Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves

Jean-Luc Beuchat[1], Emmanuel López-Trejo[2], Luis Martínez-Ramos[3], Shigeo Mitsunari[4], and Francisco Rodríguez-Henríquez[3]

[1] Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan
[2] Nehalem Platform Validation, Intel Guadalajara Design Center, Periférico Sur 7980 Edificio 4E, 45600 Tlaquepaque, Jalisco, México.
[3] Computer Science Department, Centro de Investigación y de Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, 07300 México City, México
[4] Akasaka Twin Tower East 15F, 2-17-22 Akasaka, Minato-ku, Tokyo 107-0052, Cybozu Labs, Inc.

Abstract. This paper describes the design of a fast multi-core library for the cryptographic Tate pairing over supersingular elliptic curves. For the computation of the reduced modified Tate pairing over $\mathbb{F}_{3^{509}}$, we report calculation times of just 2.94 ms and 1.87 ms on the Intel Core2 and Intel Core i7 architectures, respectively. We also try to answer one important design question that surges: how many cores should be utilized for a given application?

Keywords: Tate pairing, supersingular curve, finite field arithmetic, multi-core.

1 Introduction

During the early years of this century it was generally assumed that computing cryptographic bilinear pairings was a computationally expensive task. Taking as a starting point the breakthrough introduced by Miller [23, 24], who proposed the first iterative approach to compute a cryptographic pairing, several authors focused their efforts on finding algorithmic improvements and shortcuts to further reduce the complexity of the so-called *Miller's Algorithm* [3, 4, 9, 18, 19, 29]. Those theoretical findings were experimentally validated by different means. At first, it was thought that the rich parallelization potential shown by hardware platforms could be exploited in order to produce faster and more compact pairing implementations. Through the years, this assumption has been confirmed in many research works (see for instance [8, 20, 28] for a comprehensive bibliography). Nevertheless, with the only exception of the ASIP in [20], all hardware accelerators reported in the open literature until today, have targeted low and medium security levels.

On the other hand, in the last few years a second wave of authors have investigated the challenges associated to the efficient implementation of pairings

in software platforms [12,16,17,21]. From the results reported by those research works, it appears that software pairing libraries can compete and even outperform their hardware counterparts. Furthermore, yet another way to exploit parallelism can be instrumented when the multi-core architectures introduced just recently by Intel are targeted. Multi-core architectures can be seen as a massive way to obtain parallelism via the concurrent usage of powerful individual processors that are tightly interconnected.

To our knowledge, the only bilinear pairing library targeting a multi-core architecture was reported in [12]. However the results presented in [12] were not too optimistic. After considering several scenarios, the authors concluded that on a Core2 64-bit platform, the best option to parallelize the computation of multiple pairings in their library was to perform one pairing on each core. They state that, *"if the requirement is for two pairing evaluations, the slightly moronic conclusion is that one can perform one pairing on each core [...], doubling the performance versus two sequential invocations of any other method that does not already use multi-core parallelism internally"* [12].

This paper is devoted to the design of a software library for the reduced modified Tate pairing on supersingular elliptic curves defined over finite fields of characteristics two and three. After a careful selection of the field arithmetic (Section 2) and pairing algorithms (Section 3) we show that multi-core architectures can be effectively used to provide significant computational speedups. A single-core version of our software[1] computes the reduced modified Tate pairing in 11.19 and 7.59 ms for the extension fields $\mathbb{F}_{2^{1223}}$ and $\mathbb{F}_{3^{509}}$, respectively, on an Intel Quad Core running at 2.4 GHz. Speedups of approximately 2.6× are obtained when using the four cores available in the target architecture (Section 4).

2 Finite Field Arithmetic Using SSE

2.1 Characteristic Two Field

Frobenius and Inverse-Frobenius Operators Let $f(x)$ be an irreducible polynomial of degree m over $\mathbb{F}_2$. Then, the binary extension field $\mathbb{F}_{2^m}$ is defined as $\mathbb{F}_{2^m} \cong \mathbb{F}_2[x]/\left(f(x)\right)$. Let a be an arbitrary element in $\mathbb{F}_{2^m}$, which in polynomial basis can be written as $a(x) = \sum_0^{m-1} a_i x^i$, with $a_i \in \mathbb{F}_2$ for $i = 0, 1, \ldots, m-1$. Let us also assume that the extension degree m can be expressed as $m = 2u+1$, with $u \geq 1$. Then, the Frobenius operator applied to a consists of computing $c = a^2 \bmod f(x)$, which can be obtained as

$$c = \sum_{i=0}^{u} a_i x^{2i} + x^m \sum_{i=1}^{u} a_{u+i} x^{2i-1} \bmod f(x) = a^L + a^H x^m \bmod f(x).$$

The field element c can be efficiently calculated in software by extracting the two half-length vectors a^L and a^H along with the computation of an $\frac{m}{2}$-bit multiplication by the per-field constant x^m. The subsequent reduction process modulo $f(x)$ is typically implemented in software by using XOR and shift operations.

[1] A full-open source code for benchmarking the library will be made available.

The inverse-Frobenius operator of a is computed by determining the unique field element $b \in \mathbb{F}_{2^m}$ such that $b^2 = a$ holds. The element b can be computed in terms of the square root of the field constant x as

$$b = \sum_{i=0}^{\lfloor \frac{m-1}{2} \rfloor} a_{2i} x^i + x^{\frac{1}{2}} \sum_{i=0}^{\lfloor \frac{m-3}{2} \rfloor} a_{2i+1} x^i \bmod f(x) = a_{\text{even}} + x^{\frac{1}{2}} a_{\text{odd}} \bmod f(x).$$

The efficient computation in software of b defined as above is performed by extracting the even and odd bits of a into the half length vectors a_{even} and a_{odd}, respectively. This should be followed by multiplying the half length vector a_{odd} times the pre-computed constant $x^{\frac{1}{2}}$.

In the case that the irreducible polynomial happens to be a trinomial of the form $f(x) = x^m + x^n + 1$ with m, n odd numbers, we have that $x^{\frac{1}{2}} = x^{\frac{m+1}{2}} + x^{\frac{n+1}{2}}$. Since $x^{\frac{1}{2}} \cdot a_{\text{odd}}$ has degree $m - 1$, it follows that we do not need to perform a polynomial modular reduction and hence the inverse Frobenius operator of an arbitrary element $a \in \mathbb{F}_{2^m}$ can be obtained by computing [10]

$$b = \sum_{i=0}^{\lfloor \frac{m-1}{2} \rfloor} a_{2i} x^i + \left(x^{\frac{m+1}{2}} + x^{\frac{n+1}{2}} \right) \cdot \sum_{i=0}^{\lfloor \frac{m-3}{2} \rfloor} a_{2i+1} x^i.$$

Multiplier We implemented this arithmetic block by using a variation of the left-to-right *comb* multiplication scheme presented in [22], one of the fastest multiplier schemes for binary fields reported in the open literature.

Let w be the processor word size in bits. Then, the number of processor words required for storing an arbitrary element in the field $\mathbb{F}_{2^m}$ is $s = \lceil \frac{m}{w} \rceil$. From these definitions, authors in [22] found that the computational complexity of their algorithm (excluding the one associated to the reduction process) was of $s(\frac{m}{4})$ w-bit XOR operations and a total of $(w/4 - 1)$ 4-bit left shift operations of a $2s$-word vector. Additionally, their method makes use of a look-up table containing sixteen s-word entries, which is queried a total of $s(\frac{m}{4})$ times. The look-up table is pre-computed at a cost of three 1-bit left shift operations of an s-word vector and eleven w-bit XOR operations.

In the case of the SSE instruction set, we have $w = 128$. Hence, we can invoke specialized instructions to perform any logic or arithmetic operation over a bank of 128-bit SSE register operators. It is also possible to manipulate the contents of the SSE registers by applying left/right shift/rotate operators over them. Those shift and rotate operations are executed with very high efficiency if the operand is shifted or rotated by a constant value multiple of 8 bits. This feature motivated us to propose a right-to-left comb multiplication scheme that trades all but one of the 4-bit left shift operations required by the multiplier in [22], with 8-bit right shift operations. In the rest of this subsection, we describe our formulation.

Let us define $n = 32 \cdot s$. It appears convenient to group the bit representation of a field element $a \in \mathbb{F}_{2^m}$ into 4-bit digits as follows:

$$a = (a_{m-1} \dots a_1 a_0) \Leftrightarrow a = (A_{n-1} \dots A_1 A_0),$$

where A_i, for $i = 0, 1, \ldots, n-1$, is defined as $A_i = \sum_{j=0}^{3} a_{4i+j} x^{4i+j}$. Notice that each 128-bit SSE register can store thirty-two such digits[2].

As it will be discussed below, it appears convenient to rearrange the n digits of the field element a into a $2 \times 16s$ matrix Idx as

$$\mathrm{Idx}[2][16s] = \begin{pmatrix} A_{n-1} \ldots A_{33}\ A_{31} \ldots A_3\ A_1 \\ A_{n-2} \ldots A_{32}\ A_{30} \ldots A_2\ A_0 \end{pmatrix} \tag{1}$$

In order to calculate the product $c = a \cdot b$ we prepare first a 16-entry look-up table by pre-computing

$$\mathrm{TblMul}[i] \leftarrow (i_3 x^3 + i_2 x^2 + i_1 x + i_0) \cdot b,$$

for $i = 0, 1, \ldots, 15$ and where $i = (i_3 i_2 i_1 i_0)_2$ is the binary expansion of i. Then, the polynomial product $c = ab$ can be computed as follows:

$$\begin{aligned} ab &= \sum_{i=0}^{n-1} A_i x^{4i} b = \sum_{i=0}^{16s-1} x^{8i} \left(A_{2i} + x^4 A_{2i+1}\right) b \\ &= \sum_{i=0}^{15} \sum_{j=0}^{s-1} x^{8i} \left(A_{32j+2i} + x^4 A_{32j+2i+1}\right) x^{128j} b \\ &= \sum_{i=0}^{15} \sum_{j=0}^{s-1} x^{8i} \left(\mathrm{TblMul}[A_{32j+2i}] + x^4 \mathrm{TblMul}[A_{32j+2i+1}]\right) x^{128j} \qquad (2) \\ &= \left(\sum_{i=0}^{15} \sum_{j=0}^{s-1} x^{8(i-16)} \mathrm{TblMul}[\mathrm{Idx}[1][16j+i]] x^{128j} \right) x^{128} + \\ &\quad \left(\sum_{i=0}^{15} \sum_{j=0}^{s-1} x^{8(i-16)} x^4 \mathrm{TblMul}[\mathrm{Idx}[0][16j+i]] x^{128j} \right) x^{128}. \end{aligned}$$

Note that in the last equality of Eq. (2) we used the matrix Idx as defined in Eq. (1), which allows us to recover the digits $A_{2(16j+i)}$ and $A_{2(16j+i)+1}$ as

$$\mathrm{Idx}[1][16j+i] = A_{2(16j+i)} \quad \text{and} \quad \mathrm{Idx}[0][16j+i] = A_{2(16j+i)+1}.$$

One can compute Eq. (2) as shown in Algorithm 1. It is worth stressing that:

- In step 5, we extract the bits of a in such a way that its digits A_i for $i = 0, \ldots, n-1$ are rearranged into a two dimensional array Idx[2][16s] as described in Eq. (1).
- The shift operations of step 10, namely, $x^{128}, x^{256}, \ldots, x^{1152}$, can be performed at no cost because they correspond to shifts by entire 128-bit words.
- In step 12, a 1-byte right rotation is applied over the contents of the $2s$-word accumulator R. This rotation operation is invoked thirty-two times.

[2] Note that the $v = 128 - (m \bmod 128)$ most significant bits of the last SSE register should be filled with zeroes.

Algorithm 1 SSE implementation of a right-to-left comb multiplier over $\mathbb{F}_{2^m}$.

Input: $a, b \in \mathbb{F}_{2^m}$.
Output: $c = a \cdot b \bmod f(x) \in \mathbb{F}_{2^m}$.

1. **for** $i \leftarrow 0$ **to** 15 **do**
2. Compute the binary expansion of $i = (i_3 i_2 i_1 i_0)_2$;
3. TblMul$[i] \leftarrow (i_3 x^3 + i_2 x^2 + i_1 x + i_0) \cdot b(x)$;
4. **end for**
5. Idx$[2][16s] \leftarrow$ extractIdx(a);
6. $R \leftarrow 0$;
7. **for** $k \leftarrow 0$ **to** 1 **do**
8. **for** $i \leftarrow 0$ **to** 15 **do**
9. **for** $j \leftarrow 0$ **to** $s - 1$ **do**
10. $R \leftarrow R +$ TblMul[Idx$[k][i + 16 \cdot j]]x^{128 \cdot j}$;
11. **end for**
12. $R \leftarrow$ rotRight_Byte$(R, 1)$
13. **end for**
14. **if** $i = 0$ **then**
15. $R \leftarrow$ rotLeft_bit$(R, 4)$;
16. **end if**
17. $R \leftarrow$ rotLeft_Byte$(R, 16)$;
18. **end for**
19. $c \leftarrow R \bmod f(x)$;
20. **return** C;

- In step 15, a 4-bit left rotation over the $2s$-word accumulator R is performed. This is the only left rotation by four bits included in the algorithm.
- In step 17, a left rotation by 16 bytes must be executed. This rotation is almost for free as it only implies the reassignment of the SSE registers.
- Finally in Step 19, a modular reduction with the polynomial $f(x)$ must be performed.

It is easy to verify that the computational cost of Algorithm 1 is of $(32 + 11)s$ XOR operations, $32s$ queries to the look-up table TblMul, thirty-two 1-byte right rotations of a $2s$-word vector, one 4-bit left rotation of a $2s$-word vector, plus the computational cost of the reduction operation of Step 19.

As a final remark we state that it is straightforward to generalize the multiplier of Algorithm 1 so that it can compute field multiplications over finite fields with characteristic $p > 2$.

Multiplicative Inverse We compute the multiplicative inverse of an arbitrary field element $a \in \mathbb{F}_{2^m}$ by implementing the Almost Inverse Algorithm [15, 26], which is a variant of the binary extended Euclidean algorithm.

2.2 Characteristic Three Field

Addition and Subtraction In 2002, Galbraith *et al.* [11] showed how to compute additions of two elements a, $b \in \mathbb{F}_3$ using 12 AND, OR, XOR and NOT Boolean functions. That same year, Harrison *et al.* [17] noted that this operation could be computed using only 7 OR and XOR logical instructions. This was considered the minimal number of logical operations for this arithmetic operation until Kawahara *et al.* [21] presented in 2008 an expression that only requires 6 logical instructions. However, our experiments, performed on a multi-core processor environment, showed that the expression in [17] consistently yields a shorter computation time than the one associated to the expression in [21]. We believe that this may be due to the fact that the formulation in [21] includes an ANDN instruction[3] that in our platform is implemented less efficiently than the logical set of functions proposed by Harrison *et al.* which, as it was mentioned above, only requires OR and XOR logic operations. Hence, we decided to adopt the expression reported in [17], which is briefly describe next. Each coefficient (trit) $a \in \mathbb{F}_3$ can be encoded as two bits a_h and a_l with $a = 2a_h + a_l$. The addition of two elements a, $b \in \mathbb{F}_3$ can then be computed as [17]

$$t = (a_l | b_h) \oplus (a_h | b_l), \quad c_l = t \oplus (a_h | b_h), \text{ and } \quad c_h = t \oplus (a_l | b_l).$$

As mentioned by the authors of [17], the order in which the above expression is evaluated has a major impact on the performance of its implementation in software. In fact, we use the following equivalent expression for computing $c = a + b$:

$$t = (a_l | a_h) \,\&\, (b_l | b_h), \quad c_l = t \oplus (a_l | b_l), \text{ and } \quad c_h = t \oplus (a_h | b_h).$$

Similarly, subtraction in $\mathbb{F}_3$ can be computed using only 7 instructions as

$$t = (a_l | a_h) \,\&\, (b_l | b_h), \quad c_l = t \oplus (a_l | b_h), \text{ and } \quad c_h = t \oplus (a_h | b_l).$$

Frobenius and Inverse-Frobenius Operators Let $f(x)$ be an irreducible polynomial of degree m over $\mathbb{F}_3$. Then, the ternary extension field $\mathbb{F}_{3^m}$ can be defined as $\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/\left(f(x)\right)$. Let a be an arbitrary element of that field, which can be written in canonical basis as $a = \sum_{i=0}^{m-1} a_i x^i$, $a_i \in \mathbb{F}_3$. Assume that the extension degree m is an integer of the form $m = 3u + r$, with $u \geq 1$ and $r \in \{0, 1, 2\}$. Then, the Frobenius operator applied to a consists of computing $c = a^3$, which can be obtained as [1]

$$c = a^3 \bmod f(x) = C_0 + x^m C_1 + x^{2m} C_2 \bmod P(x), \tag{3}$$

where $C_0 = \sum_{i=0}^{u} a_i x^{3i}$, $C_1 = \sum_{i=1}^{u+r-1} a_{i+u} x^{3i-r}$, and $C_2 = \sum_{i=r}^{u+r-1} a_{i+2u} x^{3i-2r}$.

[3] The logical function ANDN is defined in MMX and SSE implementations as x ANDN $y = x \,\&\, \bar{y}$.

One can evaluate Eq. (3) by determining the constants x^m and x^{2m}, which are per-field constants. The inverse-Frobenius operator of a is computed by determining the unique field element $b \in \mathbb{F}_{3^m}$ such that $b^3 = a$ holds. The element b can be computed as [2]

$$b = \sum_{i=0}^{u-s} a_{3i}x^i + x^{1/3} \cdot \sum_{i=0}^{u+r-2} a_{3i+1}x^i + x^{2/3} \cdot \sum_{i=0}^{u-1} a_{3i+2}x^i \bmod f(x), \qquad (4)$$

where $s = 1$ if $r = 0$, and $s = 0$ otherwise. Eq. (4) allows us to compute the inverse-Frobenius operator by performing two third-length polynomial multiplications with the per-field constants $x^{\frac{1}{3}}$ and $x^{\frac{2}{3}}$.

Table 1. Pre-computation look-up table when using the comb method for $a \in \mathbb{F}_{3^m}$ with a window size $v = 2$.

Entry	Value
[00]	0
[01]	a
[02]	$\sim$ [01]
[10]	[01] $\ll 1$
[11]	[10] + [01]
[12]	[10] + [02]
[20]	$\sim$ [10]
[21]	$\sim$ [12]
[22]	$\sim$ [11]

Multiplier We use here the same comb method discussed previously. As we did in characteristic two, we selected a window size $v = 4$, which in characteristic three means that we have to pre-compute a look-up table containing $3^4 = 81$ entries. Due to the fact that almost half of the entries can be obtained by performing one single logical NOT, pre-computing such look-up table requires a moderate computational effort. As an example, consider the case where we want to build a look-up table for a given element $a \in \mathbb{F}_{3^m}$, with a size of $v = 2$. Then, we have to pre-compute $3^2 = 9$ entries. Table 1 shows how to obtain those 9 elements, where $\sim$ and $\ll$ stand for the logical negation and left shift operations, respectively. As it can be seen in Table 1, 4 out of 9 entries can be computed using logical negation only. We also need to compute two $\mathbb{F}_{3^m}$ additions, one initialization to zero and one assignment of the element a. In the case of $v = 4$, generating the 81-entry look up table requires a computational effort of 40 logical negations, 36 field additions, and 3 left-shift operations.

Multiplicative Inverse In order to compute the multiplicative inverse d of a field element $b \in \mathbb{F}_{3^m}$, namely, $d = b^{-1} \bmod P$, we used the ternary variant of the binary extended Euclidean algorithm reported in [17].

2.3 Field-Arithmetic Implementation Timings

We present in Table 2 a timing performance comparison of our field arithmetic library against the timings reported by Hankerson *et al.* in [16][4]. In both works, the libraries were executed on a Intel Core2 processor running at 2.4 GHz. It is noticed that our multipliers in characteristic two and three are faster than their counterparts in [16]. However, the field multiplier for 256-bit prime fields reported in [16] easily outperforms all the other four multipliers listed in Table 2.

Table 2. A comparison of field arithmetic software implementations on an Intel Core2 processor (clock frequency: 2.4 GHz). All timings are reported in μs.

	Field	**Prime/polynomial**	x^p	$\sqrt[p]{x}$	**Mult**
Hankerson *et al.* [16]	$\mathbb{F}_{p256}$	256-bit prime	–	–	0.129
		Hamming weight 87			
	$\mathbb{F}_{2^{1223}}$	$x^{1223}+x^{255}+1$	0.250	0.208	3.417
	$\mathbb{F}_{3^{509}}$	$x^{509}-x^{477}+x^{445}+x^{32}-1$	0.375	0.500	3.208
This work	$\mathbb{F}_{2^{1223}}$	$x^{1223}+x^{255}+1$	0.200	0.312	2.266
	$\mathbb{F}_{3^{509}}$	$x^{509}-x^{318}-x^{191}+x^{127}+1$	0.375	0.406	1.720

3 Pairing Computation on Supersingular Curves in Characteristics Two and Three

In the following, we consider a supersingular elliptic curve $E/\mathbb{F}_{p^m}$ (where $p=2$ or 3) with a distortion map ψ. The point at infinity is denoted by $\mathcal{O}$. Let ℓ be a large prime factor of $N=\#E(\mathbb{F}_{p^m})$, and suppose that the embedding degree of the curve k is larger than 1 and that there are no points of order ℓ^2 in $E(\mathbb{F}_{p^{km}})$. Let $f_{n,P}$, for $n\in\mathbb{N}$ and $P\in E(\mathbb{F}_{p^m})[\ell]$, be a family of normalized $\mathbb{F}_{p^{km}}$-rational function with divisor $(f_{n,P})=n(P)-([n]P)-(n-1)(\mathcal{O})$. The modified Tate pairing of order ℓ is a non-degenerate and bilinear pairing given by the map

$$\begin{aligned}\hat{e}: E(\mathbb{F}_{p^m})[\ell]\times E(\mathbb{F}_{p^m})[\ell] &\longrightarrow \mathbb{F}^*_{p^{km}}/(\mathbb{F}^*_{p^{km}})^\ell\\ (P,Q) &\longmapsto f_{\ell,P}(\psi(Q)).\end{aligned}$$

Note that $\hat{e}(P,Q)$ is defined as a coset of $(\mathbb{F}^*_{p^{km}})^\ell$. However, $\mathbb{F}^*_{p^{km}}/(\mathbb{F}^*_{p^{km}})^\ell$ is cyclic of order ℓ and isomorphic to the group of ℓ-th roots of unity $\mu_\ell=\{u\in\overline{\mathbb{F}}^*_{p^m}: u^\ell=1\}\subseteq\mathbb{F}^*_{p^{km}}$. Hence, in order to obtain a unique representative, which is desirable for pairing-based protocols, it suffices to raise $f_{\ell,P}(\psi(Q))$ to the $(p^{km}-1)/\ell$-th power. This operation is often referred to as *final exponentiation*. We define the reduced modified Tate pairing as $\hat{e}_r(P,Q)=\hat{e}(P,Q)^{(p^{km}-1)/\ell}$.

[4] Our library was compiled using the MS Visual Studio 2008SP1 in 64-bit mode, and it was executed on the Windows XP 64 bit SP2 environment.

3.1 Miller's Algorithm

Miller [23,24] proposed the first iterative approach to compute the function $f_{\ell,P}$. By proving the equality of the divisors, he showed that

$$f_{a+b,P} = f_{a,P} \cdot f_{b,P} \cdot \frac{l_{[a]P,[b]P}}{v_{[a+b]P}},$$

where $l_{[a]P,[b]P}$ is the equation of the line through $[a]P$ and $[b]P$ (or the tangent line if $[a]P = [b]P$), $v_{[a+b]P}$ is the equation of the vertical line through $[a+b]P$, and $f_{1,P}$ is a constant function (usually, $f_{1,P} = 1$). We obtain a double-and-add algorithm for computing the rational function $f_{n,P}$ in $\lfloor \log_2 n \rfloor$ iterations. A nice property of supersingular elliptic curves is that multiplication by p is a relatively easy operation: it involves only a few Frobenius maps and additions over $\mathbb{F}_{p^m}$ (see for instance [3] for details) and a p-ary expansion of ℓ seems perfectly suited to the computation of $f_{\ell,P}(\psi(Q))$.

Several researchers focused on shortening the loop of Miller's algorithm (see for instance [18,19,29] for a comprehensive bibliography). Barreto *et al.* [3] introduced the η_T pairing as *"an alternative means of computing the Tate pairing on certain supersingular curves"* [25, page 108]. They suggest to compute $\hat{e}_r(P,Q)$ using an order $T \in \mathbb{Z}$ that is smaller than ℓ. Their main result is a lemma giving a method to select T such that $\eta_T(P,Q)$ is a non-degenerate bilinear pairing [3]. In the case of characteristics two and three, they show that one can half the number of basic Miller's iterations by chosing $T = p^m - N$:

$$\eta_T(P,Q) = \begin{cases} f_{T,P}(\psi(Q)) \text{ if } T > 0, \text{ or} \\ f_{-T,-P}(\psi(Q)) \text{ if } T < 0. \end{cases}$$

It is worth noticing that T has a low Hamming weight and the computation of $[T]P$ (or $[-T]P$) requires $(m+1)/2$ multiplications by p and a single addition. It is therefore possible to pre-compute multiples of P by means of Frobenius maps and to parallelize Miller's algorithm on several cores. Multiplications over $\mathbb{F}_{p^{km}}$ are of course necessary to obtain $\hat{e}(P,Q)$ from the partial results computed on each core.

3.2 Reduced Modified Tate Pairing in Characteristic Two

We follow [6, Algorithm 1] to compute $\hat{e}(P,Q)$ on a supersingular curve $E/\mathbb{F}_{2^m}$, m and odd number and with embedding degree $k = 4$ given by $E : y^2 + y = x^3 + x + b$, where m is odd and $b \in \{0,1\}$. The reduced modified Tate pairing is defined by [3,6]:

$$\hat{e}_r(P,Q) = \eta_T([2^m]P,Q)^{\frac{2^{4m}-1}{N}}.$$

Loop unrolling does not allow one to reduce the number of multiplications over $\mathbb{F}_{2^m}$ and Miller's algorithm requires $(m-1)/2$ iterations which can be parallelized

on several cores[5]. Final exponentiation consists of raising $\hat{e}(P,Q)$ to the exponent $M = \frac{2^{4m}-1}{N} = (2^{2m}-1)\cdot(2^m+1-\nu 2^{(m+1)/2})$ [3], where $\nu = (-1)^b$ when $m \equiv 1, 7 \pmod 8$ and $\nu = (-1)^{1-b}$ in all other cases. We perform this operation according to a slightly optimized version of [6, Algorithm 3] (see Appendices A.1 and A.2 for technical details):

- **Raising to the $(2^m + 1)$-th power.** Raising the outcome of Miller's algorithm to the $\left(2^{2m} - 1\right)$-th power produces an element $U \in \mathbb{F}_{2^{4m}}$ of order $2^{2m} + 1$. This property allows one to save a multiplication over $\mathbb{F}_{2^{4m}}$ when raising U to the $(2^m + 1)$-th power compared to [6, page 304].
- **Raising to the $2^{\frac{m+1}{2}}$-th power.** Beuchat *et al.* [8] exploited the linearity of the Frobenius map in order to reduce the cost of successive cubings over $\mathbb{F}_{3^{6m}}$. The same approach can be straightforwardly transposed to characteristic two: raising an element of $\mathbb{F}_{2^{4m}}$ to the 2^i-th power involves $4i$ squarings and at most four additions over $\mathbb{F}_{2^m}$.

3.3 Reduced Modified Tate Pairing in Characteristic Three

We consider a supersingular curve $E/\mathbb{F}_{3^m}$ with embedding degree $k = 6$ defined by $E : y^2 = x^3 - x + b$, where m is coprime to 6 and $b \in \{-1, 1\}$. According to [3, 6], we have

$$\hat{e}_r(P,Q) = \eta_T\left(\left[-\mu b 3^{\frac{3m-1}{2}}\right] P, Q\right)^{\frac{3^{6m}-1}{N}},$$

where $\mu = 1$ when $m \equiv 1, 11 \pmod{12}$, or $\mu = -1$ otherwise. There are several ways to compute the η_T pairing (see for instance [3, 7, 8]) and the choice of an algorithm depends on the target architecture. Here, we decided to minimize the number of arithmetic operations over $\mathbb{F}_{3^m}$ and applied the well-known loop unrolling technique [13] to [7, Algorithm 3] (technical details are provided in Appendix B). This approach allows us to save several multiplications over $\mathbb{F}_{3^m}$ compared to the original algorithm. Final exponentiation is carried out according to [7, 8].

4 Results and Comparisons

We list in Table 3 the timings achieved by our library for low, medium and high security levels (66, 89, and 128 bits, respectively), including the performance obtained when using one, two, and four Intel Core2 processors. Our library

[5] Shirase *et al.* propose a loop unrolling technique in reference [27] and claim that they reduce the computation time by 14.3%. However, they assume that additions and multiplications by small constants are almost free, and inverse Frobenius maps over $\mathbb{F}_{2^m}$ are m times more expensive than Frobenius maps. Such estimates do not hold in our context (see for instance Table 2) and we did not investigate further the approach by Shirase *et al.*

was compiled using the MS Visual Studio 2008SP1 in 64-bit mode, and it was executed on the Windows XP 64 bit SP2 environment.

For comparison purposes, we also include in Table 3 the performance reported by Hankerson *et al.* [16], which is the fastest pairing library that we know of. The work by Grabher *et al.* [12] is also of interest as it is the only pairing library preceding this work that reports a multi-core platform implementation. All the pairing libraries included in Table 3 were implemented on an Intel Core2 processor. Table 4 shows the timings achieved by our library when implemented on an Intel core i7 multi-processor platform running at 2.9 GHz. Finally, in Table 5 we list some of the fastest hardware accelerators for the Tate pairing reported at low, medium and high security levels.

Table 3. Performance comparison of software implementations for pairings on an Intel Core2 processor.

	Curve	**Security [bits]**	**# of cores**	**Freq. [GHz]**	**Calc. time [ms]**
This work	$E(\mathbb{F}_{3^{97}})$	66	1	2.6	0.15
	$E(\mathbb{F}_{3^{97}})$	66	2	2.6	0.09
This work	$E(\mathbb{F}_{3^{193}})$	89	1	2.6	0.98
	$E(\mathbb{F}_{3^{193}})$	89	2	2.6	0.55
Hankerson *et al.* [16]	$E(\mathbb{F}_{p256})$	128	1	2.4	6.25
	$E(\mathbb{F}_{2^{1223}})$	128	1	2.4	16.25
	$E(\mathbb{F}_{3^{509}})$	128	1	2.4	13.75
Grabher *et al.* [12]	$E(\mathbb{F}_{p256})$	128	1	2.4	9.71
	$E(\mathbb{F}_{p256})$	128	2	2.4	6.01
This work	$E(\mathbb{F}_{2^{1223}})$	128	1	2.4	11.19
	$E(\mathbb{F}_{2^{1223}})$	128	2	2.4	6.72
	$E(\mathbb{F}_{2^{1223}})$	128	4	2.4	4.22
	$E(\mathbb{F}_{3^{509}})$	128	1	2.4	7.59
	$E(\mathbb{F}_{3^{509}})$	128	2	2.4	4.31
	$E(\mathbb{F}_{3^{509}})$	128	4	2.4	2.94

Grabher *et al.* reported in [12] a multi-core implementation of the Ate pairing defined over a Barreto-Naehrig (BN) curve [5], when using a 256-bit prime. Since the BN curves have an embedding degree of $k = 12$, this implies a 128-bit security level. As shown in Table 3, our pairing implementation in characteristic three is faster than the prime field pairing library reported in [12].

In Table 4 we report a calculation time for the reduced modified Tate pairing of just 3.08 ms and 1.87 ms for characteristics two and three, respectively. This performance, that was obtained on a two Intel quad-core i7 multi-processor platform, appears to be the fastest pairing timings yet reported.

Although our software implementation outperforms several hardware architectures previously reported for low and medium levels of security, when we compare our results against the ones in [8, 28] (Table 5), we see that there still exists a large gap between software and hardware pairing implementations for

moderate security levels. The computation of the reduced modified Tate pairing over $\mathbb{F}_{3^{193}}$ on a Virtex-4 LX FPGA reported in [8] with a medium speed grade is for instance roughly fifty times faster than our software timings. Depending on the application, this speedup may justify the usage of large FPGAs which are now available in servers and supercomputers such as the SGI Altix 4700 platform.

Table 4. Implementations timings for the reduced modified Tate pairing at the 128-bit security level on an Intel core i7 processor (clock frequency: 2.9 GHz).

Curve	# of cores	Calc. time [ms]
$E(\mathbb{F}_{3^{509}})$	1	5.22
$E(\mathbb{F}_{3^{509}})$	2	3.16
$E(\mathbb{F}_{3^{509}})$	4	2.31
$E(\mathbb{F}_{3^{509}})$	8	1.87

Curve	# of cores	Calc. time [ms]
$E(\mathbb{F}_{2^{1223}})$	1	7.94
$E(\mathbb{F}_{2^{1223}})$	2	4.53
$E(\mathbb{F}_{2^{1233}})$	4	3.13
$E(\mathbb{F}_{2^{1223}})$	8	3.08

Table 5. Some hardware accelerators for the Tate pairing.

	Curve	Security [bits]	Platform	Area	Freq. [MHz]	Calc. time [ms]
Shu *et al.* [28]	$E(\mathbb{F}_{2^{239}})$	66	xc4vlx200	29920 slices	100	0.0365
Beuchat *et al.* [8]	$E(\mathbb{F}_{3^{97}})$	66	xc4vlx60-11	18683 slices	179	0.0048
Shu *et al.* [28]	$E(\mathbb{F}_{2^{457}})$	88	xc4vlx200	58956 slices	100	0.1
Beuchat *et al.* [8]	$E(\mathbb{F}_{3^{193}})$	89	xc4vlx100-11	47433 slices	167	0.01
Shu *et al.* [28]	$E(\mathbb{F}_{2^{557}})$	96	xc4vlx200	37931 slices	66	0.6758
Kammler *et al.* [20]	$E(\mathbb{F}_{p256})$	128	130 nm CMOS	97 kGates	338	15.8

5 Conclusion

In this work we presented the multi-core implementation of a software library that is able to compute the reduced modified Tate pairing on supersingular elliptic curves at a high speed.

The sequential timings reported in this work are significantly faster than the ones achieved in [16] for pairings computed over characteristics two and three fields. We would like to interpret these results as a reduction of the gap between prime fields and binary/ternary fields.

In the light of the results obtained here, one important design question that surges is: how many cores should be utilized by a given application? As discussed in the Appendixes, our pairing library successfully parallelize the computation

of Miller's algorithm. However, if we use n cores for the implementation of the Miller's algorithm, in order to combine the n partial products generated by the n parallel sub-loops executed in each core, we are forced to add $n-1$ extra field multiplications over $\mathbb{F}_{p^{km}}$. Furthermore, due to the dependencies among the different operations involved in the final exponentiation step, this portion of the pairing has to be computed sequentially. As shown in Table 4, these two factors cause the acceleration achieved by an n-core implementation to be always less than the ideal $n\times$ speedup factor. From Table 4, we can see for instance that the acceleration provided by the eight-core implementation is modest compared with the timings achieved by the four-core one. On the other hand, when comparing the timings of the single-core and dual-core implementations, the acceleration factor is roughly $1.70\times$ for both, characteristics two and three.

Our future work includes the implementation of pairings for large characteristics on ordinary curves and the SSE implementation of our pairing library using the built-in carry-less 64-bit multiplier recently announced by Intel [14].

Acknowledgments

The authors would like to thank Jérémie Detrey for his valuable comments.

References

1. O. Ahmadi and F. Rodríguez-Henríquez. Low complexity cubing and cube root computation over $\mathbb{F}_{3^m}$ in standard basis. Cryptology ePrint Archive, Report 2009/070, 2009.
2. P.S.L.M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004.
3. P.S.L.M. Barreto, S.D. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007.
4. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368. Springer, 2002.
5. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
6. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A comparison between hardware accelerators for the modified tate pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 297–315. Springer, 2008.
7. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and arithmetic operators for computing the η_T pairing in characteristic three. *IEEE Transactions on Computers*, 57(11):1454–1468, November 2008.
8. J.-L. 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.

9. I. Duursma and H.S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C.S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.
10. K. Fong, D. Hankerson, J. López, and A. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, August 2004.
11. S.D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory – ANTS V*, number 2369 in Lecture Notes in Computer Science, pages 324–337. Springer, 2002.
12. P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In *Selected Areas in Cryptography – SAC 2008*, number 5381 in Lecture Notes in Computer Science, pages 34–49. Springer, 2008.
13. R. Granger, D. Page, and M. Stam. On small characteristic algebraic tori in pairing-based cryptography. *LMS Journal of Computation and Mathematics*, 9:64–85, March 2006.
14. S. Gueron and M.E. Kounavis. Carry-less multiplication and its usage for computing the GCM mode. Intel Corporation White Paper, May 2009.
15. D. Hankerson, J. López Hernandez, and A.J. Menezes. Software implementation of elliptic curve cryptography over binary fields. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, number 1965 in Lecture Notes in Computer Science, pages 1–24. Springer, 2000.
16. D. Hankerson, A. Menezes, and M. Scott. *Software Implementation of Pairings*, chapter 12, pages 188–206. Cryptology and Information Security Series. IOS Press, 2009.
17. K. Harrison, D. Page, and N.P. Smart. Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems. *LMS Journal of Computation and Mathematics*, 5:181–193, November 2002.
18. F. Hess. Pairing lattices. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 18–38. Springer, 2008.
19. F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, October 2006.
20. 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.
21. Y. Kawahara, K. Aoki, and T. Takagi. Faster implementation of η_T pairing over GF(3^m) using minimum number of logical instructions for GF(3)-addition. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 282–296. Springer, 2008.
22. J. López and R. Dahab. High-speed software multiplication in $\mathbb{F}_{2^m}$. In B.K. Roy and E. Okamoto, editors, *Progress in Cryptology – INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 203–212. Springer, 2000.
23. V.S. Miller. Short programs for functions on curves. Available at http://crypto.stanford.edu/miller, 1986.
24. V.S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
25. C. Ó hÉigeartaigh. *Pairing Computation on Hyperelliptic Curves of Genus* 2. PhD thesis, Dublin City University, 2006.
26. R. Schroeppel, H. Orman, S.W. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. In D. Coppersmith, editor, *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science, pages 43–56. Springer, 1995.

27. M. Shirase, T. Takagi, D. Choi, D. Han, and H. Kim. Efficient computation of Eta pairing over binary field with Vandermonde matrix. *ETRI Journal*, 31(2):129–139, April 2009.
28. C. Shu, S. Kwon, and K. Gaj. Reconfigurable computing approach for Tate pairing cryptosystems over binary fields. *IEEE Transactions on Computers*, 2009. To appear.
29. F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008.

A Reduced Modified Tate Pairing in Characteristic Two

Table 6 summarizes the parameters of the supersingular curve considered to compute $\hat{e}_r(P,Q)$ in characteristic two. Noting $T' = -\nu T$ and $P' = [-\nu 2^m]P$, we have:

$$\begin{aligned}\hat{e}_r(P,Q) &= f_{T',P'}(\psi(Q))^M \\ &= \left(l_{P'}(\psi(Q)) \cdot \left(\prod_{j=0}^{\frac{m-1}{2}} g_{\left[2^{\frac{m-1}{2}-j}\right]P'}(\psi(Q))^{2^j} \right) \right)^M , \end{aligned} \tag{5}$$

where, for all $V \in E(\mathbb{F}_{2^m})[\ell]$, l_V is the equation of the line corresponding to the addition of $[\nu]V$ with $\left[2^{\frac{m+1}{2}}\right]V$ and g_V is the rational function defined over $E(\mathbb{F}_{2^{4m}})[\ell]$ corresponding to the straight line in doubling V. More precisely, we have:

$$\left(g_{\left[2^{\frac{m-1}{2}-j}\right]P'}(\psi(Q)) \right)^{2^j} = (x_{P'}^{2^{-j}} + \alpha) \cdot (x_Q^{2^j} + \alpha) + y_{P'}^{2^{-j}} + y_Q^{2^j} + \beta + (x_{P'}^{2^{-j}} + x_Q^{2^j} + \alpha)s + t, \text{ and}$$

$$l_{P'}(\psi(Q)) = g_{\left[2^{\frac{m-1}{2}}\right]P'}(\psi(Q)) + x_{P'}^2 + x_Q + \alpha + s.$$

Algorithm 2 describes the computation of $\hat{e}_r(P,Q)$ according to Eq. (5) (this algorithm is based on [6, Algorithm 1]). The equations of all straight lines can be pre-computed by storing all square roots of x_P and y_P, as well as all squares of x_Q and y_Q (lines 12 and 13 that can be computed in parallel on two cores). Then, one can split the execution of Miller's algorithm (lines 16 to 20) into several parts that are run concurrently. We perform the final exponentiation according to an improved version of [6, Algorithm 3] detailed in the following.

A.1 Raising an Element of Order $2^{2m}+1$ to the (2^m+1)-th Power over $\mathbb{F}_{2^{4m}}$

Let F, $U \in \mathbb{F}_{2^{4m}}$ and assume that $U = F^{2^{2m}-1}$. According to Fermat's little theorem, the result of raising to the $\left(2^{2m}-1\right)$-th power produces an element of

Table 6. Supersingular curves over $\mathbb{F}_{2^m}$.

Underlying field	$\mathbb{F}_{2^m}$, where m is an odd integer
Curve	$E : y^2 + y = x^3 + x + b$, with $b \in \{0, 1\}$
Number of rational points	$N = 2^m + 1 + \nu 2^{(m+1)/2}$, with $\nu = (-1)^\delta$ and $\delta = \begin{cases} b & \text{if } m \equiv 1, 7 \pmod 8, \\ 1-b & \text{if } m \equiv 3, 5 \pmod 8, \end{cases}$
Embedding degree	$k = 4$
Distortion map	$\psi : E(\mathbb{F}_{2^m})[\ell] \longrightarrow E(\mathbb{F}_{2^{4m}})[\ell] \setminus E(\mathbb{F}_{2^m})[\ell]$ $(x, y) \longmapsto (x + s^2, y + sx + t)$ with s and $t \in \mathbb{F}_{2^{4m}}$ satisfying $s^2 = s + 1$ and $t^2 = t + s$
Tower field	$\mathbb{F}_{2^{4m}} = \mathbb{F}_{2^m}[s, t] \cong \mathbb{F}_{2^m}[X, Y]/(X^2 + X + 1, Y^2 + Y + X)$
Final exponentiation	$M = \left(2^{2m} - 1\right) \cdot \left(2^m + 1 - \nu 2^{(m+1)/2}\right)$
Parameters of Algorithm 2	$\alpha = \begin{cases} 0 & \text{if } m \equiv 3 \pmod 4, \text{ or} \\ 1 & \text{if } m \equiv 1 \pmod 4, \text{ and} \end{cases}$ $\beta = \begin{cases} b & \text{if } m \equiv 1, 3 \pmod 8, \text{ or} \\ 1-b & \text{if } m \equiv 5, 7 \pmod 8 \end{cases}$

order $2^{2m} + 1$, i.e. $U^{2^{2m}+1} = 1$. Let us write

$$U = \underbrace{u_0 + u_1 s}_{U_0} + \underbrace{(u_2 + u_3 s)}_{U_1} t,$$

where u_0, u_1, u_2, $u_3 \in \mathbb{F}_{2^m}$ and U_0, $U_1 \in \mathbb{F}_{2^{2m}}$. Since $t^{2^{2m}} = 1 + t$, we have:

$$U^{2^{2m}+1} = (U_0 + U_1 t)(U_0 + U_1 t)^{2^{2m}} = U_0^2 + U_0 U_1 + U_1^2 s = 1.$$

Therefore,

$$\begin{aligned} u_0 u_2 + u_1 u_3 &= u_0^2 + u_1^2 + u_3^2 + 1, \text{ and} \\ u_0 u_3 + u_1 u_2 + u_1 u_3 &= u_1^2 + u_2^2. \end{aligned}$$

Let $\alpha = 0$ when $m \equiv 3 \pmod 4$ and $\alpha = 1$ when $m \equiv 1 \pmod 4$. Seeing that $s^{2^m} = s + 1$ and $t^{2^m} = t + s + \alpha + 1$, we obtain:

$$U^{2^m} = \begin{cases} (u_0 + u_1 + u_3) + (u_1 + u_2)s + (u_2 + u_3)t + u_3 st & \text{if } \alpha = 1, \\ (u_0 + u_1 + u_2) + (u_1 + u_2 + u_3)s + (u_2 + u_3)t + u_3 st & \text{if } \alpha = 0. \end{cases}$$

A first solution to compute U^{2^m+1} would be to multiply U^{2^m} by U. There is however a faster way to raise U to the power of $2^m + 1$. Defining $m_0 = u_0 u_1$,

Algorithm 2 Computation of the reduced modified Tate pairing in characteristic two.

Input: P, $Q \in \mathbb{F}_{2^m}[\ell]$.
Output: $\hat{e}_r(P,Q) \in \mathbb{F}^*_{2^{4m}}$.

1. $x_P \leftarrow x_P + 1$; (1 XOR)
2. $y_P \leftarrow x_P + y_P + \alpha + \bar{\delta}$; ($\alpha + \bar{\delta}$ XOR, 1 A)
3. $u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$ (2α XOR)
4. $g_0 \leftarrow u \cdot v + y_P + y_Q + \beta$; (1 M, 2 A, β XOR)
5. $g_1 \leftarrow u + x_Q$; $g_2 \leftarrow v + x_P^2$; (1 S, 2 A)
6. $G \leftarrow g_0 + g_1 s + t$;
7. $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$; (1 A, 1 XOR)
8. $F \leftarrow L \cdot G$; (2 M, 1 S, 5 A, 2 XOR)
9. $x_P[0] \leftarrow x_P$; $y_P[0] \leftarrow y_P$;
10. $x_Q[0] \leftarrow x_Q$; $y_Q[0] \leftarrow y_Q$;
11. **for** $j = 1$ to $\frac{m-1}{2}$ **do**
12. $\quad x_P[j] \leftarrow \sqrt{x_P[j-1]}$; $x_Q[i] \leftarrow x_Q[i-1]^2$; (1 R, 1 S)
13. $\quad y_P[j] \leftarrow \sqrt{y_P[j-1]}$; $y_Q[i] \leftarrow y_Q[i-1]^2$; (1 R, 1 S)
14. **end for**
15. **for** $j = 1$ to $\frac{m-1}{2}$ **do**
16. $\quad u \leftarrow x_P[j] + \alpha$; $v \leftarrow x_Q[j] + \alpha$ (2α XOR)
17. $\quad g_0 \leftarrow u \cdot v + y_P[j] + y_Q[j] + \beta$; (1 M, 2 A, β XOR)
18. $\quad g_1 \leftarrow u + x_Q[j]$; (1 A)
19. $\quad G \leftarrow g_0 + g_1 s + t$;
20. $\quad F \leftarrow F \cdot G$; (6 M, 14 A)
21. **end for**
22. **return** F^M;

$m_1 = u_0u_3$, $m_2 = u_1u_2$, and $m_3 = u_2u_3$, we have:

$$\begin{aligned} U^{2^m+1} &= (u_0u_1 + u_0u_3 + u_1u_2 + u_0^2 + u_1^2) + \\ &\quad (\underbrace{u_0u_2 + u_1u_3}_{=u_0^2+u_1^2+u_3^2+1} + u_1u_2 + u_2u_3 + u_2^2 + u_3^2)s + \\ &\quad (u_0u_3 + u_1u_2 + u_2u_3 + u_2^2 + u_3^2)t + (u_2u_3 + u_2^2 + u_3^2)st \\ &= (m_0 + m_1 + m_2 + (u_0 + u_1)^2) + (m_2 + m_3 + (u_0 + u_1 + u_2)^2 + 1)s + \\ &\quad (m_1 + m_2 + m_3 + (u_2 + u_3)^2)t + (m_3 + (u_2 + u_3)^2)st, \end{aligned}$$

when $\alpha = 1$, and

$$\begin{aligned} U^{2^m+1} &= (\underbrace{u_0u_2 + u_1u_3}_{=u_0^2+u_1^2+u_3^2+1} + u_0u_1 + u_1u_2 + u_0^2 + u_1^2) + \\ &\quad (\underbrace{u_0u_2 + u_1u_3}_{=u_0^2+u_1^2+u_3^2+1} + u_0u_3 + u_2u_3 + u_2^2 + u_3^2)s + \\ &\quad (u_0u_3 + u_1u_2)t + (u_2u_3 + u_2^2 + u_3^2)st \\ &= (m_0 + m_2 + u_3^2 + 1) + (m_1 + m_3 + (u_0 + u_1 + u_2)^2 + 1)s + \\ &\quad (m_1 + m_2)t + (m_3 + u_2^2 + u_3^2)st, \end{aligned}$$

when $\alpha = 0$. Thus, computing U^{2^m+1} involves only four multiplications, three squarings, and eleven additions over $\mathbb{F}_{2^m}$ (Algorithm 3). This approach allows us to save one multiplication over $\mathbb{F}_{2^m}$ compared to [6].

Algorithm 3 Computation of U^{2^m+1} over $\mathbb{F}_{2^{4m}}$, where U is an element of order $2^{2m} + 1$.

Input: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$ with $U^{2^{2m}+1} = 1$.
Output: $V = U^{2^m+1}$.

1. $m_0 \leftarrow u_0 \cdot u_1$; $m_1 \leftarrow u_0 \cdot u_3$; $m_2 \leftarrow u_1 \cdot u_2$; $m_3 \leftarrow u_2 \cdot u_3$; (4 M)
2. $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow a_0 + u_2$; (2 A)
3. $s_1 \leftarrow a_1^2$; (1 S)
4. **if** $\alpha = 1$ **then**
5. $\quad a_2 \leftarrow u_2 + u_3$; $a_3 \leftarrow m_1 + m_2$; (2 A)
6. $\quad s_0 \leftarrow a_0^2$; $s_2 \leftarrow a_2^2$; (2 S)
7. $\quad v_3 \leftarrow m_3 + s_2$; (1 A)
8. $\quad v_2 \leftarrow v_3 + a_3$; (1 A)
9. $\quad v_1 \leftarrow m_2 + m_3 + s_1 + 1$; (3 A)
10. $\quad v_0 \leftarrow m_0 + a_3 + s_0$; (2 A)
11. **else**
12. $\quad s_0 \leftarrow u_2^2$; $s_2 \leftarrow u_3^2$; (2 S)
13. $\quad v_0 \leftarrow m_0 + m_2 + s_2 + 1$; (3 A)
14. $\quad v_1 \leftarrow m_1 + m_3 + s_1 + 1$; (3 A)
15. $\quad v_2 \leftarrow m_1 + m_2$; (1 A)
16. $\quad v_3 \leftarrow m_3 + s_0 + s_2$; (2 A)
17. **end if**
18. **Return** $v_0 + v_1 s + v_2 t + v_3 st$;

A.2 Computing $U^{2^{\frac{m+1}{2}}}$ over $\mathbb{F}_{2^{4m}}$

Let $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$. Noting that $s^{2^i} = s + i$ and $t^{2^i} = t + is + \lfloor \frac{i \bmod 4}{2} \rfloor$, we obtain the following formulae for U^{2^i}, depending on the

value of i modulo 4:

$$U^{2^i} = \begin{cases} u_0^{2^i} + u_1^{2^i} s + u_2^{2^i} t + u_3^{2^i} st & \text{when } i \equiv 0 \pmod 4, \\ (u_0 + u_1 + u_3)^{2^i} & \\ \quad +(u_1 + u_2)^{2^i} s + (u_2 + u_3)^{2^i} t + u_3^{2^i} st & \text{when } i \equiv 1 \pmod 4, \\ (u_0 + u_2)^{2^i} + (u_1 + u_3)^{2^i} s + u_2^{2^i} t + u_3^{2^i} st & \text{when } i \equiv 2 \pmod 4, \\ (u_0 + u_1 + u_2)^{2^i} + (u_1 + u_2 + u_3)^{2^i} s & \\ \quad +(u_2 + u_3)^{2^i} t + u_3^{2^i} st & \text{when } i \equiv 3 \pmod 4. \end{cases}$$

According to the value of $(m+1)/2 \bmod 4$, the computation of $U^{2^{\frac{m+1}{2}}}$ requires $2m+2$ squarings and at most four additions over $\mathbb{F}_{2^m}$.

B Reduced Modified Tate Pairing in Characteristic Three

In the following, we consider the computation of the reduced modified Tate pairing in characteristic three on several cores. Table 7 summarizes the parameters of the supersingular curve. Noting $T' = -\mu b T$ and $P' = (x_{P'}, y_{P'}) = \left[3^{\frac{3m-1}{2}}\right] P$, we have to compute:

$$\begin{aligned} \hat{e}_r(P,Q) &= f_{T',P'}(\psi(Q))^M \\ &= \left(l_{P'}(\psi(Q)) \cdot \left(\prod_{j=0}^{\frac{m-1}{2}} g_{\left[3^{\frac{m-1}{2}-j}\right]P'}(\psi(Q))^{3^j} \right) \right)^M , \end{aligned} \tag{6}$$

where, for all $V \in E(\mathbb{F}_{3^m})[\ell]$, l_V is the equation of the line corresponding to the addition of $[\mu b]V$ with $\left[3^{\frac{m+1}{2}}\right] V$, and g_V is the rational function introduced by Duursma and Lee [9] and having divisor $(g_V) = 3(V) + ([-3]V) - 4(\mathcal{O})$. Expanding everything, we obtain the following expressions:

$$l_{P'}(\psi(Q)) = y_Q \sigma + \lambda y_{P'}(x_{P'} + x_Q - \nu b - \rho), \text{ and}$$

$$g_{\left[3^{\frac{m-1}{2}-j}\right]P'}(\psi(Q))^{3^j} = -\lambda y_{P'}^{3^{-j}} y_Q^{3^j} \sigma - \left(x_{P'}^{3^{-j}} + x_Q^{3^j} - \nu b - \rho \right)^2 .$$

It is worth noticing that the Duursma-Lee functions can be pre-computed by building a table of all cube roots of x_P and y_P as well as all cubes of x_Q and y_Q.

Beuchat *et al.* described an algorithm to compute $\hat{e}_r(P,Q)$ according to Eq. (6) (see [7, Algorithm 3]). They took advantage of the sparsity of $l_{P'}$ and g_V to reduce the cost of the first multiplication over $\mathbb{F}_{3^m}$ and needed therefore $\frac{m-1}{2}$ iterations of Miller's algorithm to accumulate the remaining products. We propose to optimize further the algorithm by computing two iterations at a time and obtain Algorithm 4 for the case where $\frac{m-1}{2}$ is even. When $\frac{m-1}{2}$ is odd, one has to restrict the loop on j from 1 to $\frac{m-3}{4}$, and perform the last product by

Table 7. Supersingular curves over $\mathbb{F}_{3^m}$ (reprinted from [8]).

Underlying field	$\mathbb{F}_{3^m}$, where m is coprime to 6
Curve	$E : y^2 = x^3 - x + b$, with $b \in \{-1, 1\}$
Number of rational points	$N = \#E(\mathbb{F}_{3^m}) = 3^m + 1 + \mu b 3^{(m+1)/2}$, with $\mu = \begin{cases} +1 & \text{if } m \equiv 1, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 5, 7 \pmod{12} \end{cases}$
Embedding degree	$k = 6$
Distortion map	$\psi : E(\mathbb{F}_{3^m})[\ell] \longrightarrow E(\mathbb{F}_{3^{6m}})[\ell] \setminus E(\mathbb{F}_{3^m})[\ell]$ $(x, y) \longmapsto (\rho - x, y\sigma)$ with $\rho \in \mathbb{F}_{3^{3m}}$ and $\sigma \in \mathbb{F}_{3^{2m}}$ satisfying $\rho^3 = \rho + b$ and $\sigma^2 = -1$
Tower field	$\mathbb{F}_{3^{6m}} = \mathbb{F}_{3^m}[\rho, \sigma] \cong \mathbb{F}_{3^m}[X, Y]/(X^3 - X - b, Y^2 + 1)$
Final exponentiation	$M = \left(3^{3m} - 1\right) \cdot (3^m + 1) \cdot \left(3^m + 1 - \mu b 3^{(m+1)/2}\right)$
Parameters of Algorithm 4	$\lambda = \begin{cases} +1 & \text{if } m \equiv 7, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 1, 5 \pmod{12}, \text{ and} \end{cases}$ $\nu = \begin{cases} +1 & \text{if } m \equiv 5, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 1, 7 \pmod{12} \end{cases}$

means of an iteration of the original loop. Thanks to the sparsity of g_V, the cost of a double iteration is of 25 multiplications over $\mathbb{F}_{3^m}$, whereas two iterations of the original loop involve 28 multiplications [7]. The key observation is that the sparse multiplication over $\mathbb{F}_{3^{6m}}$ on line 15 requires only 8 multiplications over $\mathbb{F}_{3^m}$. Keeping in mind that our SSE implementation of multiplication over $\mathbb{F}_{3^m}$ involves a precomputation step depending on the second operand, we designed Algorithm 5 where several multiplications over $\mathbb{F}_{3^m}$ share a common operand (lines 5 and 6).

Since lines 9 and 10 of Algorithm 4 do not present dependencies, the precomputation of the Duursma-Lee functions can be performed in parallel on two cores. Then, one can split the execution of Miller's algorithm (lines 13 to 16) into several parts that are run concurrently.

Algorithm 4 Unrolled loop for computing the reduced modified Tate pairing in characteristic three when $\frac{m-1}{2}$ is even.

Input: $P, Q \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\hat{e}_r(P,Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_P \leftarrow \sqrt[3]{x_P} - (\nu+1)b$; (1 R, 1 A when $m \equiv 5, 11 \bmod 12$)
2. $y_P \leftarrow \lambda \sqrt[3]{y_P}$; (1 R)
3. $y_Q \leftarrow -\lambda y_Q$;
4. $t \leftarrow x_P + x_Q$; (1 A)
5. $R \leftarrow \lambda(y_P t - y_Q\sigma - y_P\rho) \cdot (-t^2 + y_P y_Q \sigma - t\rho - \rho^2)$; (6 M, 1 C, 6 A)
6. $x_P[0] \leftarrow x_P$; $y_P[0] \leftarrow y_P$;
7. $x_Q[0] \leftarrow x_Q$; $y_Q[0] \leftarrow y_Q$;
8. **for** $j = 1$ to $\frac{m-1}{2}$ **do**
9. $\quad x_P[j] \leftarrow \sqrt[3]{x_P[j-1]}$; $x_Q[j] \leftarrow x_Q[j-1]^3$; (1 R, 1 C)
10. $\quad y_P[j] \leftarrow \sqrt[3]{y_P[j-1]}$; $y_Q[j] \leftarrow y_Q[j-1]^3$; (1 R, 1 C)
11. **end for**
12. **for** $j \leftarrow 1$ **to** $\frac{m-1}{4}$ **do**
13. $\quad t \leftarrow x_P[2j-1] + x_Q[2j-1]$; $\quad u \leftarrow y_P[2j-1]y_Q[2j-1]$; (1 M, 1 A)
14. $\quad t' \leftarrow x_P[2j] + x_Q[2j]$; $\quad u' \leftarrow y_P[2j]y_Q[2j]$; (1 M, 1 A)
15. $\quad S \leftarrow (-t^2 + u\sigma - t\rho - \rho^2) \cdot (-t'^2 + u'\sigma - t'\rho - \rho^2)$; (8 M, 13 A)
16. $\quad R \leftarrow R \cdot S$; (15 M, 67 A)
17. **end for**
18. **return** R^M;

Algorithm 5 Computation of $(-t^2 + u\sigma - t\rho - \rho^2) \cdot (-t'^2 + u'\sigma - t'\rho - \rho^2)$.

Input: t, u, t', and $u' \in \mathbb{F}_{3^m}$.
Output: $W = (-t^2 + u\sigma - t\rho - \rho^2) \cdot (-t'^2 + u'\sigma - t'\rho - \rho^2)$.

1. $a_1 \leftarrow t + u$; $a_2 \leftarrow t' + u'$; (2 A)
2. $a_3 \leftarrow t + t'$; $a_4 \leftarrow u + u'$; (2 A)
3. $m_1 \leftarrow t \cdot t'$; $m_2 \leftarrow u \cdot u'$; $m_3 \leftarrow a_1 \cdot a_2$; (3 M)
4. $w_3 \leftarrow m_1 + m_2 - m_3$; (2 A)
5. $m_4 \leftarrow m_1 \cdot m_1$; $m_5 \leftarrow m_1 \cdot a_3$; $m_6 \leftarrow m_1 \cdot a_4$; (3 M)
6. $m_7 \leftarrow a_3 \cdot a_3$; $m_8 \leftarrow a_3 \cdot w_3$; (2 M)
7. $w_0 \leftarrow m_4 - m_2 + ba_3$; (2 A)
8. $w_1 \leftarrow m_6 + m_8$; (1 A)
9. $w_2 \leftarrow m_5 + a_3 + b$; (2 A)
10. $w_4 \leftarrow m_7 - m_1 + 1$; (2 A)
11. $w_5 \leftarrow -a_4$;
12. **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2$;