Efficient Hardware Implementation of $\mathbb{F}_p$-arithmetic for Pairing-Friendly Curves

Junfeng Fan, *Student Member, IEEE,* Frederik Vercauteren,
and Ingrid Verbauwhede, *Senior Member, IEEE*

Abstract—This paper describes a new method to speed up $\mathbb{F}_p$-arithmetic in hardware for pairing-friendly curves, such as the well known Barreto-Naehrig (BN) curves. We explore the characteristics of the modulus defined by these 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. As an application we show that the performance of pairings on BN curves in hardware can be significantly improved, resulting in a factor 2.5 speed-up compared with state-of-the-art hardware implementations.

Index Terms—Pairing-Friendly Curves, Modular reduction

1 Introduction

A bilinear pairing is a map $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ 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 [4], ate pairing [23], R-ate pairing [26] and optimal pairings [22], [33], 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 is still efficient to compute. In this paper, we propose a new modular multiplication algorithm for finite fields used in parametrized families of pairing-friendly elliptic curves. The characteristic p of such fields is given by the evaluation of a polynomial $f(z) \in \mathbb{Z}[z]$ at an integer $\bar{z} \in \mathbb{Z}$. One of the most important examples of such families are the Barreto-Naehrig (BN) curves [32] where one has $p = 36\bar{z}^4 + 36\bar{z}^3 + 24\bar{z}^2 + 6\bar{z} + 1$ for some $\bar{z} \in \mathbb{Z}$ such that p is prime. We show that when choosing $\bar{z} = 2^\tau + s$, 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 [13], [14] for fast arithmetic in $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$) can be used together with our construction. As an application we show how to choose parameters for BN curves and obtain a significant improvement on the performance in hardware of the ate and optimal ate pairing.

• *The authors are with the Department of Electrical Engineering, Katholieke Universiteit Leuven and IBBT, ESAT/SCD-COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium. E-mail: {jfan, fvercaut, iverbauw}@esat.kuleuven.be.*

The remainder of the paper is organized as follows. In Section II we review cryptographic pairings, pairing-friendly curves and modular arithmetic. In Section III we present our new modular multiplication algorithm and compare its complexity with known algorithms. Section IV provides parameters for several families of pairing-friendly curves. As an application, Section V gives in detail an architecture of an $\mathbb{F}_p$-multiplier for BN curves, and discusses the results obtained by our hardware implementation. Finally, Section VI concludes the paper.

2 Background

2.1 Bilinear Pairings

Let $\mathbb{F}_p$ be a finite field and let E be an elliptic curve defined over $\mathbb{F}_p$. Let r be a large prime dividing $\#E(\mathbb{F}_p)$ and k the embedding degree of $E(\mathbb{F}_p)$ with respect to r, namely, the smallest positive integer k such that $r|p^k - 1$. For any finite extension field $\mathbb{K}$ of $\mathbb{F}_p$, denote with $E(\mathbb{K})[r]$ the $\mathbb{K}$-rational r-torsion group of the curve. 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 a Miller function [27], [28].

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 [4] 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 [23] 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 p-th power Frobenius endomorphism, i.e. $\pi_p : E \to E : (x, y) \mapsto (x^p, y^p)$. Let

Algorithm 1 Computing the optimal ate pairing on BN curves [14]

Input: $P \in E(\mathbb{F}_p)[r]$, $Q \in E(\mathbb{F}_{p^k})[r] \bigcap \mathrm{Ker}(\pi_p - [p])$ and $a = 6\bar{z} + 2$.
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.

$P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and let $t := p + 1 - \#E(\mathbb{F}_p)$ be the trace of Frobenius, then the ate pairing is also a well-defined, non-degenerate bilinear pairing, and can be computed as

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

The R-ate pairing [26] is a generalization of the ate pairing and can be seen as an instantiation of optimal pairings [22], [33]. Since the definition of the optimal ate pairing really depends on the particular elliptic curve one is using, we only provide the definition in the case of BN curves: using the same $\mathbb{G}_1$ and $\mathbb{G}_2$ as for the ate pairing, the optimal 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 $a = 6\bar{z} + 2$, $f = f_{a,Q}(P)$ and $l_{A,B}$ denotes the line through points A and B.

Due to limited space, we only describe the algorithm to compute the optimal ate pairing for BN curves. The algorithms for Tate and ate pairings are similar, and can be found in [14].

2.2 Pairing-Friendly curves

An elliptic curve E over $\mathbb{F}_p$ is called pairing-friendly whenever there exists a large prime $r \mid \#E(\mathbb{F}_p)$ with $r \geq \sqrt{p}$ and the embedding degree k is small enough, e.g. $k \leq \log_2(r)/8$. Furthermore, if $\#E(\mathbb{F}_p) = p + 1 - t$ with t the trace of Frobenius, then $t^2 - 4p$ should have a very small squarefree part (e.g. less than 10^{10}) to be able to find the equation of the curve using the complex multiplication (CM) method [3]. These restrictions imply that pairing-friendly curves are hard to find and several specialized constructions have been proposed (see [18] for an excellent overview).

Many construction methods result in a parametrized family of elliptic curves, i.e. r and p are given by the evaluation of polynomials $r(z)$ and $f(z)$ at an integer value $\bar{z}$. To illustrate this type of curve, we provide several important examples of complete families of curves.

Example 1: 112**-bit security level** Consider the following family with $k = 8$ given in [18, Construction 6.10]: The polynomials

$$f(z) = \frac{1}{4}(81z^6 + 54z^5 + 45z^4 + 12z^3 + 13z^2 + 6z + 1)$$
$$r(z) = 9z^4 + 12z^3 + 8z^2 + 4z + 1$$

define a family of pairing-friendly curves with $k = 8$ and CM by -1, which admits quartic twists. Furthermore, finding the equation of the curve is straightforward due to CM by -1. Given a $\bar{z}$-value such that $r(\bar{z})$ is a 224-bit prime, this family provides 112-bit security.

Example 2: 128**-bit security level** The Barreto-Naehrig curves [32] are by far the most important family of pairing-friendly curves for the 128-bit security level. These curves have $k = 12$ and are defined by

$$f(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$$
$$r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1\,.$$

Furthermore, since these curves have CM by -3, the equation for such a curve is easy to find. Finally, they also admit sextic twists which speeds up all types of pairings. Given a $\bar{z}$-value such that $r(\bar{z})$ is a 256-bit prime, this family provides 128-bit security.

Example 3: 256**-bit security level** For very high security levels, the BN curves are somewhat ill-adapted and it is better to use the following cyclotomic family with $k = 24$ [9]. Recall that the 24-th cyclotomic polynomial is $\Phi_{24}(z) = z^8 - z^4 + 1$, then

$$f(z) = \frac{1}{3}(z - 1)^2\Phi_{24}(z) + z \qquad r(z) = \Phi_{24}(z)\,.$$

Again, this family has CM by -3 so sextic twists are possible. Given a $\bar{z}$-value such that $r(\bar{z})$ is a 512-bit prime, this family provides 256-bit security.

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 [5] uses a precomputed value $\mu = \lfloor \frac{2^{2n}}{p} \rfloor$ to help estimate $\frac{c}{p}$, thus integer division is avoided. Dhem [15] proposed an improved Barrett modular multiplication algorithm which has a simplified final correction.

Montgomery reduction The Montgomery reduction method [29] precomputes $p' = -p^{-1} \bmod R$, where R normally is a power of two. Given c and p, it generates q such that $c + qp$ is a multiple of R. As a result, the division of $(c + qp)$ by R is exact and can be performed by a shift operation.

Chung-Hasan reduction In [11], [12], Chung and Hasan proposed an efficient reduction method for low-weight polynomial form moduli $p = f(\bar{z}) = \bar{z}^n +$

Algorithm 2 Chung-Hasan multiplication algorithm [11].

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

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

Return $c(z)$.

$f_{n-1}\bar{z}^{n-1} + .. + f_1\bar{z} + f_0$, where $|f_i| \leq 1$. The resulting modular multiplication is given in Algorithm 2. The polynomial reduction phase is rather efficient since $f(z)$ is monic, making the polynomial long division (Steps 3-5) simple. Barrett reduction is used to perform divisions required in Phase III. According to the implementation results [11], the performance of the Chung-Hasan algorithm is more efficient than the traditional Barrett or Montgomery reduction algorithms when the moduli are large (See Fig. 5 in [11] for details). In [12], this algorithm is further extended to monic polynomials with $|f_i| \leq s$ where $s \ll \bar{z}$. Note that the polynomial reduction phase is efficient only when $f(z)$ is monic.

3 Hybrid Modular Multiplication

3.1 Polynomial Montgomery Reduction

In this section we introduce a new reduction algorithm for polynomial form moduli that can be seen as the dual algorithm to Chung-Hasan. Whereas Chung and Hasan require the polynomial defining the modulus to be monic, our algorithm requires the constant coefficient to be ± 1. As will be shown below this requirement is a very natural one.

Let $f(z) = f_{n-1}z^{n-1} + \cdots + f_1 z + f_0 \in \mathbb{Z}[z]$ with $f_0 \neq 0$ and assume the modulus is given by $p = f(\bar{z})$ for some $\bar{z} \in \mathbb{Z}$. For an integer $c = \sum_{i=0}^{2n-2} c_i \bar{z}^i$, the first step in the algorithm applies Montgomery reduction in the polynomial representation $c(z) = \sum_{i=0}^{2n-2} c_i z^i$. Montgomery reduction avoids expensive divisions by computing $c(z)z^{-n} \bmod f(z)$, instead of $c(z) \bmod f(z)$ itself. The main idea is to solve for the polynomial $q(z)$ such that

$$c(z) - q(z)f(z) \equiv 0 \bmod z^n ,$$

which shows that $q(z) = c(z)g(z) \bmod z^n$ where $g(z) \equiv f(z)^{-1} \bmod z^n$. Since $\gcd(f(z), z) = 1$, we thus obtain that

$$(c(z) - q(z)f(z))/z^n \equiv c(z)z^{-n} \bmod f(z) ,$$

where the division by z^n is exact and can be computed by a simple right shift of the coefficients. Note that $f_0 \neq 0$ implies that $g(z) \in \mathbb{Q}[z]$ exists, but $g(z)$ does not necessarily have integer coefficients. The following lemma shows that the condition $f_0 = \pm 1$ is a natural one.

Lemma 1: Let $f(z) = f_{n-1}z^{n-1} + \cdots + f_1 z + f_0 \in \mathbb{Z}[z]$ with $f_0 \neq 0$, and define $g_k(z) = f(z)^{-1} \bmod z^k$, then a necessary and sufficient condition for the g_k to have integer coefficients is $f_0 = \pm 1$.

PROOF: The argument is classical and is a special case of Newton lifting over the rational function field $\mathbb{Q}(z)$. Indeed, $g_k(z) \equiv f(z)^{-1} \bmod z^k$ is the solution to the linear equation (in the indeterminate X)

$$f(z)X - 1 \equiv 0 \bmod z^k$$

which can be solved using the Newton iteration: $g_1(z) \equiv 1/f_0 \bmod z$ and then

$$\begin{aligned} g_k(z) &\equiv g_{k-1}(z) - (f(z)g_{k-1}(z) - 1)/f(z) \\ &\equiv 2g_{k-1}(z) - g_{k-1}^2(z)f(z) \bmod z^k . \end{aligned}$$

As such $f_0 = \pm 1$ is clearly sufficient for $g_k(z)$ to be defined over $\mathbb{Z}$. However, it is also necessary, since $g_k(z) \equiv f_0^{-1} \bmod z$. □

3.2 Parallel version

Algorithm 3 describes our modular multiplication algorithm for polynomial form moduli. The algorithm is composed of four phases, i.e. polynomial multiplication, a partial coefficient reduction, polynomial reduction and coefficient reduction. The polynomial reduction uses the Montgomery reduction, while the coefficient reduction uses division. We call this algorithm Hybrid Modular Multiplication (HMM).

For Algorithm 3 to be useful in practice, we need to show that given a bounded input, the algorithm returns an output which is similarly bounded. To this end we require two short lemmata. The first lemma analyzes the coefficient reduction in Phase IV, while the second lemma gives bounds on the input, such that the output satisfies the same bounds.

Lemma 2: Let $c(z) = \sum_{i=0}^{m} c_i z^i \in \mathbb{Z}[z]$ be a polynomial of degree m. Assume that $|c_i| < B$ for $i = 0, \ldots, m-1$ and $|c_m| < D$ for bounds B and D, then the coefficient reduction

for $i = 0$ to m **do**
$\quad c_{i+1} \leftarrow c_{i+1} + (c_i \text{ div } \bar{z})$, $c_i \leftarrow c_i \bmod \bar{z}$
end for

results in $0 \leq c_i < \bar{z}$ for $i = 0, \ldots, m$ and

$$|c_{m+1}| < \frac{D}{\bar{z}} + \frac{B}{\bar{z}(\bar{z}-1)} .$$

Algorithm 3 Parallel hybrid modular multiplication algorithm.

Input: positive integers $a = \sum_{i=0}^{n-1} a_i \bar{z}^i$, $b = \sum_{i=0}^{n-1} b_i \bar{z}^i$, modulus $p = f(\bar{z}) = f_{n-1}\bar{z}^{n-1} + .. + f_1\bar{z} + f_0$ with $f_0 = \pm 1$ and polynomial inverse $g(z) \equiv f(z)^{-1} \bmod z^n$.
Output: polynomial representation of $r(\bar{z}) \equiv a(\bar{z})b(\bar{z})\bar{z}^{-n} \bmod p$

1: **Phase I: Polynomial Multiplication**
2: $c(z) = \sum_{i=0}^{2n-2} c_i z^i \leftarrow a(z)b(z)$.
3: **Phase II: Coefficient Reduction**
4: **for** $i = 0$ to $n-1$ **do**
5: $\quad c_{i+1} \leftarrow c_{i+1} + (c_i \text{ div } \bar{z})$, $c_i \leftarrow c_i \bmod \bar{z}$.
6: **end for**
7: **Phase III: Polynomial Reduction**
8: $q(z) \leftarrow (c(z) \bmod z^n)g(z) \bmod z^n$.
9: $c(z) \leftarrow (c(z) - q(z)f(z))/z^n$.
10: **Phase IV: Coefficient Reduction**
11: **for** $i = 0$ to $n-2$ **do**
12: $\quad c_{i+1} \leftarrow c_{i+1} + (c_i \text{ div } \bar{z})$, $c_i \leftarrow c_i \bmod \bar{z}$.
13: **end for**
Return $r(z) \leftarrow c(z)$.

PROOF: Using induction it is easy to see that in step i, the size of $c_i \text{ div } \bar{z}$ is bounded by

$$|c_i \text{ div } \bar{z}| < \sum_{k=1}^{i+1} |c_{i+1-k}|/\bar{z}^k < \sum_{k=1}^{i+1} B/\bar{z}^k < \frac{B}{\bar{z}-1}\,.$$

The size of c_m in step $m-1$ therefore becomes

$$|c_m| < D + \frac{B}{\bar{z}-1}\,,$$

which shows that in step m, we obtain the bound

$$|c_{m+1}| < \frac{D}{\bar{z}} + \frac{B}{\bar{z}(\bar{z}-1)}\,.$$

□

Using Lemma 2 we are now ready to show that given a bounded input, the result of Algorithm 3 is again bounded. Recall that for a polynomial $h = \sum_i h_i z^i \in \mathbb{Z}[z]$, the infinity norm is defined as $||h||_\infty = \max_i |h_i|$.

Lemma 3: Let $\tau = \lfloor \log_2(\bar{z}-1) \rfloor$, $\phi = \lceil \log_2 ||f||_\infty \rceil$ and $\zeta = \lceil \log_2 ||g||_\infty \rceil$, then if the input $a(z), b(z)$ satisfies

$$|a_i|, |b_i| \le 2^{\tau+\kappa} \quad \text{for } i = 0, \ldots, n-2\,,$$
$$|a_{n-1}|, |b_{n-1}| \le 2^{\tau/2+\kappa}\,,$$

for some $\kappa \ge 1$, then $r(z)$ computed by Algorithm 3 satisfies

$$|r_i| \le 2^{\tau+1} \qquad \text{for } i = 0, \ldots, n-2$$
$$|r_{n-1}| \le 4n^2(2^{\zeta+\phi} + 2^{2\kappa})\,.$$

This shows that if $\tau \ge 2(\log_2(4n^2(2^{\zeta+\phi} + 2^{2\kappa})) - \kappa)$, then $r(z)$ satisfies the same bounds as the input.
PROOF: After step 2, the coefficients of c_i are clearly bounded by $n2^{2\tau+2\kappa}$ for $i = 0, \ldots, 2n-3$ and $|c_{2n-2}| \le 2^{\tau+2\kappa}$. Lemma 2 shows that after the first coefficient reduction we have $|c_i| \le \bar{z}$ for $i = 0, \ldots, n-1$ and $|c_n| \le n2^{2\tau+2\kappa+1}$.

The coefficients of $q(z)$ are easily seen to be bounded by $|q_i| \le n2^{\tau+\zeta+1}$ so after step 9, we obtain

$$|c_i| \le n2^{2\tau+2\kappa+1} + n^2 2^{\tau+\zeta+\phi+1} =: B$$
$$|c_{n-2}| \le 2^{\tau+2\kappa} + n^2 2^{\tau+\zeta+\phi+1} =: D\,.$$

Applying Lemma 2 again with the above B and D, shows that

$$\begin{aligned} |c_{n-1}| &\le \tfrac{D}{\bar{z}} + \tfrac{B}{\bar{z}(\bar{z}-1)} \\ &\le n^2 2^{\zeta+\phi+2} + n2^{2(\kappa+1)} \\ &\le 4n^2(2^{\zeta+\phi} + 2^{2\kappa})\,. \end{aligned}$$

□

Given a polynomial $f(z)$ it is easy to compute the inverse $g(z) \equiv f(z)^{-1} \bmod z^n$ and thus to determine the exact values for ϕ and ζ in Lemma 3. For each polynomial, we therefore obtain a very explicit lower bound on $\bar{z}$ such that Algorithm 3 can be applied repeatedly.

3.3 Digit-serial version

Algorithm 4 presents the digit-serial version of Algorithm 3 by interleaving polynomial multiplication and reduction. While the parallel version uses the complete polynomial $g(z) \equiv f^{-1}(z) \bmod z^n$, the digit-serial reduction involves only $g_0 = \pm 1$. On the other hand, the parallel version could possibly use Karatsuba's algorithm [25] in steps 2, 8 and 9 to reduce the number of sub-word multiplications. The coefficient reduction phase is the same as in Algorithm 3.

Algorithm 4 Digit-serial hybrid modular multiplication algorithm

Input: $a(\bar{z}) = \sum_{i=0}^{n-1} a_i \bar{z}^i$, $b(\bar{z}) = \sum_{i=0}^{n-1} b_i \bar{z}^i$, and modulus $p = f(\bar{z}) = \sum_{i=0}^{n-1} f_i \bar{z}^i$ with $f_0 = \pm 1$.
Output: polynomial representation of $r(\bar{z}) \equiv a(\bar{z})b(\bar{z})\bar{z}^{-n} \bmod f(\bar{z})$.

1: $c(z) = \sum_{i=0}^{n-1} c_i z^i \leftarrow 0$.
2: **for** $i = 0$ to $n-1$ **do**
3: $\quad c(z) \leftarrow c(z) + a(z)b_i$.
4: $\quad \mu \leftarrow c_0 \text{ div } \bar{z}$, $\gamma \leftarrow c_0 \bmod \bar{z}$.
5: $\quad h(z) \leftarrow (f_{n-1}z^{n-1} + \cdots + f_1 z + f_0)(-f_0\gamma)$.
6: $\quad c(z) \leftarrow (c(z) + h(z))/z + \mu$.
7: **end for**
8: **for** $i = 0$ to $n-2$ **do**
9: $\quad c_{i+1} \leftarrow c_{i+1} + (c_i \text{ div } \bar{z})$, $c_i \leftarrow c_i \bmod \bar{z}$.
10: **end for**
Return $r(z) \leftarrow c(z)$.

The following lemma is the analogue of Lemma 3, but for the digit-serial version.

Lemma 4: Let $\tau = \lfloor \log_2(\bar{z}-1) \rfloor$ and $\phi = \lceil \log_2 ||f||_\infty \rceil$, then if $n^2 < \bar{z}$ and the input $a(\bar{z}), b(\bar{z})$ satisfies

$$|a_i|, |b_i| \le 2^{\tau+\kappa} \quad \text{for } i = 0, \ldots, n-2\,,$$
$$|a_{n-1}|, |b_{n-1}| \le 2^{\tau/2+\kappa}\,.$$

for some $\kappa \geq 1$, then $r(z)$ computed by Algorithm 4 satisfies

$$|r_i| \leq 2^{\tau+1} \qquad \text{for } i = 0, \ldots, n-2$$
$$|r_{n-1}| \leq 2(n+2)(2^{2\kappa} + 2^{\phi}) .$$

This shows that if $\tau \geq 2(\log_2(2(n+2)(2^{2\kappa} + 2^{\phi})) - \kappa)$, then $r(z)$ satisfies the same bounds as the input.

PROOF: The idea of the proof is similar to the proof of Lemma 3, i.e. before the final coefficient reduction we want to show that the coefficient c_{n-2} is small enough to ensure that after reduction, c_{n-1} is small. Note that at the beginning of each iteration in step 2, the degree of $c(z)$ is at most $n-2$, which shows that after step 7 we have $c_{n-2} = a_{n-1}b_{n-1} - \gamma_{n-1}f_{n-1}f_0$. Here γ_i denotes the γ computed in step 4 in the i-th iteration. Since $|\gamma_i| \leq |\bar{z}|$, this shows that $|c_{n-2}| \leq 2^{\tau+2\kappa} + 2^{\tau+\phi+1}$.

Assume that the coefficients c_i for $i = 0, \ldots, n-3$ are bounded by B (to be determined below) and let $D = 2^{\tau+2\kappa} + 2^{\tau+\phi+1}$, then again we can apply Lemma 2 resulting in

$$|c_{n-1}| \leq \frac{D}{\bar{z}} + \frac{B}{\bar{z}(\bar{z}-1)} \leq 2^{2\kappa} + 2^{\phi+1} + \frac{B}{\bar{z}(\bar{z}-1)} .$$

Obtaining the bound B is easy too: denote with B_i the bound on $||c(z)||_\infty$ at the start of the i-th iteration, then we have $B_0 = 0$ and for $i \geq 1$:

$$B_i \leq B_{i-1}(1 + \frac{1}{\bar{z}}) + 2^{2\tau+2\kappa+1} + 2^{\tau+\phi+1} .$$

Let $\alpha = (1 + 1/\bar{z})$ and $\beta = 2^{2\tau+2\kappa+1} + 2^{\tau+\phi+1}$, then we need to solve the recursion $B_i \leq \alpha B_{i-1} + \beta$ with $B_0 = 0$. This results in the bound $B_n \leq \beta(\alpha^n - 1)/(\alpha - 1)$ and since $\alpha = 1 + 1/\bar{z}$ this becomes $B_n \leq \beta\bar{z}(\alpha^n - 1)$. Writing out $\alpha^n - 1$ explicitly gives

$$\alpha^n - 1 = \sum_{k=1}^{n} \binom{n}{k} \frac{1}{\bar{z}^k} < \frac{n}{\bar{z}} + \sum_{j=2}^{\infty} \frac{n^j}{j!\bar{z}^j} .$$

Assuming that $n^2 \leq \bar{z}$, we can easily bound the sum above by $(e-1)/\bar{z}$, so $\alpha^n - 1 \leq (n+e-1)/\bar{z}$. So we finally obtain

$$B = B_n \leq (n+e-1)(2^{2\tau+2\kappa+1} + 2^{\tau+\phi+1}) .$$

As a result, the bound on c_{n-1} becomes

$$\begin{aligned} |c_{n-1}| &\leq 2^{2\kappa} + 2^{\phi+1} + (n+e-1)(2^{2\kappa+1} + 2^{\phi-\tau+1}) \\ &\leq (n+2)(2^{2\kappa+1} + 2^{\phi+1}) . \end{aligned}$$

□

It is important to point out that both Algorithm 3 and Algorithm 4 may involve negative numbers as intermediate or final results. As a result, a direct use of these algorithms requires the handling of negative numbers. In hardware implementations, one can avoid signed multiplication by adding a multiple of $f(\bar{z})$ to the $r(\bar{z})$. We describe this method in an example in Section 5.

3.4 Faster coefficient reduction

Algorithms 3 and 4 require division by $\bar{z}$. As in Chung-Hasan's algorithm, division can be performed with Barrett's reduction algorithm [11]. However, the complexity of the division step can be reduced if $\bar{z}$ is a pseudo-Mersenne number. Algorithm 5 transfers division by $\bar{z}$ to multiplication by s for $\bar{z} = 2^\tau + s$ where s is small.

Algorithm 5 Division by $\bar{z} = 2^\tau + s$

Input: a, $\bar{z} = 2^\tau + s$ with $0 < s < 2^{\lfloor \tau/2 \rfloor}$.
Output: (μ, ϵ) with $a = \mu\bar{z} + \epsilon$, $|\epsilon| < \bar{z}$.

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

Return μ, ϵ.

The following lemma gives the maximum number of iterations for Algorithm 5 to finish.

Lemma 5: Define π such that $|a| \leq 2^\pi$ and ν such that $0 < s \leq 2^{\nu-1}$, then Algorithm 5 requires at most $\lceil \frac{\pi-\tau-1}{\tau-\nu} \rceil + 1$ iterations.

PROOF: After one iteration we have

$$|\epsilon| = |\epsilon - s\rho| \leq |\epsilon| + s|\rho| \leq (2^\tau - 1) + 2^{\pi+\nu-1-\tau} .$$

To bound the right hand side, we need to consider the two cases:

- $(2\tau - \nu + 1) \leq \pi$, then $|\epsilon| \leq 2^{\pi-\tau+\nu}$, so in each iteration the bit length is decreased by $\tau - \nu$.
- $(2\tau - \nu + 1) > \pi$, then $|\epsilon| < 2^{\tau+1}$, so either ϵ is reduced or one last step is needed

$$|\epsilon| = |\epsilon - s\rho| \leq (2^\tau - 1) + s < \bar{z} .$$

In conclusion: $\lceil \frac{\pi-(2\tau-\nu+1)}{\tau-\nu} \rceil$ iterations are needed to reduce to the second case and then a maximum of two iterations is needed to finish reduction in the second case. In total, the algorithm thus requires $\lceil \frac{\pi-\tau-1}{\tau-\nu} \rceil + 1$ iterations. □

When using Algorithm 5 to perform step 4 and 9 in Algorithm 4, at most three iterations are required. This is guaranteed if $|c_i| \leq 2^{3\tau-2\nu+1}$ for $0 \leq i \leq n-1$. Recall that c_i satisfies the following bounds:

$$|c_i| \leq B = (n+e-1)(2^{2\tau+2\kappa+1} + 2^{\tau+\phi+1}),$$

for $0 \leq i < n-2$, and

$$|c_{n-2}| \leq D = 2^{\tau+2\kappa} + 2^{\tau+\phi+1}.$$

Since $B > D$, it is sufficient to ensure $B \leq 2^{3\tau-2\nu+1}$. Since we typically have $\phi \leq \tau$,

$$\log_2 B \leq \log_2(n+e-1) + 2\tau + 2\kappa + 2 .$$

By choosing $\tau \geq 2(\kappa+\nu) + \log_2(n+e-1) + 1$, we ensure

$$\log_2 B \leq 3\tau - 2\nu + 1 .$$

3.5 Complexity

We compare the complexity of the proposed algorithm with the algorithms by Barrett and Montgomery. Here τ, ϕ are defined as in Lemma 3, namely, $\tau = \lfloor \log_2(\bar{z}-1) \rfloor$, $\phi = \lceil \log_2 ||f||_\infty \rceil$, $\bar{z} = 2^\tau + s$ where $0 < s \leq 2^{\nu-1}$. The complexity of each step in Algorithm 4 is as follows.

- Step 3: for $i \leq n-2$, computing $a(z)b_i$ involves one $(\lceil \frac{\tau}{2} \rceil + \kappa) \times (\tau + \kappa)$ and $(n-1)$ times $(\tau+\kappa) \times (\tau + \kappa)$ multiplications. For $i = n-1$, computing $a(z)b_i$ involves one $(\lceil \frac{\tau}{2} \rceil + \kappa) \times (\lceil \frac{\tau}{2} \rceil + \kappa)$ and $(n-1)$ times $(\lceil \frac{\tau}{2} \rceil + \kappa) \times (\tau + \kappa)$ multiplications.
- Step 5: computing $f(z)(-f_0\gamma)$ needs $(n-1)$ times $\tau \times \phi$ multiplications.
- Step 4 and 9: as discussed above, three iterations are required at most. We have $|\rho| \leq \log_2(n+e-1)+\tau+2\kappa+2$ and $|\rho| \leq \log_2(n+e-1)+\nu+2\kappa+3$ in the first iteration and second iteration, respectively. For the third iteration, $|\rho| \leq 1$ is guaranteed. The cost of step 4 or 9 is one $\nu \times (\tau + \lceil \log_2(n+e-1) \rceil + 2\kappa + 2)$ multiplication and one $\nu \times (\lceil \log_2(n+e-1) \rceil + \nu + 2\kappa + 3)$ multiplication.

Table 1 compares the complexity of the different multiplication algorithms. Note that ϕ and ν are chosen to be small, so multiplication with $f(z)$ and s can be efficiently performed.

The complexity of the HMM algorithm highly depends on the chosen curve parameters since they determine $\{\tau, \nu, n, \phi, \kappa\}$. Each family of pairing-friendly curves is defined by a fixed polynomial $f(z)$, so n and ϕ are constant for a given family. The size of τ is set by the desired security level. To achieve lower complexity, it is thus desirable to choose small ν and κ. According to Lemma 4, κ can be as small as 1 and the size of ν is determined by s where $\bar{z} = 2^\tau + s$.

Computing $ABR^{-1} \bmod M$ using conventional Montgomery multiplication requires $2w^2 + w$ sub-word multiplications (see [10] for details), where w is the number of digits in A and B. Note that $w = n-1$ here since we need one more digit to represent $A(z)$ or $B(z)$. Barrett multiplication has approximately the same complexity as Montgomery's algorithm [11].

Compared to τ, the parameters ϕ and ν are much smaller. This makes the polynomial reduction and coefficient reduction very efficient. We present some parameters for the examples in Section 2.2 and compare the complexities of different multiplication algorithms.

4 Parameter Selection for Pairing-Friendly Curves

In this section, we present values for $\bar{z}$ for each of the three examples given in Section 2.2. Note that for the first and third family, $f(z)$ does not have integral coefficients, but $3f(z)$ or $4f(z)$ does, so we simply work modulo these polynomials.

Table 2 contains values $\bar{z}$ that lead to efficient instantiations of the HMM algorithm. Furthermore, the bit-length of $\bar{z}$ is chosen to reflect the ideal security level at which the respective families of curves should be used.

Table 3 compares the complexity of HMM and Montgomery's algorithm using parameter sets of Table 2. In hardware designs, one can customize the multipliers for specific operand sizes. HMM takes advantage of this freedom to reduce the complexity of the multiplier. For example, a 6×63-bit multiplier is much smaller and faster than a 64×64-bit multiplier. Compared with Montgomery multiplication, the HMM has a significantly lower complexity in the context of hardware implementation.

Note that the complexity of HMM can be reduced further by exploring the characteristics of the coefficients of $f(z)$. For example, BN-curves have $f(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$, and $f(z)(-f_0\gamma)$ can be computed as follows:

- Step 1: $6\gamma = 2^2\gamma + 2\gamma$;
- Step 2: $24\gamma = 2^2(6\gamma)$;
- Step 3: $36\gamma = 24\gamma + 2(6\gamma)$;

Instead of performing four 63×6 multiplications in each iteration, four shift and two addition operations are performed. Example 3 has $\phi = 1$, resulting in even larger savings in the polynomial reduction step, since no multiplications are required as reflected in Table 3 by the 1×64 cell.

5 Application to BN Curves

In this section, we propose an architecture for HMM in hardware. Note that Fan *et al.* described the first digit-serial architecture for the HMM algorithm [17] for BN curves, where polynomial multiplication and reduction are interleaved. In the architecture proposed in this paper, polynomial multiplication and reduction are separated as in Algorithm 3. We show that this architecture is flexible and can achieve higher throughput than the one from [17]. We choose the same parameters as [17], namely, $\bar{z} = 2^{63} + s$ and s=$2^9 + 2^8 + 2^6 + 2^4 + 2^3 + 1$. With these parameters, the implementation achieves 128-bit security.

Algorithm 6 shows the HMM algorithm for BN curves. Since $f(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$, we have $g(z) \equiv -f^{-1}(z) \equiv 324z^4 - 36z^3 - 12z^2 + 6z - 1 \bmod z^5$. Note that both $f(z)$ and $g(z)$ have relatively small coefficients. As a result, the polynomial reduction phase can be efficiently implemented.

5.1 HMM Multiplier

Figure 1 shows the architecture of the multiplier. It consists of a row of multipliers to carry out polynomial multiplication, five "Mod-1" blocks to perform the first coefficient reduction, a module to accumulate the partial products, a module to perform polynomial reduction and four "Mod-2" blocks to perform the second coefficient reduction. Figure 1 also gives the bit-length of input/output of each block.

TABLE 1
Complexity comparison of different modular multiplication algorithms
($\bar{z} = 2^\tau + s$, $0 < s \leq 2^{\nu-1}$, $\phi = \lceil \log_2 ||f||_\infty \rceil$, $\delta_1 = \tau + \lceil \log_2(n+e-1) \rceil + 2\kappa + 2$, $\delta_2 = \lceil \log_2(n+e-1) \rceil + \nu + 2\kappa + 3$.)

Algorithm	$(\lceil \frac{\tau}{2} \rceil + \kappa) \times (\lceil \frac{\tau}{2} \rceil + \kappa)$	$(\lceil \frac{\tau}{2} \rceil + \kappa) \times (\tau + \kappa)$	$(\tau + \kappa) \times (\tau + \kappa)$	$\phi \times \tau$	$\delta_1 \times \nu$	$\delta_2 \times \nu$
Barrett			$2(n-1)^2 + n - 1$			
Montgomery			$2(n-1)^2 + n - 1$			
HMM (Alg. 4)	1	$2(n-1)$	$(n-1)^2$	$n(n-1)$	$2n-1$	$2n-1$

TABLE 2
Selection of $\bar{z}$ for curves ($\bar{z} = 2^\tau + s$, $0 < s \leq 2^{\nu-1}$, $\phi = \lceil \log_2 ||f||_\infty \rceil$)

Family	k	$\bar{z}$	ϕ	ν	$\lceil \log(2,p) \rceil$	$\lceil \log(2,r) \rceil$
Example 1	8	$2^{39} + 1175$	7	12	239	159
Example 2	12	$2^{63} + 857$	6	11	258	258
Example 3	24	$2^{64} + 11757$	1	15	639	513

TABLE 3
Multiplication complexity for each set of parameters

Example 1: 80-bit security ($n = 7$, $\tau = 39, \nu = 12, \phi = 7, \kappa = 1, \delta_1 = 47, \delta_2 = 21$)						
	21×21	21×40	40×40	7×39	47×12	21×12
Montgomery			78			
HMM (Alg. 4)	1	12	36	42	13	13
Example 2: 128-bit security ($n = 5$, $\tau = 63, \nu = 11, \phi = 6, \kappa = 1, \delta_1 = 70, \delta_2 = 19$)						
	33×33	33×64	64×64	6×63	70×11	19×11
Montgomery			36			
HMM (Alg. 4)	1	8	16	20	9	9
Example 3: 256-bit security ($n = 11$, $\tau = 64, \nu = 15, \phi = 1, \kappa = 1, \delta_1 = 72, \delta_2 = 24$)						
	33×33	33×65	65×65	1×64	72×15	24×15
Montgomery*			210			
HMM (Alg. 4)	1	20	100	110	21	21

* 64 bits used for each digit (10x64=640), thus 64×64 bit multiplier is used.

Phase I. Five integer multipliers, including four 65×65 and one 65×32 multipliers, are used to carry out polynomial multiplication. Clearly, given enough area, the polynomial multiplication stage can be fully parallelized (using 13 multipliers [30]). The architecture used here is a tradeoff between area and throughput. Using the schoolbook method, it requires 5 cycles to finish Phase I. Each 65×65 multiplier is implemented using a two-level Karatsuba's method, and each multiplication has a delay of 7 cycles.

Phase II. The partial products ($a_i b_j$, $0 \leq i, j < n$) are reduced immediately after they are generated. Figure 1(d) shows the structure of the "Mod-1" block. Since $s = 2^5 \cdot (2^4 + 2^3) + 2^6 + (2^4 + 2^3) + 1$, multiplication by s is implemented with four additions. The output of Phase II is then accumulated and shifted in the accumulator, shown in Figure 1(f). Note that the output buffers of the adders, except the one on the most right side, should be set to zero for each $\mathbb{F}_p$ multiplication.

The "Mod-1" block only performs one round of reduction, thus it does not give fully reduced results. It is easy to see that the output of "Mod-1" is always less than 2^{77}. We shall see this is good enough for this architecture.

Phase III. Once the partially reduced results ($c_8, .., c_0$) are ready, Phase III is applied. The polynomial reduction is performed with only addition and shift operations, e.g. $6\alpha = 2^2\alpha + 2\alpha$, $9\alpha = 2^3\alpha + \alpha$ and $36\alpha = 2^5\alpha + 2^2\alpha$.

Since the output of "Mod-1" is less than 2^{77}, we have $|c_i| < (i+1) \cdot 2^{77}$ for $0 \leq i \leq 4$. In the computation of $q(z)$ in Algorithm 6, $q_4 = -c_4 + 6(c_3 - 2c_2 - 6(c_1 - 9c_0))$, resulting in $|q_4| < (5 + 6 \cdot (4 + 2 \cdot 3 + 6 \cdot (2 + 9)))2^{77} = 16596 \cdot 2^{77}$. Likewise, we can compute the size of h_i and v_i for $0 \leq i \leq 3$. One can verify that $v(z)$ has coefficients $|v_i| < 2^{92}$ for $0 \leq i \leq 3$.

Phase IV. The "Mod-2" block is implemented in the same way as "Mod-1". However, the input of "Mod-2" is only 93-bit. Thus, the output of the proposed multiplier has the following bounds: $|r_i| < 2^{63} + 2^{41}$, $0 \leq i \leq 3$, and $|r_4| < 2^{30}$.

Note that the resulting polynomial, $r(z)$ may have negative coefficients. For future multiplications, negative coefficients as input are not desirable. We can ensure positive coefficients by adding to $r(z)$ the following polynomial:

$$\begin{aligned} l(z) = \quad & (36\vartheta - 2)z^4 + (36\vartheta + 2\bar{z} - 2)z^3 + \\ & (24\vartheta + 2\bar{z} - 2)z^2 + (6\vartheta + 2\bar{z} - 2)z + (\vartheta + 2\bar{z}), \end{aligned}$$

where $\vartheta = 2^{25}$. One can verify that $l(\bar{z}) = 2^{25} f(\bar{z})$. Let $r'(z) = \Sigma_{i=0}^{4} r'_i z^i = r(z) + l(z)$, then we have $0 \leq r'_4 < 2^{32}$. For $0 \leq i \leq 3$, $2\bar{z} < l_i$ and $|r_i| < 2\bar{z}$, thus we have $r'_i > 0$. On the other hand, $r_i + l_i < 2(2^{63} + s) + 36\vartheta - 2 + 2^{63} + 2^{41} < 2^{65}$. Thus, $r'(z)$ has only positive coefficients and satisfies the input bounds.

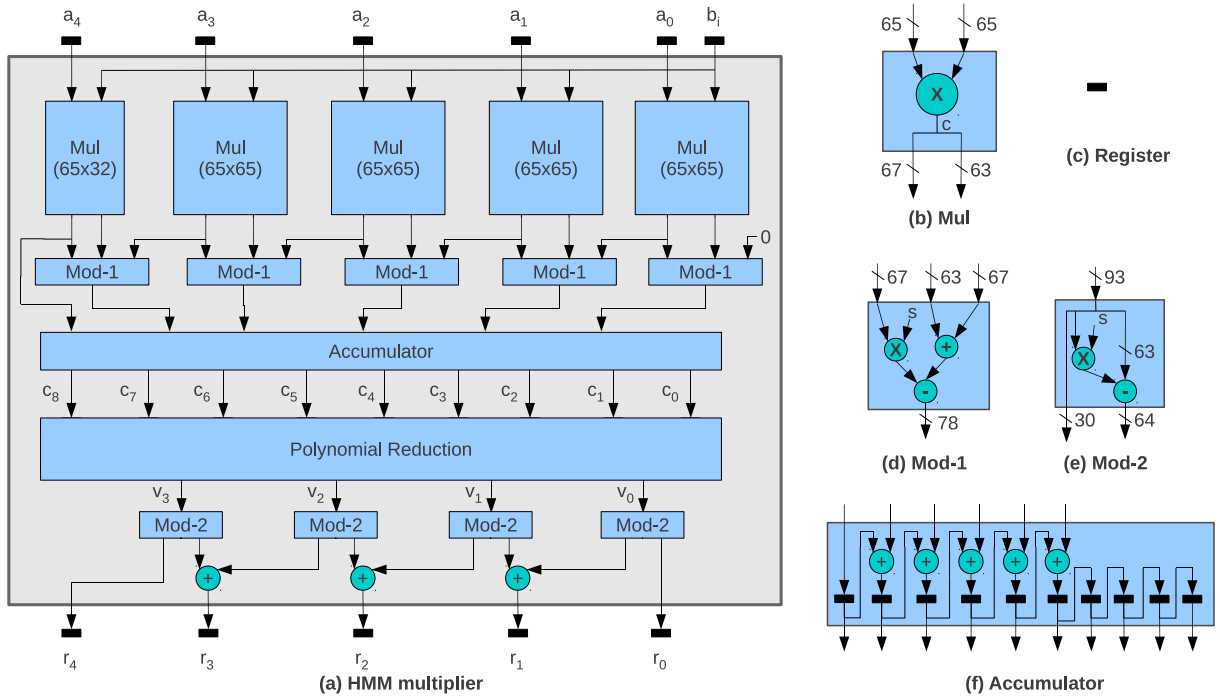


Fig. 1. $\mathbb{F}_p$ multiplier using algorithm HMMB.

Algorithm 6 Parallel hybrid modular multiplication algorithm for BN curves.

Input: positive integers $a = \sum_{i=0}^{4} a_i \bar{z}^i$, $b = \sum_{i=0}^{4} b_i \bar{z}^i$, modulus $p = f(\bar{z}) = 36\bar{z}^4 + 36\bar{z}^3 + 24\bar{z}^2 + 6\bar{z} + 1$.
Output: polynomial representation of $r(\bar{z}) \equiv a(\bar{z})b(\bar{z})\bar{z}^{-5} \bmod p$

1: **Phase I: Polynomial Multiplication**
2: $c(z) = \sum_{i=0}^{8} c_i z^i \leftarrow a(z)b(z)$.
3: **Phase II: Coefficient Reduction**
4: **for** $i = 0$ to 4 **do**
5: $c_{i+1} \leftarrow c_{i+1} + (c_i \text{ div } \bar{z})$, $c_i \leftarrow c_i \bmod \bar{z}$.
6: **end for**
7: **Phase III: Polynomial Reduction**
8: $q(z) = \sum_{i=1}^{4} q_i t^i \leftarrow (-c_4 + 6(c_3 - 2c_2 - 6(c_1 - 9c_0)))z^4 + (-c_3 + 6(c_2 - 2c_1 - 6c_0))z^3 + (-c_2 + 6(c_1 - 2c_0))z^2 + (-c_1 + 6c_0)z$.
9: $h(z) = \sum_{i=0}^{3} g_i t^i \leftarrow (36q_4)z^3 + 36(q_4 + q_3)z^2 + 12(2q_4 + 3(q_3 + q_2))z + 6(q_4 + 4q_3 + 6(q_2 + q_1))$.
10: $v(z) \leftarrow c(z)/z^5 + h(z)$;
11: **Phase IV: Coefficient Reduction**
12: **for** $i = 0$ to 3 **do**
13: $v_{i+1} \leftarrow v_{i+1} + (v_i \text{ div } \bar{z})$, $v_i \leftarrow v_i \bmod \bar{z}$.
14: **end for**
Return $r(z) \leftarrow v(z)$.

5.2 Implementation Results for Pairing Co-processor

We used Xilinx FPGA as the design platform. In order to achieve a high clock frequency, a 16-stage pipeline is used in the HMM multiplier. Note that the polynomial multiplication takes five iterations. One multiplication on the proposed multiplier has a delay of 20 cycles. On the other hand, the multiplier finishes one multiplication every 5 cycles, thus has a throughput of 1/5.

Using the multiplier described above, we built a pairing processor. It consists of a multiplier, an adder, a 74Kbit data memory and an instruction ROM. The data memory and instruction ROM are essentially implemented with Block RAMs on the FPGA. The data memory has one read port and one write port.

In order to keep the HMM multiplier busy, we explore the parallelism within the pairing computation. Consider $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 + 2)$ and two elements in $\mathbb{F}_{p^2}$: $(a + bu)$ and $(c + du)$, $a, b, c, d \in \mathbb{F}$, then $(a + bu)(c + du)$ can be computed as follows:

$$(a + bu)(c + du) = (ac - 2bd) + ((a + b)(c + d) - ac - bd)u.$$

The $\mathbb{F}_p$ multiplications can be performed one after another. We use a C++ program to schedule each high level function, e.g. sub-routines of the Miller loop and the final exponentiation. The scheduling is then transfered into micro-instructions stored in the instruction ROM. Table 4 gives the cycle counts of each function.

On a Xilinx Virtex-6 FPGA (XC6VLX240), the design uses 4,014 Slices, 42 DSP48E1s and 5 Block RAMs (RAMB36E1s). The design achieves a maximum frequency of 210 MHz. Table 5 compares the result with the state-of-the-art implementations.

Kammler *et al.* [24] reported a hardware implementation of cryptographic pairings using an application-specific instruction-set processor (ASIP). They chose $\bar{z}$=0x6000000000001F2D to generate a 256-bit BN curve.

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	optimal ate
#Cycles	220	342	196	120	540	432	138,302	336,366	245,430

TABLE 5
Performance comparison of software and hardware implementations of pairings

Design	Pairing	Security [bit]	Platform	Algorithm	Area	Freq. [MHz]	Cycle	Delay [ms]
This design	ate	128	Xilinx FPGA (Virtex-6)	HMM (Parallel)	4,014 Slices 42 DSP48E1s	210	336,366	1.60
	optimal ate						245,430	1.17
[17]	ate	128	ASIC (130 nm)	HMM (Digit-serial)	183 kGates	204	861,724	4.22
	optimal ate						592,976	2.91
[19]	Tate	128	Xilinx FPGA Virtex-4	Blakley [8]	52k Slices	50	1,730,000	34.6
	ate						1,207,000	24.2
	optimal ate						821,000	16.4
[24]	Tate	128	ASIC (130 nm)	Montgomery	97 kGates	338	11,627,200*	34.4
	ate						7,706,400*	22.8
	optimal ate						5,340,400*	15.8
[16]	Tate over $\mathbb{F}_{3^{5 \cdot 97}}$	128	Xilinx FPGA (Virtex-4)	-	4755 Slices 7 BRAMs	192	428,853	2.23
[21]	ate	128	64-bit Core2	Montgomery	-	2400	15,000,000	6.25
	optimal ate						10,000,000	4.17
[20]	ate	128	64-bit Core2	Montgomery		2400	14,429,439	6.01
[31]	optimal ate	128	Core2 Quad	Hybrid Mult.	-	2394	4,470,408	1.86
[6]	optimal ate	126	Core i7	Montgomery	-	2800	2,330,000	0.83
[1]	optimal ate	127	Phenom II	Montgomery	-	3000 †	1,562,000	0.52
[2]	η_T over $\mathbb{F}_{2^{1223}}$	128	Xeon (8 cores)	-	-	2000	3,020,000	1.51
[7]	η_T over $\mathbb{F}_{3^{509}}$	128	Core i7 (8 cores)	-	-	2900	5,423,000	1.87

* Estimated by the authors.
† Processor frequency is not mentioned in the original paper. We take 3.0 GHz (typical frequency) for delay estimation.

Montgomery's algorithm is used for $\mathbb{F}_p$ multiplication. Fan *et al.* [17] reported an ASIC implementation using HMM algorithm. They choose $\bar{z}$=$2^{63} + 857$. It achieves a factor 5.4 speed-up compared with the one of [24]. On the other hand, the ASIP design [24] offers higher flexibility than the ASIC implementation [17] that is optimized for a dedicated parameter set. Ghosh *et al.* [19] reported the first FPGA implementation of pairings based on BN curves. They also chose $\bar{z}$=0x6000000000001F2D to achieve 128-bit security. The $\mathbb{F}_p$ multiplier used in [19] is based on Blakley's algorithm [8], and it does not make use of the DSP slices on FPGA.

The design reported in this paper achieves a factor 2.5 speed-up compared with the one in [17]. The speed-up comes mainly from the high throughput multiplier. The multiplier used in [17] requires 23 cycles for each multiplication, while the multiplier described in this paper finishes one multiplication every 5 cycles. The Montgomery multiplier used in [24] requires 68 cycles for one multiplication. On the other hand, the speed-up for pairing computations is less than the speed-up of the multiplier. This is mainly due to the read-after-write (RAW) dependency that introduces pipeline bubbles.

Table 5 also includes the state-of-the-art software implementations [6], [20], [21], [31]. Software implementations try to make use of available features (fast multipliers, vector registers and so on) on the target processor. The speed records have been updated shortly after they are set, mainly due to new implementation techniques and the evolution of processors. The current speed record [1] of optimal ate pairing using BN curves is achieved by Aranha *et al.* on an AMD Phenom II processor, who use $\bar{z} = -(2^{62} + 2^{55} + 1)$. This choice not only reduces the complexity of both the Miller loop and the final exponentiation, but more importantly admits for extensive use of lazy reduction techniques. This implementation achieves so far the best performance.

An interesting attempt to use hybrid multiplication algorithm in software has been made [31]. Naehrig *et al.* adapted the hybrid multiplication algorithm and proposed a software-oriented algorithm for fast modular multiplication. An element in $\mathbb{F}_p$ is represented as a polynomial of degree 11. As such, the coefficients are short enough to fit in the vector registers and overflows in multiplication are avoided. Polynomial reduction can be efficiently performed since $p(x)$ is made monic. This implementation shows a significant speedup compared to previous software implementations [20], [21]. On the other hand, on CPUs where 64-bit multiplication is not much slower than addition, the traditional Montgomery algorithm seems to achieve a higher performance [1], [6].

Table 5 also includes the state-of-the-art implementations of pairings over binary or ternary fields achieving 128-bit security. The results of these implementations are comparable with pairings using BN curves in software. To the best of our knowledge, Estibals reported so far the only hardware implementation of 128-bit Tate pairing over super-singular curves in small characteristic. On a

Virtex-4 FPGA, it requires only 2.2 ms for the pairing computation. It is however difficult to give a fair comparison due to the use of different platforms.

6 Conclusions

In this paper we introduced a new modular multiplication algorithm for polynomial form moduli called Hybrid Modular Multiplication. The algorithm combines Montgomery's algorithm in the polynomial representation with a fast coefficient reduction based on pseudo-Mersenne numbers.

The algorithm results in a substantial speed-up for hardware implementation of finite field arithmetic appearing in pairing based cryptography. To illustrate this efficiency, we implemented pairings on Barreto-Naehrig curves at the 128-bit security level and obtained a factor 2.5 speed-up compared with the state-of-the-art hardware implementations.

Acknowledgments

This work was supported in part by K.U. Leuven-BOF (OT/06/40), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the Research Council K.U.Leuven: GOA TENSE, by FWO project G.0300.07 and by IBBT.

References

[1] D.F. Aranha, K. Karabina, P. Longa, C.H. Gebotys, and J. López. Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In *Eurocrypt 2011*. To appear.

[2] D.F. Aranha, J. López, and D. Hankerson. High-Speed Parallel Software Implementation of the η_T Pairing. In *CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2010.

[3] R.M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005.

[4] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.

[5] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1986.

[6] 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. In *Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.

[7] J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves. In *CANS 2009*, volume 5888 of *Lecture Notes in Computer Science*, pages 413–432. Springer, 2009.

[8] G.R. Blakley. A Computer Algorithm for Calculating the Product AB Modulo M. *IEEE Trans. Comput.*, 32(5):497–500, 1983.

[9] F. Brezing and A. Weng. Elliptic Curves Suitable for Pairing Based Cryptography. *Designs, Codes and Cryptography*, 37:133–141, 2003.

[10] Ç. K. Koç, T. Acar, and B. S. Kaliski. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16:26–33, 1996.

[11] J. Chung and M.A. Hasan. Low-Weight Polynomial Form Integers for Efficient Modular Multiplication. *IEEE Trans. Comput.*, 56(1):44–57, 2007.

[12] 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.

[13] A. Devegili, C. Ó' hÉigeartaigh, M. Scott, and R. Dahab. Multiplication and Squaring on Pairing-Friendly Fields. Cryptology ePrint Archive, Report 2006/471. Available from http://eprint.iacr.org.

[14] A. Devegili, M. Scott, and R. Dahab. Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In *Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 197–207. Springer, 2007.

[15] J.-F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards*. PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1998.

[16] N. Estibals. Compact Hardware for Computing the Tate Pairing over 128-Bit-Security Supersingular Curves. In *Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 397–416. Springer, 2010.

[17] J. Fan, F. Vercauteren, and I. Verbauwhede. Faster $\mathbb{F}_p$-Arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves. In *CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2009.

[18] D. Freeman, M. Scott, and E. Teske. A Taxonomy of Pairing-Friendly Elliptic Curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[19] S. Ghosh, D. Mukhopadhyay, and D.R. Chowdhury. High Speed Flexible Pairing Cryptoprocessor on FPGA Platform. In *Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 450–466, 2010.

[20] P. Grabher, J. Großschädl, and D. Page. On Software Parallel Implementation of Cryptographic Pairings. In *SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2008.

[21] D. Hankerson, A. Menezes, and M. Scott. Software Implementation of Pairings. In M. Joye and G. Neven, editors, *Identity-Based Cryptography*, 2008.

[22] F. Hess. Pairing Lattices. In *Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2008.

[23] F. Hess, N.P. Smart, and F. Vercauteren. The Eta Pairing Revisited. *Information Theory, IEEE Transactions on*, 52(10):4595–4602, Oct. 2006.

[24] 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. In *CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 254–271. Springer, 2009.

[25] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962.

[26] 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/.

[27] V.S. Miller. Short Programs for Functions on Curves, 1986. Unpublished manuscript, Available at http://crypto.stanford.edu/miller/miller.pdf.

[28] V.S. Miller. The Weil Pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

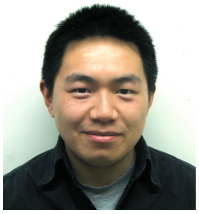
[29] P.L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[30] P.L. Montgomery. Five, Six, and Seven-Term Karatsuba-Like Formulae. *IEEE Trans. Comput.*, 54(3):362–369, 2005.

[31] M. Naehrig, R. Niederhagen, and P. Schwabe. New Software Speed Records for Cryptographic Pairings. In *LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2010.

[32] P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.

[33] F. Vercauteren. Optimal Pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.



Junfeng Fan received the BS and MS degrees in electrical engineering from Zhejiang University, China, in 2003 and 2006, respectively. Since 2006, he has been a PhD student in the Electrical Engineering Department (ESAT), Katholieke Universiteit Leuven (KU Leuven), Belgium. His research interests include computer arithmetics, with speical focus on efficient implementations for Public Key Cryptography (PKC). He is also interested in physical security of embedded systems and secure design methodologies.

Frederik Vercauteren received the M.Sc. degree in computer science, the M.Sc. degree in pure mathematics, and the Ph.D. degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium. He is currently a Postdoctoral Fellow of the Research Foundation-Flanders (FWO Vlaanderen) at the Department of Electrical Engineering, Katholieke Universiteit Leuven, Belgium. Previously, he held a lecturer position at the Department of Computer Science, University of Bristol, U.K. His current research interests include applications of computational number theory and arithmetic geometry in cryptography.

Ingrid Verbauwhede received the electrical engineering degree and PhD degree from the Katholieke Universiteit Leuven (KU Leuven), Belgium, in 1991. From 1992 to 1994, she was a postdoctoral researcher and visiting lecturer at the Electrical Engineering and Computer Sciences Department, University of California, Berkeley. From 1994 to 1998, she worked for TCSI and ATMEL in Berkeley, California. In 1998, she joined the faculty of University of California, Los Angeles (UCLA). She is currently a professor at the KU Leuven and an adjunct professor at UCLA. At KU Leuven, she is a codirector of the Computer Security and Industrial Cryptography (COSIC) Laboratory. Her research interests include circuits, processor architectures and design methodologies for real-time embedded systems for security, cryptography, digital signal processing, and wireless communications. This includes the influence of new technologies and new circuit solutions on the design of next-generation systems on chip. She was the program chair of the Ninth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 07), the 19th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 08), and the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED 02). She was also the general chair of ISLPED 2003. She was a member of the executive committee of the 42nd and 43rd Design Automation Conference (DAC) as the design community chair. She is a senior member of the IEEE.