Memory-saving computation of the pairing final exponentiation on BN curves[*]

Sylvain Duquesne and Loubna Ghammam

March 3, 2015

Abstract

In this paper, we describe and improve efficient methods for computing the hard part of the final exponentiation of pairings on Barreto-Naehrig curves.
Thanks to the variants of pairings which decrease the length of the Miller loop, the final exponentiation has become a significant component of the overall calculation. Here we exploit the structure of BN curves to improve this computation.
We will first present the most famous methods in the literature that ensure the computing of the hard part of the final exponentiation. We are particularly interested in the memory resources necessary for the implementation of these methods. Indeed, this is an important constraint in restricted environments.
More precisely, we are studying Devegili et al. method, Scott et al. addition chain method and Fuentes et al. method. After recalling these methods and their complexities, we determine the number of required registers to compute the final result, because this is not always given in the literature. Then, we will present new versions of these methods which require less memory resources (up to 37%). Moreover, some of these variants are providing algorithms which are also more efficient than the original ones.

Keywords: BN curves, Tate pairing, final exponentiation, memory resources, addition chain.

1 Introduction

The most significant complexity parameter of a pairing-friendly elliptic curve is the embedding degree k. It is defined as the smallest integer for which $r|p^k-1$, where r is the prime order of a large group of points of an elliptic curve E and p is the base field characteristic. The embedding degree changes from one curve to another and it is usually chosen in pairing based cryptography in the form $k=2^i3^j$ with $i\geq 1, j\geq 0$ [1]. In this paper we are interested in pairings on Barreto-Naehrig curves defined over $\mathbb{F}_p$ for which $k=12$.

[*]This work was supported in part by French ANR projects PEACE and SIMPATIC, LIRIMA MACISA project and centre Henri Lebesgue.

Tate pairing and its derivates have two steps. After computing the Miller loop, we must carry out an extra step to ensure a unique result for the pairing. This second step is called the final exponentiation, where the Miller loop result f_1 must be raised to the power $\frac{p^k-1}{r}$.
This final exponentiation can be broken down into three components.
Let $k' = k/2$, then

$$\frac{p^k-1}{r} = \left(p^{k'}-1\right)\left[\frac{(p^{k'}+1)}{\phi_k(p)}\right]\left[\frac{\phi_k(p)}{r}\right]$$

where $\phi_k(.)$ is the k-th cyclotomic polynomial. This decomposition is usually used for the calculation of the final exponentiation.
In our case $k = 12$, so the final exponent becomes

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

To compute the first part $f = f_1^{\left(p^6-1\right)\left(p^2+1\right)}$ we have to rise f_1 to the power p^6 and p^2 which are just easy Frobenius operations. We also have an inversion to perform. We are not interested here in this easy part of the computation of the final exponentiation. However it has an important consequence for rest of the computation. Indeed, powering f_1 to the p^6-1 makes the result unitary [2]. By this way, during the hard part of the final exponentiation (the computation of $f^{\frac{p^4-p^2+1}{r}}$) all the elements involved are unitary. This simplifies computations, for example any future inversion can be implemented as a Frobenius operator, more precisely $f^{-1} = f^{p^6}$ which is just a conjugation [2], [3]. So we assume in the following that inversions are free.
Many methods allow to compute the hard part of the final exponentiation but they are using many memory resources. This can be a significant drawback for restricted environment. For example a single temporary variable in this computation requires $\frac{256\times 12}{8} = 384$ bytes of (fast and easily available) memory at the 128 bits security level. The most famous methods are Devegili's method [4], addition chain method [5] and Fuentes method [6].

The paper is organized as follows. The section 2 is devoted to a presentation of BN Curves. The section 3 is a state of the art for computing the hard part of the final exponentiation, specially Devegilli, addition chain and Fuentes et al. methods.
Then we present our new methods in section 4 namely a new development of $\frac{p^4-p^2+1}{r}$, a new addition chain, a new development of Fuentes method and a variant of Fuentes based on a new multiple of the final exponentiation. These four new methods require less memory resources than the previous ones. In some cases we have a gain of complexity and in others the losses are negligible which makes these method very interesting for implementations in restricted environments.
Finally, in section 5, we compare the complexities and the number of temporary variables used between our algorithms and literature ones.

Notations and Assumptions:
In the rest of this paper we use the following notations.

- M_k is a multiplication in $\mathbb{F}_{p^k}$.
- S_k is a squaring in $\mathbb{F}_{p^k}$.
- F_k is a Frobenius map application in $\mathbb{F}_{p^k}$.
- I_k is an inversion in $\mathbb{F}_{p^k}$.
- w_e is the Hamming weight of an integer $\mid e \mid$.
- l_e is the length of $\mid e \mid$ in base 2.

For simplicity, we use M, S and I instead of M_1, S_1 and I_1.
Practically, when we compute the final exponentiation, we must perform the operations one by one in each line. For that, we assume that all operations of type $a \leftarrow ab$ are possible in place. This hypothesis is reasonable because our computations are in the field extensions. Anyway, our results would be similar if such operations were not possible.

2 Barreto and Naehrig Curves

Barreto and Naherig presented in [7] a method to generate pairing friendly elliptic curves over a prime field $\mathbb{F}_p$ with embedding degree $k = 12$ and a prime order n.
These curves are called BN curves and are defined over $\mathbb{F}_p$ by the following equation

$$E : \; y^2 = x^3 + b,$$

where $b \neq 0$ is nor a square neither a cube and by a parameter u such that

$$t = 6u^2 + 1$$
$$n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$$
$$p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$

where t is the trace of Frobenius map on the curve. The parameter u is chosen such that E has prime order. It has been proved that it is not restrictive to choose u sparse, for efficiency reasons. We assume this is the case in this paper, and more precisely in our examples we will choose a special value for u given in the following example.

Example 2.1 *Nogami et al. [8] have suggested the nice choice of*

$$u = -(4080000000000001)_{16}.$$

The Hamming weight of $-u$ is $w_u = 3$ and the length of $-u$ in base 2 is $l_u = 63$.

From the given expressions of p and r, the hard part of the final exponentiation can be written differently. After computing the fraction $\frac{p^4-p^2+1}{r}$ as a function of u [5], we obtain,

$$\frac{p^4 - p^2 + 1}{r} = \lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3$$

where

$$\begin{cases} \lambda_0 = -36u^3 - 30u^2 - 18u - 2 \\ \lambda_1 = -36u^3 - 18u^2 - 12u + 1 \\ \lambda_2 = 6u^2 + 1 \\ \lambda_3 = 1 \end{cases}$$

3 State of the Art

Let f_1 the result of Miller loop and $f = f_1^{(p^6-1)(p^2+1)}$ the result of the easy part of the final exponentiation. This section is devoted to the methods used in the literature to compute the hard part of the final exponentiation namely $f^{\frac{p^4-p^2+1}{r}}$. We take advantage of this state of the art to determine the memory resources needed for each method, because is not always given.

3.1 Naive method

A naive method is to compute directly $f^{\frac{p^4-p^2+1}{r}}$ using classical fast exponentiations algorithms [10], [9].

```
Algorithm 1: Square and Multiply algorithm
Input: $f$, $d = (d_{n-1}, d_{n-2}, ............d_1, d_0)_2$
Output: $f^d$
$t_0 \leftarrow f$
for $i = n-2$ down to $0$ do
    $t_0 \leftarrow t_0^2$
    If $d_i = 1$, then
        $t_0 \leftarrow t_0 f$
return $t_0$
```

The main advantage of this method in our context is that it uses only one temporary variable t_0. But we will see in the following that we can do much better in term of efficiency.
It is not hard to prove that the Hamming weight of $\lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3$, which is a function of u, is about $100w_u$ (see remark 3.1), and that its length is about 12 times the binary length of u.
So computing directly $f^{\frac{p^4-p^2+1}{r}}$ using algorithm 1 requires about $100w_u$ multiplications and $12l_u$ squarings in $\mathbb{F}_{p^{12}}$.

Remark 3.1 *To estimate the Hamming weight of $\frac{p^4-p^2+1}{r}$, we use the fact that if u is sparse then $w_{u^4} \simeq 4w_u$ and $w_{36u^4} \simeq 4w_u + 2$. An easy computation then yield to $w\left(\lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3\right) \simeq 100w_u$. Moreover the last expression is in degree 12 in u so that $l\left(\lambda_0 + \lambda_1 p + \lambda_2 p^2 + \lambda_3 p^3\right) \simeq 12l_u$.*

Example 3.2 *If we choose the value of u proposed by Nogami et al. as in example 2.1, we have to perform 759 squarings and 306 multiplications in $\mathbb{F}_{p^{12}}$ to compute $f^{\frac{p^4-p^2+1}{r}}$.*

We can do better than algorithm 1 by choosing advanced exponentiation methods such as sliding window. But we are not presenting them because they require more temporary variables for precomputations and because these are useless in case of sparse exponents.

Let us now present the most used methods in the literature.

3.2 Lucas Sequences Method

Lucas sequences is another method for implementing exponentiation in subgroups when we are working on extension fields of even degree. The exponentiation is done via a "laddering" algorithm [**?**, **?**] similar to the Montgomery ladder for elliptic curves [**?**]. As mentioned in [**?**], such an algorithm requires very little memory, which is our goal in this work. Lucas sequences were first used to compute the hard part of the final exponentiation in [2].

Let $m \in \mathbb{N}^*$, $p_1, p_2, \ldots, p_m \in \mathbb{R}$, a sequence $\{v_0, v_1, v_2, \ldots, v_{m-1}, \ldots\}$ is called a Lucas sequence of order m if and only if

$$\forall i \geq 0,\ v_{m+i} = p_1 v_{m+i-1} + p_2 v_{m+i-2} + \ldots p_m v_i$$

Example 3.3 *We give a simple example that we will use later. Let $M \in \mathbb{R}$, if $v_0 = 2,\ v_1 = M$ and $p_1 = M,\ p_2 = -1$, then*

$$v_{n+2} = M v_{n+1} - v_n$$

is a Lucas sequence of order 2.

A Lucas Sequence has the propriety

$$v_{n+m} = v_n v_m - v_{|n-m|}$$

From this property we deduce two proprieties which are useful in our context.

- $v_{2n} = v_n^2 - v_0$
- $v_{2n+1} = v_n v_{n+1} - v_1$

We have $f \in \mathbb{F}_{p^{12}}$, let us write $f = a + zb$ where $a, b \in \mathbb{F}_{p^6}$ and $F_{p^{12}} = F_{p^6}[z]$. To compute f^n, we use the fact that

$$f^n = (a + bz)^n = \frac{v_n}{2} + zbu_n$$

where

- v_n is a Lucas Sequence as the one given in example 3.3.
- $u_n = \frac{2v_{n+1} - Mv_n}{M^2 - 4}$.

The main interest of this method is that we are performing $\mathbb{F}_{p^6}$ operations instead of $\mathbb{F}_{p^{12}}$ ones. Algorithm 2 ensures the computation of the hard part of the final exponentiation using Lucas sequences.

Algorithm 2: Lucas Sequence

```
Input: $f = a + bz,\ d = (d_{n-1}, d_{n-2}, \dots d_1, d_0)_2$
Output: $f^d$
Temp. var.: $t_0, M, v_0, v_1$
1: $M \leftarrow 2a$
   $v_0 \leftarrow 2$
   $v_1 \leftarrow M$
2: for $i = n - 2$ down to $0$ do
     if $d_i = 0$ then
3:       $v_1 \leftarrow v_0 v_1 - M$
         $v_0 \leftarrow v_0^2 - 2$
     else
4:       $v_0 \leftarrow v_0 v_1 - M$
         $v_1 \leftarrow v_1^2 - 2$
   end for
5: $v_0 \leftarrow v_0 v_1 - M$
   $v_1 \leftarrow v_1^2 - 2$
# $v_0$ and $v_1$ are carrying $v_{n+1}$ and $v_n$. Let us now compute $u_n$.
6: $t_0 \leftarrow M v_0$
   $v_1 \leftarrow 2v_1 - t_0$
   $M \leftarrow M^2 - 4$
   $v_1 \leftarrow v_1 / M$
   $v_1 \leftarrow b v_1$
7: $v_0 \leftarrow v_0 / 2$
8: return $v_0 + v_1 z$
```

This algorithm requires four temporary variables in $\mathbb{F}_{p^6}$. Concerning its cost, we have at steps 3 and 4 about $l_d = 12 l_u$ multiplications and $12 l_u$ squarings in $\mathbb{F}_{p^6}$. At step 5 we have an extra multiplication and an extra squaring. The computation of u_n at step 6 requires 3 multiplications, an inversion and a squaring in $\mathbb{F}_{p^6}$. So, the total cost of this algorithm is around $12 l_u + 4$ multiplications, $12 l_u + 2$ squarings and an inversion in $\mathbb{F}_{p^6}$. Note that this method does not take any advantage of the sparsity of u.

Example 3.4 *With the choice of u given in example 2.1, the cost of Lucas Sequence method is 763 M_6, 761 S_6 and I_6. Moreover, we need* 4 *temporary variables in $\mathbb{F}_{p^6}$ to compute $f^{\frac{p^4-p^2+1}{r}}$.*

3.3 Devegili & al. method

In 2000 Devegili and al. proposed a very efficient method for the computation of $f^{\frac{p^4-p^2+1}{r}}$ [4]. They developed the exponent $\frac{p^4-p^2+1}{r}$ in a tricky way in order to reuse some exponents ($6u^2 + 1$ and $6u + 5$).

$$\begin{aligned}\frac{p^4-p^2+1}{r} &= \lambda_3 p^3+\lambda_2 p^2+\lambda_1 p+\lambda_0\\ &= p^3+(6u^2+1)p^2+(-36u^3-18u^2-12u+1)p\\ &\quad +(-36u^3-30u^2-18u-2)\\ &= p^3+p^2(6u^2+1)+p\left((-6u-5)(6u^2+1)+2(6u^2+1)\right)\\ &\quad +p\left((-6u-5)+9\right)+(-6u-5)(6u^2+1)\\ &\quad +(-6u-5)+(-6u-5)+9+4\end{aligned}$$

So they finally compute $f^{\frac{p^4-p^2+1}{r}}$ as

$$f^{\frac{p^4-p^2+1}{r}}=f^{p^3}\left(f^{p^2}\left(f^pf\right)^{-6u-5}\left(f^p\right)^2\right)^{6u^2+1}\left(f^pf\right)^{-6u-5}f^{-6u-5}\left(f^pf\right)^9f^4$$

Devegili and al. presented algorithm 3 to compute this expression.

```
Algorithm 3: Devegili et al. [4]
Input: $f$, $u$
Output: $f^{\frac{p^4-p^2+1}{r}}$
1: $a \leftarrow f^{-6u-5}$
2: $b \leftarrow a^p$
3: $b \leftarrow ab$
4: Compute $f^p, f^{p^2}$, and $f^{p^3}$
5: $f \leftarrow f^{p^3}[f^{p^2}b\,(f^p)^2]^{6u^2+1}b\,(f^pf)^9\,af^4$
```

Note that the exponentiations to the power of p (steps 2 and 4) are efficiently computed using Frobenius.
In step 1, we use algorithm 1 to rise f to the power $-6u-5$ which is a (l_u+2)-bit integer of Hamming weight w_u+2 (u is chosen sparse). So we need w_u+1 multiplications and l_u+1 squarings in $\mathbb{F}_{p^{12}}$ to compute a. In the same way $l_{6u^2+1}=2l_u+1$ and $w_{6u^2+1}\simeq 2w_u+7$, so for computing the exponentiation to the power $6u^2+1$, we need $2l_u$ squarings and $2w_u+6$ multiplications.
The cost of step 2 is F_{12} and the cost of step 3 is M_{12}. For step 5 we need $3F_{12}$ to compute f^p, f^{p^2} and f^{p^3}, $2M_{12}+S_{12}$ to compute $f^{p^2}b(f^p)^2$, $3S_{12}+2M_{12}$ for the 9-th power and $2S_{12}$ to compute f^4.
Finally, 5 extra M_{12} are needed to multiply the terms together. So the total cost of this algorithm is approximately $(3l_u+7)S_{12}$, $(3w_u+17)M_{12}$ and $4F_{12}$.

The classical work of Devigili et al. was presented in algorithm 3, but they do not take into account temporary variables used in their algorithm which can cause a problem in restricted environments. So let us give a more detailed version of algorithm 3 which take into account the temporary variables.

Algorithm 4: detailed Devegili et al.	Term computed and comments
Input: f, u	
Temp. var.: t_0, t_1, t_2, t_3	
Output: $f^{\frac{p^4-p^2+1}{r}}$	
$t_0 \leftarrow f^{-6u-5}$	# using algorithm 1
$t_1 \leftarrow t_0^p$	
$t_1 \leftarrow t_0 t_1$	$(f^p f)^{-6u-5}$
$t_0 \leftarrow t_0 t_1$	$(f^p f)^{-6u-5} f^{-6u-5}$
$t_2 \leftarrow f^p$	
$t_3 \leftarrow t_2 f$	
$t_3 \leftarrow {t_3}^9$	# using algorithm 1
$t_0 \leftarrow t_0 t_3$	
$t_3 \leftarrow f^4$	
$t_0 \leftarrow t_0 t_3$	
$t_2 \leftarrow t_2^2$	
$t_2 \leftarrow t_2 t_1$	
$t_1 \leftarrow f^{p^2}$	
$t_1 \leftarrow t_1 t_2$	$f^{p^2} (f^p f)^{-6u-5} (f^p)^2$
$t_2 \leftarrow t_1^{6u^2+1}$	# using algorithm 1
$t_0 \leftarrow t_2 t_0$	
$t_1 \leftarrow f^{p^3}$	
$t_1 \leftarrow t_1 t_0$	$f^{\frac{p^4-p^2+1}{r}}$
return t_1	

For this algorithm we need 4 temporary variables in $\mathbb{F}_{p^{12}}$ to compute $f^{\frac{p^4-p^2+1}{r}}$.

Example 3.5 *If u is chosen as in example 2.1, $-6u-5$ is a $65-$bit integer of Hamming weight 5 and $6u^2+1$ is a 127-bit integer of Hamming weight 13. The first step costs $64S_{12}+4M_{12}$ and we need $126S_{12}+12M_{12}$ to compute the $6u^2+1$-th powering. So the total cost of the hard part of the final exponentiation, using Devegili et al. method, is $196S_{12}+26M_{12}+4F_{12}$.*

3.4 Addition chain

In 2009 Scott et al. presented a new approach based on addition chain for computing the hard part of the final exponentiation [5]. For this they write $f^{\frac{p^4-p^2+1}{r}}$ as groups of terms having the same exponents.

$$\begin{aligned}
f^{\frac{p^4-p^2+1}{r}} &= f^{\lambda_3 p^3} f^{\lambda_2 p^2} f^{\lambda_1 p} f^{\lambda_0} \\
&= f^{p^3} f^{\left(6u^2+1\right)p^2} f^{\left(-36u^3-18u^2-12u+1\right)p} f^{\left(-36u^3-30u^2-18u-2\right)} \\
&= f^{p^3} f^{p^2} \left(f^{p^2}\right)^{6u^2} (f^p)^{-36u^3} (f^p)^{-18u^2} (f^p)^{-12u} f^p f^{-36u} f^{-30u^2} f^{-18u} f^{-2} \\
&= \left[f^p f^{p^2} f^{p^3}\right] [1/f]^2 \left[\left(f^{u^2}\right)^{p^2}\right]^6 [1/(f^u)^p]^{12} \left[1/f^u \left(\left(f^{u^2}\right)^p\right)\right]^{18} \left[1/\left(f^{u^2}\right)\right]^{30} \\
&\quad \left[1/f^{u^3}\left(f^{u^3}\right)^p\right]^{36}.
\end{aligned}$$

So

$$f^{\frac{p^4-p^2+1}{r}} = y_0 y_1^2 y_2^6 y_3^{12} y_4^{18} y_5^{30} y_6^{36} \tag{1}$$

where

$$\begin{cases} y_0 = f^p f^{p^2} f^{p^3} \\ y_1 = 1/f \\ y_2 = \left(f^{u^2}\right)^{p^2} \\ y_3 = (f^u)^p \\ y_4 = 1/f^u \left(\left(f^{u^2}\right)^p\right) \\ y_5 = 1/\left(f^{u^2}\right) \\ y_6 = 1/f^{u^3}\left(f^{u^3}\right)^p \end{cases}$$

Olivos algorithm allows to evaluate any expressions of this form, with a minimum of multiplications ([11] and [12] chapter 9.2).
The first point is to find an addition chain which includes the exponents appearing in (1). In this case, an optimal addition chain is given by

$$\{1,\ 2,\ \overline{3},\ 6,\ 12,\ 18,\ 30,\ 36\}$$

Note that 3 appears in the addition chain but is not an exponent in (1). As mentioned by Scott, this is certainly an advantage, because it means that we have less work to do in the evaluation of (1). Expression (1) can be then computed thanks to the algorithm 5.

```
Algorithm 5: Addition Chain [11], [5]
Input: $f$, $u$
Temp. Var: $t_0, t_1$
Output: $f^{\frac{p^4-p^2+1}{r}}$
$t_0 \leftarrow y_6^2$
$t_0 \leftarrow t_0 y_4$
$t_0 \leftarrow t_0 y_5$
$t_1 \leftarrow y_3 y_5$
$t_1 \leftarrow t_0 t_1$
$t_0 \leftarrow t_0 y_2$
$t_1 \leftarrow t_1^2$
$t_1 \leftarrow t_1 t_0$
$t_1 \leftarrow t_1^2$
$t_0 \leftarrow t_1 y_1$
$t_1 \leftarrow t_1 y_0$
$t_0 \leftarrow t_0^2$
$t_0 \leftarrow t_0 t_1$
return $t_0$
```

For this algorithm, Scott et al. used only 2 temporary variables t_0 and t_1, 9 multiplications and 4 squarings in $\mathbb{F}_{p^{12}}$.
However they first have to compute $y_0, y_1, y_2, y_3, y_4, y_5$ and y_6. For this, they need 7 extra temporary variables and t_0. So, 9 temporary variables are necessary to compute the hard part of the final exponentiation using this method.

The cost of computing $y_0, y_1, y_2, y_3, y_4, y_5$ and y_6 is $3(w_u - 1)$ multiplications and $3(l_u - 1)$ squarings for computing f^{-u}, f^{u^2} and f^{-u^3}, 7 Frobenius applications and finally 4 multiplications to multiply terms together. Remember that, as noticed in the introduction, inversions are free because f is unitary.
So, the total cost of the hard part of the final exponentiation using addition chain method is $7F_{12}$, $(3w_u + 10)M_{12}$ and $(3l_u + 1)S_{12}$.

Example 3.6 *If u is chosen as in example 2.1, the cost of algorithm 5 is $190S_{12} + 19M_{12} + 7F_{12}$. Scott et al. used 9 temporary variables to compute this result.*

Remark 3.7 *Our calculation of the complexities of the two last methods allows us to recover the result obtained by practical implementations in the literature namely Scott et al. method is 5 percent faster than Devegili et al. method to compute the final exponentiation.*
However, we can see than it is much more memory consuming. At the 128 bits security level the 9 temporary variables used in Scott method are representing 3.4 *KB which can widely exceed the capacities of a device.*

3.5 Fuentes et al. method

More recently, Fuentes et al. presented a new way for computing the hard part of the final exponentiation [6].
The idea of their method is to use a multiple d' of $d = \frac{p^4 - p^2 + 1}{r}$, (with r not dividing d'). Then they compute $f^{d'}$ instead of computing f^d using the fact that a fixed power of a pairing is also a pairing.
They presented a lattice-based method for determining d' such that $f^{d'}$ can be computed at least as efficiently as f^d. Thanks to LLL algorithm [13], the best vector d' that they found is given by

$$d'(u) = \alpha_0 + \alpha_1 p + \alpha_2 p^2 + \alpha_3 p^3 = sd(u)$$

where

$$\begin{cases} s = 2u\left(6u^2 + 3u + 1\right) \\ \alpha_0 = 1 + 6u + 12u^2 + 12u^3 \\ \alpha_1 = 4u + 6u^2 + 12u^3 \\ \alpha_2 = 6u + 6u^2 + 12u^3 \\ \alpha_3 = -1 + 4u + 6u^2 + 12u^3 \end{cases}$$

Then, they use Devegili method so they developed the power d' as follow

$$\begin{aligned} f^{d'} &= f^{\alpha_0} f^{\alpha_1 p} f^{\alpha_2 p^2} f^{\alpha_3 p^3} \\ &= \left(af^{6u^2} f\right) (b)^p (a)^{p^2} \left(bf^{-1}\right)^{p^3} \end{aligned}$$

where $a = f^{6u} f^{6u^2} f^{12u^3}$ and $b = af^{-2u}$.
As they do not give a detailed algorithm, we will present algorithm 6 for computing $f^{d'}$ taking into account temporary variables.

Algorithm 6: Fuentes et al. [6]	Term computed and comments	Cost
Input: f, u		
Output: $f^{s\frac{p^4-p^2+1}{r}}$		
Temp. var.: t_0, t_1, t_2, t_3, t_4		
$t_0 \leftarrow f^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_0 \leftarrow t_0^2$		S_{12}
$t_1 \leftarrow t_0^2$		S_{12}
$t_1 \leftarrow t_0t_1$		M_{12}
$t_2 \leftarrow t_1^{-u}$	f^{6u^2}	$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_3 \leftarrow t_1^{-1}$	f^{6u}	
$t_1 \leftarrow t_2t_3$		M_{12}
$t_3 \leftarrow t_2^2$		S_{12}
$t_4 \leftarrow t_3^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_4 \leftarrow t_4^{-1}$		
$t_4 \leftarrow t_4t_1$	$f^{6u}f^{6u^2}f^{12u^3}=f^{\alpha_2}$	M_{12}
$t_3 \leftarrow t_4t_0$	$f^{4u}f^{6u^2}f^{12u^3}=f^{\alpha_1}$	M_{12}
$t_0 \leftarrow t_2t_4$		M_{12}
$t_0 \leftarrow t_0f$	$af^{6u^2}f=f^{\alpha_0}$	M_{12}
$t_2 \leftarrow t_3^p$		F_{12}
$t_0 \leftarrow t_2t_0$		M_{12}
$t_2 \leftarrow t_4^{p^2}$		F_{12}
$t_0 \leftarrow t_2t_0$		M_{12}
$t_2 \leftarrow f^{-1}$		
$t_2 \leftarrow t_2t_3$	f^{α_3}	M_{12}
$t_2 \leftarrow t_2^{p^3}$		F_{12}
$t_0 \leftarrow t_2t_0$	$f^{s\frac{p^4-p^2+1}{r}}$	M_{12}
return t_0		

For this algorithm we used 5 temporary variables in $\mathbb{F}_{p^{12}}$. The cost of this algorithm is given step by step in algorithm 6.
The overall cost of computing $f^{d'}$ is then $3l_uS_{12}+(3w_u+7)M_{12}+3F_{12}$.

Example 3.8 *With the value of u chosen in example 2.1, the total cost of this algorithm is* $189S_{12}+16M_{12}+3F_{12}$.

Remark 3.9 *Using this method we compute a power of a pairing. This is not a problem because a fixed power of a pairing is also a pairing. But it can be a disadvantage if we implement a standard pairing such as Optimal Ate pairing [14]. In case of interoperability and compatibility requirements, this method could be avoided.*

4 Variants of previous methods

In this section, we will present our contribution to the computation of the hard part of the final exponentiation. Our aim is to decrease the number of temporary variables in $\mathbb{F}_{p^{12}}$ required for this operation to make it friendly with restricted environments. We will present four new variants of the state of the art methods.

More precisely we first write a new development of $\frac{p^4-p^2+1}{r}$ to obtain a variant of Devegili's method which is not only less memory consuming but also more efficient. Then we give a new addition chain providing a variant of Scott's method requiring much less temporary variables. Finally we also write a new way to compute $f^{d'}$ in Fuentes method and a new exponent d_1 allowing to decrease the number of temporary variables required for this method.

4.1 New Development of $f^{\frac{p^4-p^2+1}{r}}$

In this paragraph, we present a new way to develop $\frac{p^4-p^2+1}{r}$. This development is chosen in order to bring out repeated expression in λ_0, λ_1, λ_2 and λ_3. So that we will compute them just once. The best we can do is with the exponents $6u^2+1$, $-6u-1$, and $-6u-5$.

$$\begin{aligned}
\lambda_0 &= -36u^3-30u^2-18u-2\\
&= (-6u-5)(6u^2+1)+2(-6u-1)+5)\\
\lambda_1 &= -36u^3-30u^2-12u+1\\
&= (-6u-5)(6u^2+1)+(-6u-1)+(12u^2+7)\\
\lambda_2 &= 6u^2+1\\
\lambda_3 &= 1.
\end{aligned}$$

So,

$$\begin{aligned}
f^{\frac{p^4-p^2+1}{r}} &= f^{\lambda_3}f^{\lambda_2}f^{\lambda_1}f^{\lambda_0}\\
&= f^{p^3}\left(f^{6u^2+1}\right)^{p^2}\left(f^{-6u-1}\right)^{p}\left(\left(f^{6u^2+1}\right)^{-6u-5}\right)^{p}\left(f^{7}\right)^{p}\left(f^{12u^2}\right)^{p}\\
&\quad\times\left(f^{6u^2+1}\right)^{-6u-5}\left(f^{-6u-1}\right)^{2}f^{5}
\end{aligned}$$

Practically, to compute the hard part of the final exponentiation, we use the algorithm 7 based on this new development of $\frac{p^4-p^2+1}{r}$.
In this algorithm, we compute all the terms one by one and we accumulate their product in the temporary variable t_1.

Algorithm 7: new variant of Devegili et al.	Term computed and comments	Cost
Input: f, u		
Output: $f^{\frac{p^4-p^2+1}{r}}$		
Temp. var.: t_0, t_1, t_2		
$t_0 \leftarrow f^{-2u}$		$l_u S_{12} + (w_u - 1)M_{12}$
$t_1 \leftarrow t_0^2$		S_{12}
$t_0 \leftarrow t_0 t_1$		M_{12}
$t_1 \leftarrow f^{-1}$		
$t_1 \leftarrow t_0 t_1$		M_{12}
$t_2 \leftarrow t_1^p$	$\left(f^{-6u-1}\right)^p$	F_{12}
$t_1 \leftarrow t_1^2$	$\left(f^{-6u-1}\right)^2$	S_{12}
$t_1 \leftarrow t_1 t_2$	# accumulate	M_{12}
$t_2 \leftarrow (t_0)^{-u}$		$(l_u - 1)S_{12} + (w_u - 1)M_{12}$
$t_0 \leftarrow t_2^2$		S_{12}
$t_0 \leftarrow t_0^p$	$\left(f^{12u^2}\right)^p$	F_{12}
$t_1 \leftarrow t_1 t_0$	# accumulate	M_{12}
$t_2 \leftarrow t_2 f$		M_{12}
$t_0 \leftarrow (t_2)^{p^2}$	$\left(f^{6u^2+1}\right)^{p^2}$	F_{12}
$t_1 \leftarrow t_0 t_1$	# accumulate	M_{12}
$t_0 \leftarrow (t_2)^{-6u-5}$	$\left(f^{6u^2+1}\right)^{-6u-5}$	$(l_u + 1)S_{12} + (w_u + 1)M_{12}$
$t_1 \leftarrow t_1 t_0$	# accumulate	M_{12}
$t_0 \leftarrow t_0^p$		F_{12}
$t_1 \leftarrow t_1 t_0$	# accumulate	M_{12}
$t_0 \leftarrow f^2$		S_{12}
$t_2 \leftarrow t_0^2$		S_{12}
$t_2 \leftarrow f t_2$	f^5	M_{12}
$t_1 \leftarrow t_2 t_1$	# accumulate	M_{12}
$t_2 \leftarrow t_2 t_0$		M_{12}
$t_2 \leftarrow t_2^p$	$\left(f^7\right)^p$	F_{12}
$t_1 \leftarrow t_2 t_1$	# accumulate	M_{12}
$t_2 \leftarrow f^{p^3}$		F_{12}
$t_1 \leftarrow t_2 t_1$	$f^{\frac{p^4-p^2+1}{r}}$	M_{12}
return t_1		

For this algorithm we use only 3 temporary variables in $\mathbb{F}_{p^{12}}$.
To compute any exponentiation, as mentioned in section 2, we use algorithm 1. As $-6u - 5$ is a $l_u + 2$-bit integer of Hamming weight $w_u + 2$ (assuming u is sparse), we need $w_u + 1$ multiplications and $l_u + 1$ squarings in $\mathbb{F}_{p^{12}}$ to compute t_0^{-6u-5}. The cost of algorithm 7 is given step by step inside the algorithm. The overall cost to compute $f^{\frac{p^4-p^2+1}{r}}$ using our new development of $\frac{p^4-p^2+1}{r}$ is $(3l_u + 5)S_{12} + (3w_u + 12)M_{12} + 6F_{12}$.

Example 4.1 *With the value of* u *chosen in example 2.1, the total cost of this algorithm is* $194S_{12}$, $21M_{12}$ *and* $6F_{12}$.

This new development is a variant of Devegili et al. method, so we compare them in table 1.

Method	Complexity			Temp. var.
	S_{12}	M_{12}	F_{12}	
Devegili et al. (algorithm 4)	$3l_u+7$	$3w_u+17$	4	4
New development (algorithm 7)	$3l_u+5$	$3w_u+12$	6	3

Table 1: Comparison between Devegili and our new development

Example 4.2 *To complete the comparison, we choose the particular value of u given in example 2.1. We get the table 2.*

The Method	Complexity	Temp. var.
Devegili et al. method	$196\ S_{12}+\ 26M_{12}+4F_{12}$	4
New Development	$194\ S_{12}+\ 21M_{12}+6F_{12}$	3

Table 2: Example of Table 1.

If we compare the complexity of our algorithm with Devegili et al. one [4], we can easily remark that we save 5 multiplications and 2 squarings. We have two extra Frobenius map applications but this is not a problem because the cost of 2 Frobenius is less than the cost of one multiplication. We also remark that our new algorithm use only 3 temporary variables in $\mathbb{F}_{p^{12}}$ instead of 4 which was our initial goal.

4.2 New Addition Chain

In this section, we are interested in improving the addition chain method presented by Scott et al. especially in term of memory usage. The idea of this variant is to present $f^{\frac{p^4-p^2+1}{r}}$ as a product of terms whose exponents are smaller than the exponents appearing in the expression presented by Scott et al. in [5]. This allows to use less temporary variables for computing the hard part of the final exponentiation.

$$\begin{aligned}
f^{\frac{p^4-p^2+1}{r}} &= f^{\lambda_3 p^3} f^{\lambda_2 p^2} f^{\lambda_1 p} f^{\lambda_0} \\
&= f^{p^3} f^{\left(6u^2+1\right)p^2} f^{\left(-36u^3-18u^2-12u+1\right)p} f^{-36u^3-30u^2-18u-2} \\
&= f^{p^3} f^{p^2} \left(\left(f^{u^2}\right)^{p^2}\right)^6 \left(f^p\right)^{-36u^3} \left(f^p\right)^{-18u^2} \left(f^p\right)^{-12u} f^p f^{-36u^3} f^{-30u^2} f^{-18u} f^{-2} \\
&= \left[f^p f^{p^2} f^{p^3}\right] \left[\left(f^{3u^2}\right)^{p^2} f^{-1} f^{-6u^2}\right]^2 \left[\left(f^p f\right)^{-4u} f^{-2u}\right]^3 \left[\left(f^p f\right)^{-6u^3} \left(f^p f\right)^{-3u^2}\right]^6 \quad (2)
\end{aligned}$$

Using the following precomputations

$$\begin{aligned}
y_0 &= f^p f^{p^2} f^{p^3}, \\
y_1 &= \left(f^{3u^2}\right)^{p^2} f^{-1} f^{-6u^2}, \\
y_2 &= \left(f^p f\right)^{-4u} f^{-2u}, \\
y_3 &= \left(f^p f\right)^{-6u^3} \left(f^p f\right)^{-3u^2},
\end{aligned}$$

expression (2) becomes

$$f^{\frac{p^4-p^2+1}{r}} = y_0 y_1^2 y_2^3 y_3^6. \tag{3}$$

To compute y_0, y_1, y_2 and y_3 we propose algorithm 8.

Algorithm 8: precomputations for new addition chain	Term computed	Cost
Input: $f,\ u$		
Temp. var.: y_0, y_1, y_2, y_3, t_0		
Output: y_0, y_1, y_2, y_3		
$t_0 \leftarrow f^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$y_3 \leftarrow t_0^2$		S_{12}
$y_0 \leftarrow t_0 y_3$	f^{-3u}	M_{12}
$y_2 \leftarrow y_3^p$		F_{12}
$y_2 \leftarrow y_3 y_2$		M_{12}
$y_2 \leftarrow y_2^2$		S_{12}
$y_2 \leftarrow y_3 y_2$	y_2	M_{12}
$t_0 \leftarrow y_0^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$y_0 \leftarrow f^{-1}$		
$y_1 \leftarrow t_0^{p^2}$		F_{12}
$y_1 \leftarrow y_0 y_1$	$(f^{3u^2})^{p^2} f^{-1}$	M_{12}
$t_0 \leftarrow t_0^{-1}$		
$y_3 \leftarrow t_0^p$		F_{12}
$y_3 \leftarrow t_0 y_3$	$(f^p f)^{-3u^2}$	M_{12}
$t_0 \leftarrow t_0^2$		S_{12}
$y_1 \leftarrow t_0 y_1$	y_1	M_{12}
$t_0 \leftarrow y_3^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_0 \leftarrow t_0^2$		S_{12}
$t_0 \leftarrow t_0^{-1}$		
$y_3 \leftarrow t_0 y_3$	y_3	M_{12}
$t_0 \leftarrow f^p$		F_{12}
$y_0 \leftarrow f^{p^2}$		F_{12}
$y_0 \leftarrow t_0 y_0$		M_{12}
$t_0 \leftarrow f^{p^3}$		F_{12}
$y_0 \leftarrow t_0 y_0$	y_0	M_{12}
return y_0, y_1, y_2, y_3		

For this algorithm we use only 5 temporary variables in $\mathbb{F}_{p^{12}}$. The cost of these precomputations is given step by step inside algorithm 8 and the overall cost is $(3l_u+1)S_{12}+(3w_u+6)M_{12}+6F_{12}$.

Thanks to Olivos algorithm, we can now easily compute the expression (3). The starting point is to find an addition chain where these exponents appear. In our case, it is not hard to see that an optimal addition chain is given by

$$\{1,\ 2,\ 3,\ 6\}$$

Now we carry out Olivos algorithm and we obtain, as a result, the following addition vectors,

$$\begin{array}{c}(1,\ 0,\ 0,\ 0)\\(0,\ 1,\ 0,\ 0)\\(0,\ 0,\ 1,\ 0)\\(0,\ 0,\ 0,\ 1)\\(2,\ 0,\ 0,\ 0)\\(2,\ 1,\ 0,\ 0)\\(2,\ 1,\ 0,\ 1)\\(2,\ 1,\ 1,\ 0)\\(4,\ 2,\ 2,\ 0)\\(6,\ 3,\ 2,\ 1)\end{array}$$

This allows to evaluate the expression (3) thanks to algorithm 9, without using any new temporary variable.

Algorithm 9: New addition chain

```
Input: $f, y_0, y_1, y_2, y_3$
Output: $f^{\frac{p^4-p^2+1}{r}}$
Temp. var.: $t_0$
$t_0 \leftarrow y_3^2$
$t_0 \leftarrow t_0 y_2$
$y_3 \leftarrow t_0 y_0$
$t_0 \leftarrow t_0 y_1$
$t_0 \leftarrow t_0^2$
$t_0 \leftarrow t_0 y_3$
return $t_0$
```

For this algorithm we have to perform 4 multiplications and 2 squarings to compute the final result in $\mathbb{F}_{p^{12}}$.
So the total cost of the hard part of the final exponentiation (which is the combined cost of algorithms 8 and 9) using this new addition chain is 6 Frobenius, $3w_u + 10$ multiplications and $3l_u + 3$ squarings in $\mathbb{F}_{p^{12}}$.

Example 4.3 *With the value of u chosen in example 2.1, the cost of the new addition chain method is* $19M_{12} + 192S_{12} + 6F_{12}$. *However we use only 5 temporary variables.*

In table 3, we compare Scott et al addition chain and our new addition chain.

Method	Complexity			Temp. var.
	S_{12}	M_{12}	F_{12}	
Scott addition chain (algorithm 5)	$3l_u + 1$	$3w_u + 10$	7	9
New addition chain (algorithm 9)	$3l_u + 3$	$3w_u + 10$	6	5

Table 3: Comparison between addition chains

Example 4.4 *To have a full comparison, we choose the particular value of u given in example 2.1. We get the table 4.*

Method	Complexity	Temp. var.
Scott addition Chain	$190\ S_{12}+\ 19M_{12}+7F_{12}$	9
New addition Chain	$192\ S_{12}+\ 19M_{12}+6F_{12}$	5

Table 4: Example of Table 3

In these tables we can easily remark that our new addition chain requires 2 extra squarings but we save one Frobenius. This means that our algorithm is slightly less efficient than Scott's one. However we save 4 temporary variables in $\mathbb{F}_{p^{12}}$ which is an important improvement for implementations in restricted environments.

4.3 Variant of Fuentes Method

In this paragraph we present a new way of developing the exponent d' presented in Fuentes et al. paper [6]. The main idea of this development is to make $6u^2+1$ appear in α_2 and α_0, and then to write α_0, α_1 and α_3 in terms of α_2.

$$\begin{aligned}
\alpha_2 &= 12u^3+6u^2+6u \\
&= \left(6u^2+1\right)(2u+1)+4u-1 \\
\alpha_1 &= \alpha_2-2u \\
\alpha_0 &= \alpha_2+6u^2+1 \\
\alpha_3 &= \alpha_1-1.
\end{aligned}$$

To evaluate $f^{d'}$ we apply algorithm 10.

Algorithm 10: new variant of Fuentues et al.	Term Computed and comments	Cost
Input: f, u		
Output: $f^{s\frac{p^4-p^2+1}{r}}$		
Temp. var.: t_0, t_1, t_2, t_3		
$t_0 \leftarrow f^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_0 \leftarrow t_0^2$		S_{12}
$t_2 \leftarrow t_0^{-u}$	f^{2u^2}	$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_1 \leftarrow t_2^2$		S_{12}
$t_2 \leftarrow t_2 t_1$	f^{6u^2}	M_{12}
$t_2 \leftarrow t_2 f$	f^{6u^2+1}	M_{12}
$t_1 \leftarrow t_2^{-2u-1}$	$\left(f^{6u^2+1}\right)^{-2u-1}$	$l_u S_{12}+(w_u-1)M_{12}$
$t_3 \leftarrow t_1^{-1}$		
$t_1 \leftarrow t_0^2$	f^{-4u}	
$t_1 \leftarrow t_1 f$		M_{12}
$t_1 \leftarrow t_1^{-1}$	f^{4u-1}	
$t_1 \leftarrow t_1 t_3$	f^{α_2}	M_{12}
$t_0 \leftarrow t_0 t_1$	$f^{\alpha_1}=f^{\alpha_2-2u}$	M_{12}
$t_2 \leftarrow t_2 t_1$	$f^{\alpha_0}=f^{\alpha_2+6u^2+1}$	M_{12}
$t_3 \leftarrow t_1^{p^2}$		F_{12}
$t_2 \leftarrow t_2 t_3$	$(f^{\alpha_2})^{p^2} f^{\alpha_0}$	M_{12}
$t_3 \leftarrow f^{-1}$		
$t_3 \leftarrow t_0 t_3$	$f^{\alpha_3}=f^{\alpha_1-1}$	M_{12}
$t_1 \leftarrow t_3^{p^3}$	$(f^{\alpha_3})^{p^3}$	F_{12}
$t_2 \leftarrow t_2 t_1$		M_{12}
$t_1 \leftarrow t_0^p$	$(f^{\alpha_1})^p$	F_{12}
$t_1 \leftarrow t_2 t_1$	$f^{d'}$	M_{12}
return t_1		

For this algorithm we used 4 temporary variables in $\mathbb{F}_{p^{12}}$.
As $-2u-1$ is a l_u+1-bit integer of Hamming weight w_u (assuming u is sparse), we need w_u-1 multiplications and l_u squarings in $\mathbb{F}_{p^{12}}$ to compute t_2^{-2u-1}. The cost to compute $f^{d'}$ is given step by step inside algorithm 10. The overall cost of this algorithm is then $(3w_u+7)M_{12}$, $3l_uS_{12}$ and $3F_{12}$.

Example 4.5 *With the value of u chosen in example 2.1, the cost of the new development of Fuentes method is $16M_{12}+189S_{12}+3F_{12}$. We use only 4 temporary variables to compute a multiple of the hard part of the final exponentiation.*

4.4 New Multiple of d

As described in Fuentes et al. paper, we can find many multiples d' of the exponent d involved in the hard part of the final exponentiation.
In order to minimize the number of operations, Fuentes et al. imposed the condition that the largest coefficient of d' is 12. In our case we relax this constraint but we impose a constant vector among the α_i to use less temporary variables.

With this constraint, a brute force search of linear combinations of the LLL basis [13] provides 4 non-zeros vectors. Among these vectors, we consider the following one,

$$(0,\ 6,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 36,\ 24,\ 18,\ 1,\ 36,\ 18,\ 12,\ -2)$$

which corresponds to the multiple

$$d_1 = \alpha_0 + \alpha_1 p + \alpha_2 p^2 + \alpha_3 p^3 = s_1 d$$

where

$$\begin{cases} s_1 = 36u^3 + 18u^2 + 6u + 1 \\ \alpha_0 = 6u^2 + 1 \\ \alpha_1 = 1 \\ \alpha_2 = 36u^3 + 24u^2 + 18u + 1 \\ \alpha_3 = 36u^3 + 18u^2 + 12u - 2 \end{cases}$$

So that,

$$f^{d_1} = f^{\alpha_0} f^{\alpha_0 p} f^{\alpha_0 p^2} f^{\alpha_0 p^3}$$

Let us now write the α_i in a simpler way. For this, we use the same technique as in section 4.1 to compute f^{d_1} efficiently. The following development is chosen because the three exponents $6u^2+1$, $6u-1$ and $6u+4$ are used several times but we will compute them just once.

$$\begin{aligned} \alpha_2 &= 36u^3 + 24u^2 + 18u + 2 \\ &= (6u+4)\left(6u^2+1\right) + 2\,(6u-1) - 1 \\ \alpha_3 &= 36u^3 + 18u^2 + 12u - 2 \\ &= (6u+4)\left(6u^2+1\right) - \left(6u^2+1\right) + (6u-1) - 4 \\ \alpha_0 &= 6u^2 + 1 \\ \alpha_1 &= 1. \end{aligned}$$

So, f^{d_1} becomes

$$\begin{aligned} f^{s_1 \frac{p^4-p^2+1}{r}} &= f^{\alpha_0} f^{\alpha_0 p} f^{\alpha_0 p^2} f^{\alpha_0 p^3} \\ &= f^{6u^2+1} f^{p} \left(f^{(6u^2+1)(6u+4)}\right)^{p^2} \left(f^{2(6u-1)} f^{-1}\right)^{p^2} \left(f^{(6u^2+1)(6u+4)}\right)^{p^3} \\ &\quad \left(f^{-(6u^2+1)}\right)^{p^3} \left(f^{6u-1} f^{-4}\right)^{p^3}. \end{aligned} \tag{4}$$

In algorithm 11, we compute all the terms of (4) one by one and we accumulate their product in the temporary variable t_1.

Algorithm 11: new multiple of d	Term computed and comments	Cost
Input: $f,\ u$		
Output: $f^{s_1 \frac{p^4-p^2+1}{r}}$		
Temp. var.: t_0, t_1, t_2		
$t_0 \leftarrow f^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_1 \leftarrow t_0^2$		S_{12}
$t_1 \leftarrow t_0^2$		S_{12}
$t_0 \leftarrow t_0 t_1$	f^{-6u}	M_{12}
$t_1 \leftarrow f t_1$		M_{12}
$t_2 \leftarrow f^2$		S_{12}
$t_2 \leftarrow t_2^2$		S_{12}
$t_2 \leftarrow t_2 t_1$		M_{12}
$t_2 \leftarrow t_2^{-1}$	$f^{6u-1} f^{-4}$	
$t_2 \leftarrow t_2^{p^3}$	$\left(f^{6u-1} f^{-4}\right)^{p^3}$	F_{12}
$t_1 \leftarrow t_1^2$		S_{12}
$t_1 \leftarrow t_1 f$		M_{12}
$t_1 \leftarrow t_1^{-1}$	$f^{2(6u-1)} f^{-1}$	
$t_1 \leftarrow t_1^{p^2}$	$\left(f^{2(6u-1)} f^{-1}\right)^{p^2}$	F_{12}
$t_1 \leftarrow t_1 t_2$	# accumulate	M_{12}
$t_2 \leftarrow t_0^{-u}$		$(l_u-1)S_{12}+(w_u-1)M_{12}$
$t_2 \leftarrow f t_2$	f^{6u^2+1}	M_{12}
$t_1 \leftarrow t_2 t_1$	# accumulate	M_{12}
$t_0 \leftarrow t_2^{-1}$		
$t_2 \leftarrow t_0^{p^3}$	$\left(f^{-(6u^2+1)}\right)^{p^3}$	F_{12}
$t_1 \leftarrow t_1 t_2$	# accumulate	M_{12}
$t_2 \leftarrow t_0^{-6u-4}$		$(l_u+1)S_{12}+(w_u+1)M_{12}$
$t_0 \leftarrow (t_2)^{p^2}$	$\left(f^{(6u^2+1)(6u+4)}\right)^{p^2}$	F_{12}
$t_1 \leftarrow t_1 t_0$	# accumulate	M_{12}
$t_2 \leftarrow t_0^{p}$	$\left(f^{(6u^2+1)(6u+4)}\right)^{p^3}$	F_{12}
$t_1 \leftarrow t_1 t_2$	# accumulate	M_{12}
$t_0 \leftarrow f^{p}$		F_{12}
$t_1 \leftarrow t_1 t_0$	f^{d_2}	M_{12}
return t_1		

For this algorithm we used only 3 temporary variables in $\mathbb{F}_{p^{12}}$. The cost of computing f^{d_1} is detailed in algorithm 11.
As $-6u-4$ is a l_u+2-bit integer of Hamming weight w_u+2 (assuming u is sparse), we need w_u+1 multiplications and l_u+1 squarings in $\mathbb{F}_{p^{12}}$ to compute t_0^{-6u-4}. The cost of the computation of f^{d_1} is given step by step inside algorithm 11. The overall cost of this algorithm is then $3l_u+4$ squarings, $3w_u+10$ multiplications and 6 Frobenius in $\mathbb{F}_{p^{12}}$.

Example 4.6 *With the value of u chosen in example 2.1, the cost of computing our new multiple of the hard part of the final exponentiation is $19M_{12}+193S_{12}+6F_{12}$ and we use only 3 temporary variables in $\mathbb{F}_{p^{12}}$.*

The last two methods are variants of Fuentes et al method, so we compare them in the table 5.

Method	Complexity			Temp. var.
	S_{12}	M_{12}	F_{12}	
Fuentes (algorithm 6)	$3l_u$	$3w_u + 7$	3	5
New development of d' (algorithm 10)	$3l_u$	$3w_u + 7$	3	4
New multiple of d (algorithm 11)	$3l_u + 4$	$3w_u + 10$	6	3

Table 5: Comparison of Fuentes method and our variants

Example 4.7 *To have a full comparison, we choose the particular value of u given in example 2.1. We get table 6.*

Method	Complexity	Temp. var.
Fuentes Method	$16M_{12} + 189S_{12} + 3F_{12}$	5
New development of d'	$16M_{12} + 189S_{12} + 3F_{12}$	4
New multiple of d	$19M_{12} + 193S_{12} + 6F_{12}$	3

Table 6: Example of Table 5

Through these tables, we remark that using the new development of the multiple d', presented by Fuentes et al., we get the same complexity as Fuentes et al. method but we save memory resources. We can save more memory resources if we consider the new multiple of $\frac{p^4-p^2+1}{r}$ presented in this section but at the cost of some additional operations in $\mathbb{F}_{p^{12}}$.

5 Comparison

In this section we will compare state of the art methods of computing the hard part of the final exponentiation with our results in terms of efficiency but also in terms of memory usage. We first recall the complexity results obtained in this paper in table 7.

Method	Algo	Complexity			Complexity for u as in example 2.1	Temp. var. in $\mathbb{F}_{p^{12}}$
		S_{12}	M_{12}	F_{12}		
Naive	1	$12l_u$	$100w_u$		$759S_{12} + 306M_{12}$	1
Lucas sequence	2	$(12l_u + 4)M_6+$ $(12l_u + 2)S_6 + I_6$			$761S_6 + 763M_6$ $+I_6$	2
Devegili	4	$3l_u + 7$	$3w_u + 17$	4	$196S_{12} + 26M_{12}$ $+4F_{12}$	4
Our variant	**7**	$\mathbf{3l_u + 5}$	$\mathbf{3w_u + 12}$	**6**	$\mathbf{194S_{12} + 21M_{12}}$ $\mathbf{+6F_{12}}$	**3**
Addition chain	5	$3l_u + 1$	$3w_u + 10$	7	$190S_{12} + 19M_{12}$ $+7F_{12}$	9
Our variant	**8+9**	$\mathbf{3l_u + 3}$	$\mathbf{3w_u + 10}$	**6**	$\mathbf{192S_{12} + 19M_{12}}$ $\mathbf{+6F_{12}}$	**5**
Fuentes	6	$3l_u$	$3w_u + 7$	3	$189S_{12} + 16M_{12}$ $+3F_{12}$	5
Our variant	**10**	$\mathbf{3l_u}$	$\mathbf{3w_u + 7}$	**3**	$\mathbf{189S_{12} + 16M_{12}}$ $\mathbf{+3F_{12}}$	**4**
New multiple	**11**	$\mathbf{3l_u + 4}$	$\mathbf{3w_u + 10}$	**6**	$\mathbf{193S_{12} + 19M_{12}}$ $\mathbf{+6F_{12}}$	**3**

Table 7: Comparison of complexities to compute the hard part of the final exponentiation.

To obtain a more precise comparison we need to choose an extension tower so that we will be able to express all the complexities in term of $\mathbb{F}_p$ arithmetic. For this we choose the curve and the extension tower usually used in the literature. But another choice would certainly not change the final result. We choose the u given in example 2.1 and E the elliptic curve defined over $\mathbb{F}_p$ by

$$E : y^2 = x^3 + 2.$$

In this case -1 is not a square and $(1 + i)$ is neither a cube nor a square. So $\mathbb{F}_{p^{12}}$ is build using the following extension tower.

- $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 + 1)$
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/\big(v^3 - (1 + i)\big)$
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[z]/\big(z^2 - v\big)$

In this case the cost of arithmetic operations in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ are given by table 8 [15] .

Operation	Notation	Cost in $\mathbb{F}_p$
Multiplication in $\mathbb{F}_p$	M	M
Squaring in $\mathbb{F}_p$	S	S
Inversion in $\mathbb{F}_p$	I	I
Multiplication in $\mathbb{F}_{p^2}$	M_2	$3\,M$
Squaring in $\mathbb{F}_{p^2}$	S_2	$2\,M$
Multiplication in $\mathbb{F}_{p^6}$	M_6	$18\,M$
Squaring in $\mathbb{F}_{p^6}$	S_6	$12\,M$
Inversion in $\mathbb{F}_{p^6}$	I_6	$37\,M + I$
Multiplication in $\mathbb{F}_{p^{12}}$	M_{12}	$54\,M$
Cyclotomic squaring in $\mathbb{F}_{p^{12}}$	S_{12}	$18\,M$
Frobenius in $\mathbb{F}_{p^{12}}$	F_{12}	$15\,M$

Table 8: Operations cost in the extension tower

In this paper, we are not only interested in complexity but also in memory usage. Then we have to determine the number of $\mathbb{F}_p$ variables required for each algorithm. Our estimation is based on the fact that a temporary variable in $\mathbb{F}_{p^{12}}$ requires 12 temporary variables in $\mathbb{F}_p$. But this is not sufficient since additional temporary variables are required during $\mathbb{F}_{p^{12}}$ operations. It is not difficult to see that at least 10 of them are necessary for a multiplication and less than 10 for other operations (squaring, Frobenius, inversion). Moreover the input element f is an element of $\mathbb{F}_{p^{12}}$ and then requires 12 elements in $\mathbb{F}_p$ to be stored. This means that 22 variables in $\mathbb{F}_p$ are necessary in addition to the $\mathbb{F}_{p^{12}}$ temporary variables used in the algorithms presented in this paper. Note that depending on the context the variables used for storing f could be reused for computations. However, making such an assumption does not change the conclusions of our study.

Assuming this particular (but wide-spread) context, Let us now give in table 9 the cost of the algorithms studied in this paper in terms of $\mathbb{F}_p$ arithmetic.

Method	Algorithm	Cost in $\mathbb{F}_p$	Cost saving	Temp. var. in $\mathbb{F}_p$	Memory saving
Naive	1	$30186\,M$		34	
Lucas Sequence	2	$I + 22903\,M$		46	
Devigili	4	$4992\,M$		70	
Our variant	**7**	**4716 M**	**5.5%**	**58**	**17%**
Addition chain	5	$4551\,M$		130	
Our variant	**8+9**	**4572 M**	**−0.4%**	**82**	**37%**
Fuentes method	6	$4311\,M$		82	
Our variant	**10**	**4311 M**	**0%**	**70**	**15%**
New multiple	**11**	**4590 M**	**−6.4%**	**58**	**29%**

Table 9: Comparison and savings obtained by our variants in terms of $\mathbb{F}_p$ arithmetic

6 Conclusion

In restricted environments we must find a balance between efficiency and memory resources. In this paper we suggested four new methods for the implementation of the hard part of the final exponentiation in the computing of the Tate pairings and its derivates, which are faster or competitive, generally applicable and which require less memory than previous methods in the literature.
We first presented a new development of the exponent $\frac{p^4-p^2+1}{r}$ which is a variant of Devegili et al. method. This new development is certainly faster than the method described in [4]. Moreover, the memory resources of this new method are significantly less than the memory resources of Devegili et al. method.
Then, we presented a new addition chain which is a variant of Scott et al. addition chain [5]. By presenting this new addition chain we save an important number of temporary variables but our algorithm is insignificantly slower.
We also presented a new way of writing the exponent d' presented by Fuentes et al. where we use less temporary variables to compute $f^{d'}$. Finally we produced a new multiple d_1 of d such that the computation of $f^{d'}$ requires less memory resources but, in this case, our algorithm is also slightly slower. We implemented our new algorithms in Sage [**?**] to verify their correctness [**?**].
As a conclusion, our new methods for computing the hard part of the final exponentiation are in some cases more efficient and in others cases slightly slower than previous methods in the literature but they are always less memory intensive. For that our methods are an interesting alternative for pairing implementation in restricted environments.

Acknowledgements. The authors thank John Boxall for helpful discussions and comments on this paper.

References

[1] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, pages 209–223, 2010.

[2] Michael Scott and Paulo S. L. M. Barreto. Compressed pairings. In *Advances in cryptology—CRYPTO 2004*, volume 3152 of *Lecture Notes in Comput. Sci.*, pages 140–156. Springer, Berlin, 2004.

[3] Martijn Stam and Arjen K. Lenstra. Efficient subgroup exponentiation in quadratic and sixth degree extensions. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 318–332, 2002.

[4] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing cryptographic pairings over barreto-naehrig curves. In *Pairing-Based Cryptography - Pairing 2007, First International Conference, Tokyo, Japan, July 2-4, 2007, Proceedings*, pages 197–207, 2007.

[5] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography - Pairing 2009, Third International Conference, Palo Alto, CA, USA, August 12-14, 2009, Proceedings*, pages 78–88, 2009.

[6] Laura Fuentes Castaneda, Edward Knapp, and Francisco Rodrguez Henrquez. Faster hashing to ${\mathbb G}_2$. In *Selected Areas in Cryptography - 18th International Workshop, 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, pages 412–430, 2011.

[7] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, pages 319–331, 2005.

[8] Yasuyuki Nogami, Masataka Akane, Yumi Sakemi, Hidehiro Katou, and Yoshitaka Morikawa. Integer variable chi-based ate pairing. In *Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings*, pages 178–191, 2008.

[9] Robert Granger, Dan Page, and Nigel P. Smart. High security pairing-based cryptography revisited. In *Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings*, pages 480–494, 2006.

[10] Lei Hu, Jun-Wu Dong, and Dingyi Pei. Implementation of cryptosystems based on tate pairing. *J. Comput. Sci. Technol.*, 20(2):264–269, 2005.

[11] M. Joye and J. J. Quisquater. Efficient computation of full lucas sequences. *Electronics Letters*, 36(6):537–538, 1996.

[12] Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2003.

[13] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp.*, 48(177):243–264, 1987.

[14] Jorge Olivos. On vectorial addition chains. *J. Algorithms*, 2(1):13–21, 1981.

[15] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren, editors. *Handbook of elliptic and hyperelliptic curve cryptography.* Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2006.

[16] Ionica Smeets, Arjen K. Lenstra, Hendrik Lenstra, László Lovász, and Peter van Emde Boas. The history of the lll-algorithm. In *The LLL Algorithm - Survey and Applications*, pages 1–17. 2010.

[17] Diego F. Aranha, Paulo S. L. M. Barreto, Patrick Longa, and Jefferson E. Ricardini. The realm of the pairings. In *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, pages 3–25, 2013.

[18] Jean-Luc Beuchat, Jorge Enrique González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. In *Pairing-Based Cryptography - Pairing 2010 - 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings*, pages 21–39, 2010.

[19] W. A. Stein et al. *Sage Mathematics Software (Version 5.9)*. The Sage Development Team, 2015. `http://www.sagemath.org`.

[20] S. Duquesne and L. Ghammam. http://sage.lacim.uqam.ca/home/pub/41/.