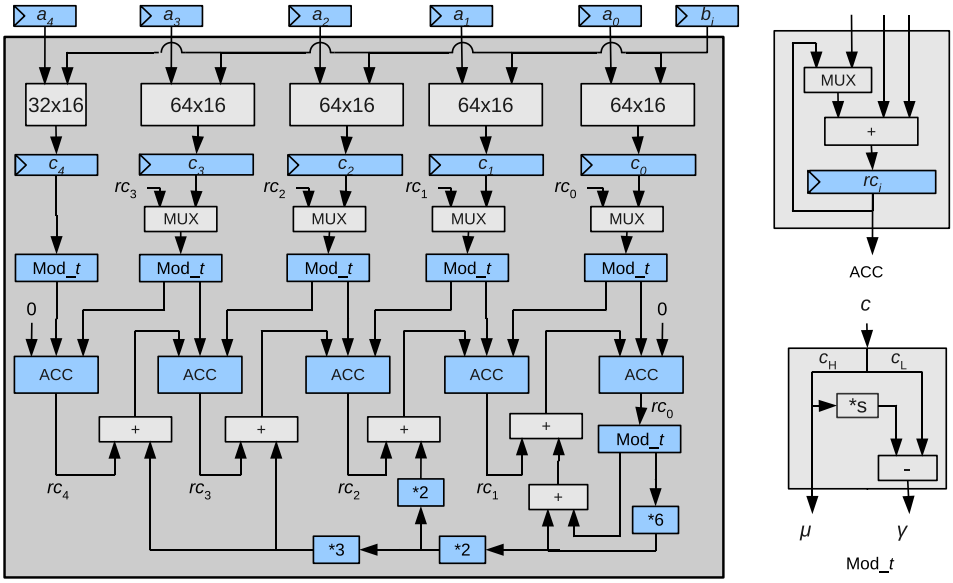

Fig. 1. $\mathbb{F}_p$ multiplier using algorithm HMMB.

rithm. We chose $\bar{t} = 2^{63} + s$, where s=2^9+2^8+$2^6 + 2^4 + 2^3 + 1$. With this setting, the implementation achieves 128-bit security.

4.1 Multiplier

Figure 1 shows the realization of the HMMB algorithm. A row of multipliers is used to carry out step 3, namely, $a_i b_j$ for $0 \leq i \leq 4$. We used a 64x16 bit multiplier, thus four cycles are required for each iteration. One can adjust the size of multiplier for different design purposes, i.e. high clock frequency, small area and so on. The partial product is then reduced by the "Mod_t" component. The "Mod_t" component, which is composed of a multiplier and a subtracter, generates μ and γ from c_i, namely, $\mu \leftarrow c_i$ div 2^m and $\gamma \leftarrow (c_i \bmod 2^m) - s\mu$. Note that the "Mod_$t$" component below rc_0 is slightly different, where $\gamma \leftarrow (s\mu - (rc_0 \bmod 2^m))$.

The dataflow of this implementation is slightly different from that in Alg. 6. Instead of performing coefficient reduction in the end, we reduce the coefficients before polynomial reduction. This reduces the length of c_i and rc_i in Fig. 1, and one instead of two coefficient reduction loop is required in the end. The "Mod_t" components are reused to perform the final reduction loop. After that, $r(t)$ is ready in the accumulators.

4.2 Pairing Processor Architecture

Using the multiplier described above, we built a pairing processor. Figure 2 shows the block diagram of the processor. It consists of a micro-controller, a program ROM, an $\mathbb{F}_p$ multiplier and adder/subtracter, a register file and an IO buffer. The data is stored in a 64-bit single port RAM. The program ROM contains subroutines that are used in Miller's loop, such as point addition, point doubling, line evaluation, multiplication in $\mathbb{F}_{p^{12}}$, and so on. The micro-controller realizes Miller's loop by calling the corresponding sub-routines.

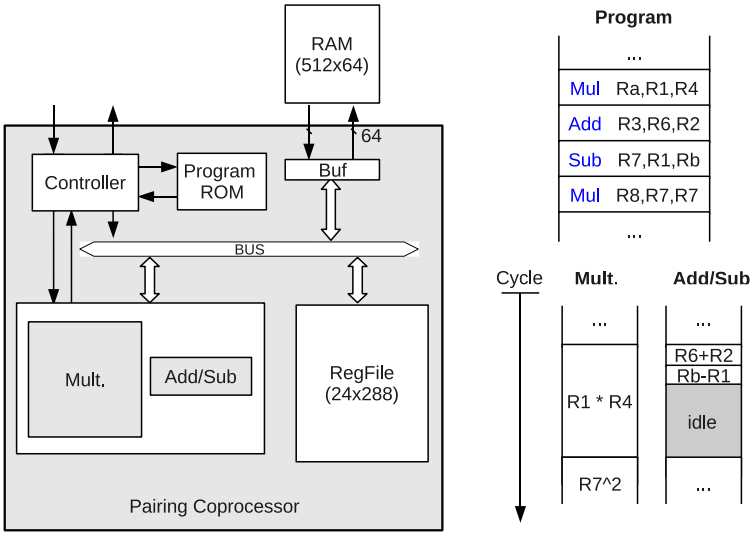

Fig. 2. Block diagram of the system architecture.

The ALU is able to execute multiplication and addition/subtraction instructions in parallel. A simple example is shown in Fig. 2. When performing the `mul` operation, the micro-controller fetches the next instruction and checks if it is an `add` or `sub` instruction. If it is, then it is executed in parallel if there is no data dependency on the ongoing `mul` instruction. Table 4 gives the number of clock cycles that are required for each sub-routine and pairing.

5 Implementation Results

The whole system is synthesized using 130 nm standard cell library. It can run at a maximum frequency of 204 MHz. The pairing processor alone

Table 4. Number of clock cycles required by different subroutines

	2T	T+Q	$l_{T,T}(P)$	$l_{T,Q}(P)$	f^2	$f \cdot l$	$f^{(p^k-1)/r}$	ate	R-ate
#Cycles	574	984	422	260	1541	1239	281558	861724	592976

uses around 183 kGates, including 70 kGates used by Register File and 25 kGates used by controller and Program ROM. It finishes one ate and R-ate pairing computation in 4.22 ms and 2.91 ms, respectively. Table 5 compares the result with the state-of-the-art implementations.

Table 5. Performance comparison of software and hardware implementations of pairing

Design	Pairing	Security [bit]	Platform	Area	Frequency [MHz]	Performance [ms]
this design	ate	128	130 nm ASIC	183 kGates	204	4.22
	R-ate					2.91
[12]	Tate	128	130 nm ASIC	97 kGates	338	34.4
	ate					22.8
	R-ate					15.8
[10]	ate	128	64-bit core2	-	2400	6.25
	R-ate					4.17
[9]	ate	128	64-bit core2	-	2400	6.01
[18]	η_T over $\mathbb{F}_{2^{239}}$	67	XC2VP100-6	25278 slices	84	0.034
	η_T over $\mathbb{F}_{2^{283}}$	72		37803 slices	72	0.049
[3]	η_T over $\mathbb{F}_{3^{97}}$	66	XC4VLX60-11	18683 slices	N/A	0.0048
	η_T over $\mathbb{F}_{3^{193}}$	89	XC4VLX100-11	47433 slices	N/A	0.010

Kammler *et al.* [12] reported the first, and so far the only, hardware implementation of cryptographic pairings achieving a 128-bit security. They chose $\bar{t}$=0x6000000000001F2D to generate a 256-bit BN curve. The Montgomery algorithm is used for $\mathbb{F}_p$ multiplication. Compared with this design, our implementation is about 5 times faster in terms of R-ate pairing calculation. The main speedup comes from fast modular multiplication in $\mathbb{F}_p$ and larger register file. For an $\mathbb{F}_p$ multiplication, the multiplier shown in Fig. 1 takes 23 cycles excluding memory access, while 68 cycles are required in [12]. Though the area of our design is around 1.9 times larger, the area-latency product is still smaller than that in [12].

The results of software implementations [10, 9] are quite impressive. On an Intel 64-bit core2 processor, R-ate pairing requires only 4.17 ms. The advantages of Intel core2 is that it has a fast multiplier (two full 64-

bit multiplication in 8 cycles) and relatively high clock frequency. Though it takes 16 times more clock cycles (10^7 cycles for R-ate [10]) than our hardware implementation, the overall speed is only 1.4 times lower.

There are also some hardware implementations [18, 3] for η_T pairing over binary or cubic curves. Note that the security achieved using the reported parameters is much lower than 128-bit, which makes a fair comparison difficult.

6 Conclusions

In this paper, we studied a new fast implementation of cryptographic pairings using BN curves. We introduce a new modular multiplication algorithm and a method to select multiplication-friendly parameters. We show that with careful selection of parameters the proposed algorithm has much lower computational complexity than traditional Barrett or Montgomery methods.

As a verification, we also implemented ate and R-ate pairing in hardware using this algorithm. Our results outperform previous hardware implementations by a factor of roughly 5. Note that smaller digit size can be used when targeting a compact hardware implementation. For future work, it is also definitely interesting to see the performance of this algorithm implemented in software. Finally, we remark that the described algorithms also generalize to other pairing friendly finite fields and even more generally, to other types of finite fields.

Acknowledgments

The authors would like to thank the anonymous referees for detailed review. We adequately appreciate their observations and helpful suggestions.

References

1. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology CRYPTO 2002*, pages 354–369, 2002.
2. P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Proc. CRYPTO'86*, pages 311–323, 1986.
3. J. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez. Hardware Accelerator for the Tate Pairing in Characteristic Three Based on Karatsuba-Ofman Multipliers. Cryptology ePrint Archive, Report 2009/122, 2009. http://eprint.iacr.org/.

4. J. Chung and M.A. Hasan. Low-Weight Polynomial Form Integers for Efficient Modular Multiplication. *IEEE Trans. Comput.*, 56(1):44–57, 2007.
5. J. Chung and M.A. Hasan. Montgomery Reduction Algorithm for Modular Multiplication Using Low-Weight Polynomial Form Integers. In *ARITH '07: Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 230–239, Washington, DC, USA, 2007. IEEE Computer Society.
6. R. Dahab, A. Devegili, C. Ó' hÉigeartaigh, and M. Scott. Multiplication and Squaring on Pairing-Friendly Fields. Cryptology ePrint Archive, Report 2006/471. Available from http://eprint.iacr.org.
7. A. Devegili, M. Scott, and R. Dahab. Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In *Pairing-Based Cryptography Pairing 2007*, pages 197–207, 2007.
8. J.-F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards*. PhD thesis, Universite catholique de Louvain, Louvain-la-Neuve, Belgium, 1998.
9. P. Grabher, J. Großchäedl, and D. Page. On Software Parallel Implementation of Cryptographic Pairings. In *Selected Areas in Cryptography – SAC 2008*, pages 34–49. Springer Verlang, LNCS 5381, August 2008.
10. D. Hankerson, A. Menezes, and M. Scott. Software implementation of Pairings. In M. Joye and G. Neven, editors, *Identity-Based Cryptography*, 2008.
11. F. Hess, N.P. Smart, and F. Vercauteren. The Eta Pairing Revisited. *Information Theory, IEEE Transactions on*, 52(10):4595–4602, Oct. 2006.
12. D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, R. Leupers, R. Mathar, and H. Meyr. Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. Cryptology ePrint Archive, Report 2009/056, 2009. Available from http://eprint.iacr.org/.
13. E. Lee, H.-S. Lee, and C.-M. Park. Efficient and Generalized Pairing Computation on Abelian Varieties. Cryptology ePrint Archive, Report 2009/040. Available from http://eprint.iacr.org/.
14. V.S. Miller. Short Programs for Functions on Curves. 1986. Unpublished manuscript, Available at http://crypto.stanford.edu/miller/miller.pdf.
15. V.S. Miller. The Weil Pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
16. P. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.
17. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Proceedings of SAC 2005, volume 3897 of LNCS*, pages 319–331. Springer-Verlag, 2006.
18. C. Shu, S. Kwon, and K. Gaj. FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields. In *Proceedings of IEEE International Conference on Field Programmable Technology (FPT)*, pages 173–180, 2006.

APPENDIX

A: Proof of lemma 1.

Proof. The proof proceeds in two parts. The first part proves a bound on the coefficients of $c(t)$ after Step 7 and the second part analyzes the two coefficient reduction loops (Step 8-12).

Denote $c_{i,j}$ the coefficients of $c(t)$ at the beginning of the j-th iteration, so $c(t)$ in Step 7 has coefficients $c_{i,5}$ (i.e. $j \leq 5$). Let $\Delta = 2^{2m+3}$ then we first show by induction on j that

$$|c_{i,j}| \leq j\Delta\,. \tag{1}$$

Clearly Equation (1) holds for $j = 0$, since $c_{i,0} = 0$. Now assume that (1) holds for j, then we will show the inequality holds for $j + 1$. In Step 3, $c_{i,j}$ increases by maximum 2^{2m+2}. In Step 4, we thus obtain

$$|\mu| \leq 2^{m+2} + \frac{j\Delta}{2^m} \quad \text{and} \quad |\gamma| \leq 2^m + s|\mu|\,.$$

In Step 5, we have $|g_i| \leq 36|\gamma|$, so in Step 6 we finally obtain

$$|c_{i,j+1}| \leq 2^{2m+2} + j\Delta + 36|\gamma| + |\mu| = (j+1)\Delta - 2^{2m+2} + 36|\gamma| + |\mu|\,,$$

so it suffices to prove that $36|\gamma| + |\mu| \leq 2^{2m+2}$. Rewriting this leads to

$$36 \cdot 2^m + (36s+1)|\mu| = 36 \cdot 2^m + \xi|\mu| \leq 2^{2m+2}\,,$$

which concludes the proof of (1).

For $c_{3,5}$ we need to obtain a better bound since the bound on the final r_4 is also smaller. Note that coefficient $c_{3,5}$ is computed as $c_{3,5} = a_4 b_4 + 36\gamma$ where γ can be bounded by $2^m + 2^{m+6}s$. This finally leads to the bound

$$c_{3,5} < 2^m + 36(2^m + 2^{m+5}s) < 37 \cdot 2^m + 2^{m+m/2-1}\,.$$

For the first coefficient reduction step, it is easy to see that for $i = 0, 1, 2$ we have $|\mu| \leq 5 \cdot 2^{m+3} + 5 \cdot 2^3$, so after the first reduction we obtain for $i = 0, \ldots, 3$

$$|c_i| \leq 2^m + s|\mu| < 2^m + (5 \cdot 2^{m+3} + 5 \cdot 2^3)s < 2^{m+6}s\,.$$

For c_3 however, we obtain $|\mu| < 37 + 2^{m/2-1} + 2^6 s$ which becomes c_4.

For the second coefficient reduction step, it is again easy to see that $i = 0, 1, 2$ we have $|\mu| \leq 2^6 s$ and thus $|c_i| \leq 2^m + 2^6 s^2 < 2^{m+1}$. For c_4 we obtain, $c_4 = 37 + 2^{m/2-1} + 2^7 s < 2^{m/2}$, since $m > 26$. □