Hardware Implementations of Pairings at Updated Security Levels

Arthur Lavice[1,2,3] [0000-0003-2804-1608], Nadia El Mrabet[1] [0000-0003-3840-584X], Alexandre Berzati[2], Jean-Baptiste Rigaud[1] [0000-0003-7394-5345], and Julien Proy[2] [0000-0002-0402-6780]

[1] Mines Saint-Etienne, CEA, Leti, Centre CMP, F-13541 Gardanne France, firstname.lastname@emse.fr

[2] Thales DIS Design Services SAS, Meyreuil, France, firstname.lastname@thalesgroup.com

[3] ARMINES, Paris, France

Abstract. Pairings are cornerstones to several interesting cryptographic protocols including Non-interactive ARgument of Knowledge currently used in Zcash cryptocurrency. The Kim and Barbulescu Number Field Sieve attack has weakened pairing-friendly curves. Most impacted are the famous BN curves which now require an increase of the parameters to provide equivalent security. Recent cost estimations of pairings have recommended switching to other curves but their selections are no longer clearly straightforward. This paper aims at providing the first hardware-based pairing implementations on the best curve candidates at both 128-bit and 192-bit security levels. The proposed architecture intends to fit both lightweight FPGA and ASIC purposes and the design is prototyped on a Kintex-7 FPGA device. It computes a pairing within 42.7 ms for 128-bit of security and 184.2 ms for 192-bit.

Keywords: Pairings · Lightweight Hardware/Software Implementations · Updated Key Size · Parallel Computation

1 Introduction

Pairings are cryptographic tools whose bilinearity property allows finding efficient solutions to many protocols such as the tripartite Diffie-Hellman key exchange [23] or short signature schemes [9]. It also enables the creation of new protocols such as Identity-Based Encryption [8] or zero knowledge-Succinct Non-interactive ARgument of Knowledge (zk-SNARK) [7] used in Zcash cryptocurrency. A pairing is a bilinear and non-degenerate map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ where $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) is generally taken as a subgroup of an elliptic curve over $E(\mathbb{F}_p)$ (resp. $E(\mathbb{F}_{p^k})$) and $\mathbb{G}_3$ is usually a subgroup of $\mathbb{F}_{p^k}$. $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_3$ are subgroups of prime order r. Pairing computation strongly depends on curve parameters such as $\rho = \frac{\log_2(p)}{\log_2(r)}$. But pairings friendly elliptic curves are rare and a lot of research has been done to find suitable curves such as BLS [5], BN [6], KSS [24] and DCC [13] curves. A taxonomy of these methodologies is found in [15].

Through numerous arithmetic optimizations, BN curves were found to be the best choice for pairings at the 128-bit security level. But the Kim and Barbulescu attack [27] has improved the discrete logarithm attack against these curves and thus threatens the security of many families of pairings. Since this attack depends on curve parameters, it has reshuffled the field and, BN curves are no longer the best ones. Now, performance time seems to be similar on several other curves at the 128-bit security level. In [4], the authors make an extensive literature review to study actual security of pairings. Their estimations only take into account the complexity of modular multiplications and neglect other operations. This approach may not be sufficient to determine the best curves at the 128-bit security level. Another approach from [18] is to create new pairing-friendly curves resistant to the Kim and Barbulescu attack.

There are many time-efficient software versions of pairings but their implementations on constrained devices are challenging. Indeed, one pairing computation could take seconds to complete with the 128-bit security level [36] prior to the Kim and Barbulescu attack. Having fast pairing implementations on small devices is essential,for example, to guarantee user-friendly utilization of Zcash currency on a hardware wallet.

Our contribution. This paper proposes a way to efficiently support emerging curves at both the 128 and the 192-bit security levels. A new formula for squaring over cyclotomic fields $\mathbb{G}_{\phi_2(q)}$ is proposed and is more suitable for curves introduced in [18] than the previous one given in [17]. Our work provides a $time \times area$ efficient lightweight coprocessor with configurable modulo to support multiple curves. This coprocessor enables parallel computation of modular multiplications with additional operations in order to cut down additional costs brought by neglected operations such as modular additions. This paper also presents a hardware-software co-design architecture based on a Microblaze CPU to demonstrate the performance of our coprocessor. The genericity of our design allows us to give the first comparison between hardware implementations of several pairings at updated security levels with the same design. Finally, this paper shows that, following the Kim and Barbulescu attack, the optimal choice of a curve at the 128-bit security level depends on the target platform.

Organization of the paper. Section 2 provides some mathematical background on pairings as well as a summary of the latest estimations of pairing costs at updated security levels. Section 3 details curve parameters and recalls some arithmetic optimization of the Miller algorithm and the final exponentiation formula for the 3 best candidates at the 128-bit security level and for the best candidate at the 192-bit security level. It also provides a new formula for cyclotomic squaring in $\mathbb{G}_{\phi_2(q^2)}$ (see Eq. 13). In Section 4, we present our dedicated hardware implementation used to accelerate operations on the base field and our hardware/software codesign used for pairing implementations. Finally, we summarize our work and discuss future research directions in Section 5.

Notation. In this paper, we will use the following notation. $\mathbb{F}_p$: a finite field of prime characteristic p. $\mathbb{F}_{p^k}$: an extension field of degree k of $\mathbb{F}_p$. $\mathbb{G}[r]$: a subgroup of order r of $\mathbb{G}$. e (resp. n): the number of words used to represent numbers in $\mathbb{F}_p$ (resp. $\log_2(p)$). M_q (resp, S_q, A_q, Dbl_q): a multiplication (resp. square, addition, double) in $\mathbb{F}_q$. $Mulx_q$: a multiplication of an element $\mathbb{F}_q$ by x, a *small* constant in $\mathbb{F}_q$.

2 Background on Pairings

The following part gives some background about pairings and their implementations. There are several pairings such as [22,20,34] but constructions of the most efficient ones are similar to the Ate pairing defined below [19].

2.1 Introduction and definition

Definition 1. ***(Ate pairing).*** *Let E be an elliptic curve defined over $\mathbb{F}_p$; r be a large prime divisor of $\#E(\mathbb{F}_p)$; t be the trace of E and k be the embedding degree of E with respect to r (k is the smallest integer such thar $r|p^k-1$).*
Let $\mathbb{G}_1 \subseteq E(\mathbb{F}_p)[r]$, $\mathbb{G}_2 \subseteq E(\mathbb{F}_{p^k})[r]$, $\mathbb{G}_3 = \mathbb{F}_{p^k}[r]$ and $u = t-1$.
The Ate pairing is defined as:

$$\begin{cases} e : E(\mathbb{F}_p)[r] \times E(\mathbb{F}_{p^k})[r] \rightarrow \mathbb{F}_{p^k}[r], \\ (P,Q) \qquad\qquad\qquad\quad \rightarrowtail f_{u,Q}(P)^{\frac{p^k-1}{r}}. \end{cases} \tag{1}$$

The computation of such pairing relies on two distinct steps. First, the function $f_{u,Q}(P)$ is computed with Miller's algorithm (see Algorithm 1). The complexity of Miller's algorithm depends on the Hamming Weight(HW) and the $\log_2$ of u. To decrease the complexity of Miller's algorithm, the Non-Adjacent Form (NAF) is classically used to represent u. The second part is the so-called final exponentiation. It raises $f_{u,Q}(P)$ at the power of $(p^k-1)/r$.

Algorithm 1 Miller's algorithm [30]

```
Input: $u = (u_{n-1} \dots u_0)$ NAF decomposition of $t-1, P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^k})$
Output: $f_{u,Q}(P) \in \mathbb{F}_{p^k}[r]$)
1: $T \leftarrow Q$; $f_1 \leftarrow 1$;
2: for $i = n-2, ..., 0$ do
3:     $T \leftarrow 2T$; $f_1 \leftarrow f_1^2 \times l_{Q,Q}(P)/v_{2Q}(P)$; Where $l_{Q,Q}$ is the tangent of $E$ at point $Q$,
       and $v_{2Q}$ is the vertical line of $E$ at point $[2]Q$.
4:     if $u_i = 1$ then
5:         $T \leftarrow T+Q$; $f_1 \leftarrow f_1 \times l_{Q,T}(P)/v_{Q+T}(P)$; Where $l_{Q,T}$ is the line $(QT)$,
           and $v_{Q+T}$ is the vertical line of $E$ at point $Q+T$.
6:     else if $u_i = -1$ then
7:         $T \leftarrow T+Q$; $f_1 \leftarrow f_1 \times l_{-Q,T}(P)/v_{-Q+T}(P)$;
8:     end if
9: end for
10: return $f_1 = f_{u,Q}(P)$
```

2.2 Pairing optimizations

Pairing implementations are based on different arithmetics presented in Fig. 1. Elliptic curves and extension field arithmetics depend on modular arithmetic which again depends on integer arithmetic. Curve parameters have a direct impact on the complexity of these operations. The three principal optimizations regarding these parameters are cited below:

Embedding degree k. It is a crucial parameter since it defines the extension fields used during computations. Having k in the form $k = 2^i 3^j$ enables efficient extension field arithmetic with Karatsuba and Toom-Cook formulae [28] and is one prerequisite to using twisted curves during computations [14].

Twisted curves. Let E be an elliptic curve defined over $\mathbb{F}_{p^k}$. An elliptic curve $\tilde{E}$ defined over $\mathbb{F}_{p^{k/d}}$ is called a twisted curve of degree d of E if there exists an isomorphism ψ_d from $\tilde{E}$ into E According to the value of k, the potential degrees for a twist are $d = 2, 3, 4$ or 6. Computing Miller's algorithm on the twisted curve also enables avoiding the computation of the denominator when k is a multiple of 2 [29] or 3 [38]. Twist also makes line and tangent evaluations sparse elements of $\mathbb{F}_{p^k}$ (with at least one null coefficient).

Generation of curves. The generation of pairing-friendly elliptic curves is the most important step because it conditions the use of optimizations cited in this section. A family of pairing-friendly elliptic curves is a mathematical method to create curves with a prescribed embedding degree as in the taxonomy presented in [15]. The characteristic p, the trace t and a large prime factor of r such that $r|\#E(\mathbb{F}_p)$ are given by polynomials evaluated in an integer u. This integer (u) has a significant impact on the complexity of pairings. Hence it is important to choose an appropriate generator u with low Hamming Weight representation.

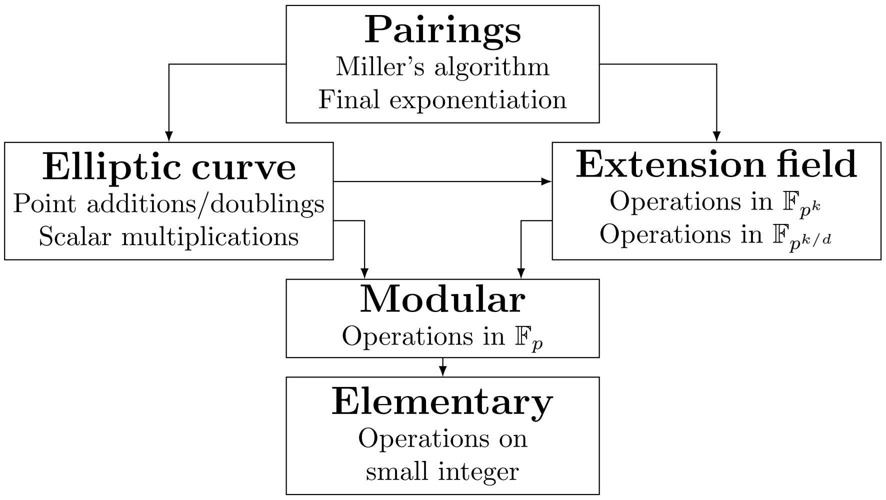

Fig. 1: Required operations for pairings

3 Selection of pairing-friendly curves and parameters

3.1 Summary of estimated pairings complexity

The arithmetic required to implement a pairing depends on curve parameters. For this reason, much attention was given to Optimal Ate pairings [34] on BN curves. But parameters that make a curve pairing-friendly also make it vulnerable to the extended tower Number Field Sieve (NFS) attack presented in [27]. As a result, recent security analysis of pairings presented in [3] and [4] have led to new key size requirements. BN curves are the most impacted and are no longer considered as interesting curves for pairings. Recent research has identified more resilient curves against such attacks. To our knowledge, the most promising alternative are curves presented in [18].

In [4] the authors estimate pairings complexity by taking into account only the cost of modular multiplications. They use the compressed squaring formulae given in [25] to estimate the complexity of the final exponentiation of pairings on BLS12, BLS24 and KSS18 [24] curves which admit a twist of degree 6. Compressed squaring is an interesting method since it decreases the number of multiplications but it computes several modular inversions. To compute modular inversions of a number $\alpha \in \mathbb{F}_p$ with the coprocessor presented in Section 4, we have to use Fermat's theorem and compute α raised to the power of $p-2$. Then, the cost of a modular inversion is approximated for naive implementation at the cost of $\lceil 3n/2 \rceil M_p$, which is significant.

Moreover, compressed squaring requires storing several elements in $\mathbb{F}_{p^k}$ [2]. This increases the memory needed by approximately 30% for a slight latency benefit compared to cyclotomic squaring of [17]. We choose to use the formula proposed in [17] in order to target implementation on constrained devices. Table 1 presents the best pairing candidates at the 128-bit and 192-bit security levels according to [4] and [18]. In what follows, we denote by GMT8 the curve presented in [18] with embedding degree $k = 8$.

The complexity of Miller's algorithm is critical when computing the product of pairings in short signatures for example. When computing scalar multiplications on $\mathbb{G}_1$, $\log_2(r)$ and $\log_2(p)$ are crucial parameters; $\log_2(r)$, $\log_2(p)$, the embedding degree k and the degree of the twist d are also important when computing a scalar multiplication on $\mathbb{G}_2$.

Table 1: Theoretical complexity of pairings at different security levels [4,18]
**: M_p cost depends on $\log_2(p)$*

Security	**Method**	k	$\log_2(p)$	**Miller (M_p*)**	**Final Expo. (M_p*)**	**Total (M_p*)**
128-bit	**BLS**	**24**	319	9 381	**23 400**	**32 781**
	BLS	**12**	460	7 438	**8 151**	**15 589**
	GMT	**8**	**544**	**4 502**	**7 056**	**11 558**
	DCC	15	383	6 836	19 190	26 026
	KSS	16	340	7 534	18 514	26 048
192-bit	**BLS**	**24**	559	16 368	**36 573**	**52 941**
	KSS	18	657	13 488	30 473	43 961

At the 128-bit security level, several curves have similar complexity but have different points of interest. This is especially the case of the first curve from Table 1 (BLS24) and the second candidate (GMT8). At the 192-bit security level, KSS18 and BLS24 have similar complexity but also similar parameters. KSS18 is a bit less interesting than BLS24 when looking at the complexity of Miller's algorithm, the characteristic p, or the order r. At this security level, BLS24 should be the better choice whatever the target application. Based on these estimations, we choose to implement the two best pairing candidates at the 128-bit security level: BLS24 and GMT8. Since BLS12 is still one of the best candidates, we consider it as the reference curve at the 128-bit security level and implement it. We also provide an implementation of BLS24 at 192-bit security.

In the following section, we present the parameters of the chosen curves and provide some aspects of their implementation.

3.2 Pairing arithmetic and implementation aspects

All the selected curves (BLS12, BLS24, GMT8) have an even embedding degree. In this case, the vertical lines computed in Algorithm 1 are elements of $\mathbb{F}_{p^{k/d}}$ and will be sent to 1 during the final exponentiation. Hence the computation of denominators v_{2Q} or v_{Q+T} can be omitted during Miller's algorithm.

"High-level" operations such as extension fields or elliptic curve operations can be computed with a succession of modular operations (or operations in the base field: $\mathbb{F}_p$). Searching for a cost-efficient hardware coprocessor to compute these operations is a way to enhance the efficiency of pairing implementations. Moreover, arithmetic used to implement pairings highly depends on the pairing family. Hence, the sequence of modular operations also depends on the curve. To ensure the flexibility of our design, we choose to focus on modular operations to design a hardware accelerator suitable for all curves.

Common operations: multiplication and squaring in $\mathbb{F}_{p^2}$. During pairing computations, most of the operations are computed over the extension field $\mathbb{F}_{p^{k/d}}$. For the selected curves, $\mathbb{F}_{p^{k/d}} = \mathbb{F}_{p^2}$ or $\mathbb{F}_{p^4}$. An element A of $\mathbb{F}_{p^k}$ is a polynomial of degree n, with $0 \leq n \leq k-1$ and with its coefficient in $\mathbb{F}_p$. Let P be an irreducible polynomial of degree k. Let A and B be two elements of $\mathbb{F}_{p^k}$. The result C of the multiplication of A by B is defined as the Euclidean remainder of the polynomial $A \times B$ by the polynomial P. As previously said, the curves selected in our study all have an embedding degree of $k = 2^i 3^j$. To construct extension fields of these embedding degrees, the classical method is to use extension field towers. For instance, $\mathbb{F}_{p^4}$ can be seen as an extension of degree 2 of $\mathbb{F}_{p^2}$. Algorithm 2 (resp. Algorithm 3) is the standard way to compute a multiplication (resp. a square) in extension fields of degree 2.

Curves admitting a twist of degree 6. BLS curves are defined over $\mathbb{F}_p$ by $E : y^2 = x^3 + b$ and by a parameter $u \in \mathbb{Z}$ such that the parameters p, r, t are evaluations of some polynomials at u ($p = p(u)$, $r = r(u)$, and $t = t(u)$).

In our implementations, we select the same parameters as in [4] which are $u = -2^{32}+2^{28}+2^{12}$ (resp. $u = -2^{56}-2^{43}+2^{9}-2^{6}$) For BLS24 at 128-bit (resp. 192-bit) security and $u = -2^{77}+2^{50}+2^{33}$ for BLS12.

Algorithm 2 Multiplication in $\mathbb{F}_{p^k} = \mathbb{F}_{p^{k/2}}[g], g^2 = v, v \in \mathbb{F}_{p^{k/2}}$

```
Input: $A = a_0 + a_1 g$,
$B = b_0 + b_1 g \in \mathbb{F}_{p^k}$
Output: $Z \leftarrow AB \in \mathbb{F}_{p^k}$
Cost: $3M_{p^{k/2}} + 5A_{p^{k/2}} + 1Mulv_{p^{k/2}}$
Begin
  1: $t_0 \leftarrow a_0 b_0$; $t_1 \leftarrow a_0 + a_1$
  2: $z_0 \leftarrow b_0 + b_1$; $z_1 \leftarrow a_1 b_1$;
  3: $t_1 \leftarrow t_1 z_0$; $z_0 \leftarrow t_0 - v z_1$;
  4: $z_1 \leftarrow t_0 + z_1$; $z_1 \leftarrow z_1 - t_1$;
  5: return $Z = z_0 + z_1 g$;
End
```

Algorithm 3 Square in $\mathbb{F}_{p^k} = \mathbb{F}_{p^{k/2}}[g], g^2 = v, v \in \mathbb{F}_{p^{k/2}}$

```
Input: $A = a_0 + a_1 g \in \mathbb{F}_{p^k}$,
Output: $Z \leftarrow A^2 \in \mathbb{F}_{p^k}$
Cost: $2M_{p^{k/2}} + 4A_{p^{k/2}} + 1Dbl_{p^{k/2}} + 2Mulv_{p^{k/2}}$
Begin
  1: $z_0 \leftarrow a_0 a_1$; $t_0 \leftarrow a_0 - a_1$;
  2: $t_1 \leftarrow a_0 - v a_1$; $t_0 \leftarrow t_0 t_1$;
  3: $z_1 \leftarrow 2 z_0$; $z_0 \leftarrow t_0 + (v+1) z_0$;
  4: return $Z = z_0 + z_1 g$;
End
```

To our knowledge, the most efficient way to compute Miller's algorithm on these curves is to use mixed affine-projective coordinates along with the line evaluations proposed in [12]. Then the characteristic $p(u)$, the order of the subgroups $r(u)$ and the trace $t(u)$ of E are given by Eq. 2 for BLS12 and by Eq. 3 for BLS24.

$$\begin{cases} r(u) = u^4 - u^2 + 1, \\ p(u) = (u-1)^2 r/3 + u, \\ t(u) = u + 1. \end{cases} \quad (2) \qquad \begin{cases} r(u) = u^8 - u^4 + 1, \\ p(u) = (u-1)^2 r/3 + u, \\ t(u) = u + 1. \end{cases} \quad (3)$$

The same extension fields and elliptic curves as in [26] are used for BLS12 (see Eq. 4) at 128-bit security and for BLS24 (see Eq. 5) at 192-bit security. Multiplications and squares over $\mathbb{F}_{p^{12}}$ (resp. $\mathbb{F}_{p^{24}}$) are computed with formulae given in [28] for cubic extension.

$$\begin{cases} \log_2(p) = 461, \\ \log_2(r) = 308, \\ \mathbb{F}_{p^2} = \mathbb{F}_p[i], i^2 = -1, \\ \mathbb{F}_{p^4} = \mathbb{F}_{p^2}[v], v^2 = i+1, \\ \mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[g], g^3 = v, \\ E(\mathbb{F}_p) : y^2 = x^3 + 4, \\ \tilde{E}(\mathbb{F}_{p^2}) : y^2 = x^3 + 4(i+1). \end{cases} \quad (4) \qquad \begin{cases} \log_2(p) = 559, \\ \log_2(r) = 449, \\ \mathbb{F}_{p^2} = \mathbb{F}_p[i], i^2 = -1, \\ \mathbb{F}_{p^4} = \mathbb{F}_{p^2}[v], v^2 = i+1, \\ \mathbb{F}_{p^{24}} = \mathbb{F}_{p^4}[g'], g'^6 = v, \\ E(\mathbb{F}_p) : y^2 = x^3 + 5, \\ \tilde{E}(\mathbb{F}_{p^4}) : y^2 = x^3 + 9(-i+1)v/2. \end{cases} \quad (5)$$

For BLS24 at 128-bit security, g' can not be chosen such that $g'^{12} = i+1$ as in Eq. 4 because this extension tower does not construct a field. Therefore, we choose to define $\mathbb{F}_{p^{24}}$ and E as described in Eq. 6 to simplify the expression of $\tilde{E}$, and to ease the computation of Miller's algorithm.

$$\begin{cases} \log_2(p) = 318, & \mathbb{F}_{p^{24}} = \mathbb{F}_{p^4}[g'], g'^6 = v, \\ \log_2(r) = 256, & E(\mathbb{F}_p) : y^2 = x^3 + 5, \\ \mathbb{F}_{p^2} = \mathbb{F}_p[i], i^2 = -1, & \tilde{E}(\mathbb{F}_{p^4}) : y^2 = x^3 + 5/v, \\ \mathbb{F}_{p^4} = \mathbb{F}_{p^2}[v], v^2 = i + 3, & \tilde{E}(\mathbb{F}_{p^4}) : y^2 = x^3 + (-i+3)v/2. \end{cases} \tag{6}$$

The case of GMT8. The curves proposed in [18] differ from BLS12 or BLS24 curves as the modulo p and the order of subgroups r can not be represented by polynomials in the variable u. Moreover, This curve admits a twist of degree $d = 4$ and is generated using a variant of the Cocks-Pinch algorithm [11]. In [18], they define $\mathbb{F}_{p^8}$ as $\mathbb{F}_p[g]$ with $g^8 = 5$.

The most efficient formula to compute Miller's algorithm for these kinds of curves is the one proposed in [12] along with the mixed affine-"weight-(1, 2) coordinates." These coordinates represent points of E by $(X : Y : Z)$, which corresponds to the affine point (x, y) where $x = \frac{X}{Z}$ and $y = \frac{Y}{Z^2}$. The parameters of the GMT8 curve at 128-bit security are given in Eq. 7.

$$\begin{cases} r = 0xff0060739e18d7594a978b0ab6ae4ce3d & \log_2(r) = 256, \\ \quad bfd52a9d00197603fffdf0000000101, & \log_2(p) = 544, \\ p = 0xbb9dfd549299f1c803ddd5d7c05e7cc03 & \mathbb{F}_{p^2} = \mathbb{F}_p[v], v^2 = 5, \\ \quad 73d9b1ac15b47aa5aa84626f33e58fe6694 & \mathbb{F}_{p^4} = \mathbb{F}_{p^2}[u], u^2 = v, \\ \quad 3943049031ae4ca1d2719b3a84fa363bcd2 & \mathbb{F}_{p^8} = \mathbb{F}_{p^4}[g], g^2 = u, \\ \quad 539a5cd02c6f4b6b645a58c1085e14411, & E(\mathbb{F}_p) : y^2 = x^3 + 2x, \\ t = 2^{64} - 2^{54} + 2^{37} + 2^{32} - 4, & \tilde{E}(\mathbb{F}_{p^2}) : y^2 = x^3 + 2vx. \end{cases} \tag{7}$$

Summary of operations required by Miller's algorithm. For curves $E : y^2 = x^3 + ax + b$ that admit a sextic or quartic twist, the complexity of Miller's algorithm of the selected curves is given in Table 2. For the sake of simplification, p^k (resp $p^{k/d}$) is denoteted q (resp. l). Since a and b are small coefficients, Multiplication by a or b can be computed without modular multiplications. Pairing implementations on BLS12 and BLS24 curves both rely on $\mathbb{F}_{p^2} = \mathbb{F}_p[i]$ but GMT8 relies on $\mathbb{F}_{p^2} = \mathbb{F}_p[v]$. The formulae used to compute operations on these fields are similar since they only differ in the reduction step (see Algorithm 2 and Algorithm 3).

The dependency between operations required by Miller's algorithm for the selected curves is summarized in Fig. 2, where operations specific to each curve are framed. Modular operations are the common points of selected curves even if the size of the characteristic (p) differs between BLS24, BLS12 and GMT8.

Table 2: The complexity of Miller's step using twist

Operation	**Complexity**	
Twist	Sextic twist	Quartic twist
Doubling(D)	$k/3.M_p + 3M_l + 5S_l + M_q + S_q$	$k/2.M_p + 3M_l + 6S_l + M_q + S_q + Mula_l$
Mixed add(MA)	$k/3.M_p + 10M_l + 2S_l + M_q + Mulb_l$	$k/2.M_p + 9M_l + 5S_l + M_q$
Miller (total)	$\log_2(u).D + HW(u).MA$	

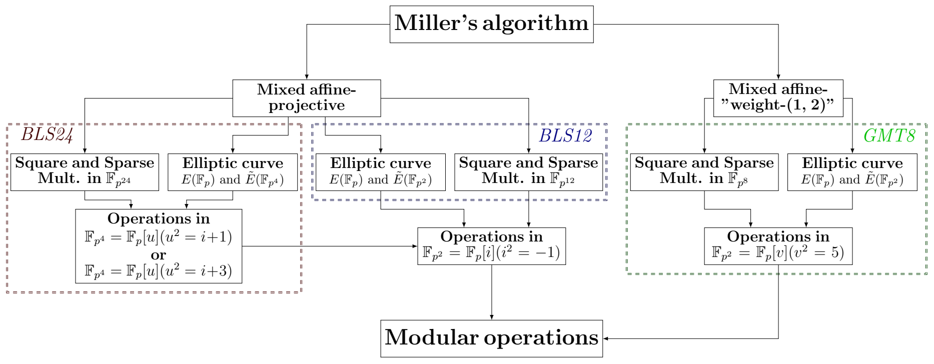
Miller's algorithm
Mixed affine-projective
Mixed affine-"weight-(1, 2)"
BLS24
BLS12
GMT8
Square and Sparse Mult. in $\mathbb{F}_{p^{24}}$
Elliptic curve $E(\mathbb{F}_p)$ and $\tilde{E}(\mathbb{F}_{p^4})$
Elliptic curve $E(\mathbb{F}_p)$ and $\tilde{E}(\mathbb{F}_{p^2})$
Square and Sparse Mult. in $\mathbb{F}_{p^{12}}$
Square and Sparse Mult. in $\mathbb{F}_{p^8}$
Elliptic curve $E(\mathbb{F}_p)$ and $\tilde{E}(\mathbb{F}_{p^2})$
Operations in $\mathbb{F}_{p^4} = \mathbb{F}_p[u](u^2 = i+1)$ or $\mathbb{F}_{p^4} = \mathbb{F}_p[u](u^2 = i+3)$
Operations in $\mathbb{F}_{p^2} = \mathbb{F}_p[i](i^2 = -1)$
Operations in $\mathbb{F}_{p^2} = \mathbb{F}_p[v](v^2 = 5)$
Modular operations

Fig. 2: Operations during Miller's algorithm selected curves

3.3 Implementation of the final exponentiation

The second part of a pairing calculation is computing Miller's algorithm result raised to the power of $\frac{p^k-1}{r}$. This is called the final exponentiation and its complexity depends on different curve parameters. This section presents arithmetic optimizations used to compute this step on selected curves.

Curves admitting a twist of degree 6. The decomposition of the final exponentiation is a well-known optimization of pairings. To our knowledge, the most efficient is proposed in [16] for both BLS12 and BLS24. They used the parametrization of modulo p to provide fast and memory-efficient implementations. For BLS12, the ratio $\frac{p^{12}-1}{r}$ is split into an easy part $(p^6-1)(p^2+1)$, and a hard one $(p^4-p^2+1)/r$. Then, the hard part is also decomposed as: $(p^4-p^2+1)/r = \lambda_0+\lambda_1 p+\lambda_2 p^2+\lambda_3 p^3$ and the λ_i are calculated according to Eq. 8. The same methodology applies for BLS24. The ratio $\frac{p^{24}-1}{r}$ is split into an easy part $(p^{12}-1)(p^4+1)$ and a hard one $(p^8-p^4+1)/r$. Then, the hard part is decomposed into: $(p^8-p^4-1)/r = \lambda_0+\lambda_1 p+\lambda_2 p^2+\lambda_3 p^3+\lambda_4 p^4+\lambda_5 p^5+\lambda_6 p^6+\lambda_7 p^7$ and the λ_i are calculated according to Eq. 9.

$$\begin{cases} \lambda_3 = u^2-2u+1, \\ \lambda_2 = \lambda_3 u, \\ \lambda_1 = \lambda_2 u - \lambda_3, \\ \lambda_0 = \lambda_1 u + 3. \end{cases} \quad (8) \qquad \begin{cases} \lambda_7 = u^2-2u+1, & \lambda_3 = \lambda_4 u - \lambda_7, \\ \lambda_6 = \lambda_7 u, & \lambda_2 = \lambda_3 u, \\ \lambda_5 = \lambda_6 u, & \lambda_1 = \lambda_2 u, \\ \lambda_4 = \lambda_5 u, & \lambda_0 = \lambda_1 u + 3. \end{cases} \quad (9)$$

After the easy part of the final exponentiation, all computations are done in cyclotomic subgroups of $\mathbb{F}_{p^k}$. This allows faster squaring formulae as the ones presented in [17].

The case of GMT8. The curves presented in [18] are defined over a finite field of characteristic p, where p can not be represented as a polynomial evaluated in u. Hence, formulae similar to Eq. 8 or Eq. 9 for BLS are not available. The method proposed by the authors of [18] consists in breaking down the final exponentiation $\frac{p^8-1}{r}$ again into an easy part (p^4-1) and a hard part $(\frac{p^4+1}{r})$. Then, the hard part is represented as in Eq. 10.

Once again, the second part of the final exponentiation is done in cyclotomic subgroups. Since this curve admits a twist of even degree ($d = 4$), a square in this subgroup costs approximately 2 squares in $\mathbb{F}_{p^4}$.

$$\begin{cases} t_0 &= p+1 \mod r, \\ c &= \frac{p+1-t_0}{r}, \\ \frac{p^4+1}{r} &= \frac{(t_0-1)^4+1}{r} + (p+t_0-1)(p^2+(t_0-1)^2)c. \end{cases} \tag{10}$$

Let $\mathbb{F}_{p^8}$ and $\mathbb{F}_{p^4}$ be defined as in Eq. 7 and let $a = a_0 + a_1 g$, with $a \in \mathbb{F}_{p^8}$, $a_0, a_1 \in \mathbb{F}_{p^4}$. A square in $\mathbb{G}_{\phi_2(\mathbb{F}_{p^4})}$ is computed as follow:

$$a^2 = (a_0^2 + va_1^2) + 2a_0a_1g = a_0^2 + a_1^2 + [(a_0+a_1)^2 - (a_0^2+a_1^2)]g. \tag{11}$$

Following [17], $a \in \mathbb{G}_{\phi_2(\mathbb{F}_{p^4})} \Rightarrow a_0^2 - va_1^2 = 1$. Thus, we can replace a_1^2 in the Eq. 11 by $\frac{1-a_0^2}{v}$ and compute a^2 with the following formula:

$$a^2 = 2a_0^2 - 1 + [(a_0+b_0)^2 - a_0^2 - (a_0^2-1)/v]g. \tag{12}$$

If $\frac{1}{v}$ can be computed without a modular inverse, then the cost of the above formula is 2 squares in $\mathbb{F}_{p^4}$ and some additions.

However, this is not the case with the GMT8 curve as $\frac{1}{v} = \frac{v^3}{5}$. We can precompute this value but it increases the cost of Eq. 11 by at least a multiplication of a $\mathbb{F}_{p^4}$ element by a $\mathbb{F}_p$ element (which costs four M_p). To avoid these multiplications, we propose to replace a_0^2 by $1+va_1^2$ in Eq. 11. It leads to the following formula which does not require any inversion:

$$a^2 = 2va_1^2 + 1 + [(a_0+a_1)^2 - 1 - (v+1)a_1^2]g. \tag{13}$$

Thus our formula is more suitable to compute squaring in the cyclotomic subgroup for pairing on the GMT8 curve. [4]

Summary of operations required by final exponentiation. Analogously to the computation of Miller's algorithm, we summarize the different operations required during the final exponentiation in Fig. 3. Basic operations are the same as in Miller's algorithm but there are other operations such as efficient squaring in the cyclotomic subgroups.

Modular operations form the basis of pairing arithmetic. The following section, presents a lightweight coprocessor suitable to compute pairings on different curves and at different security levels.

[4] At the time of submitting this article, the proposed formula was new in the literature. However, we later realized that it also appears in the RELIC project [1].

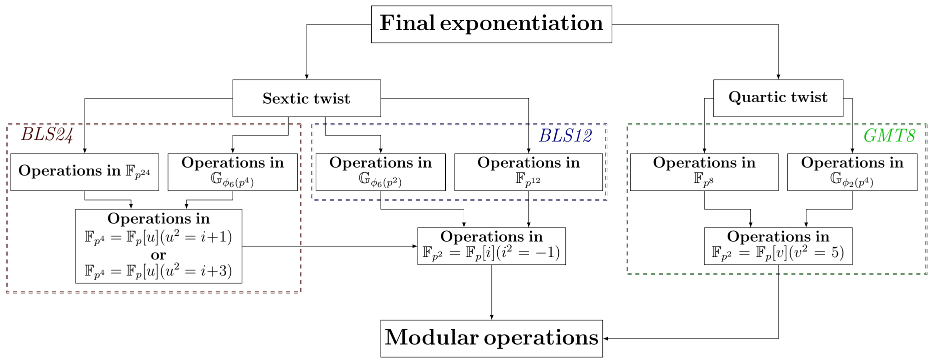
Final exponentiation
Sextic twist
Quartic twist
BLS24
BLS12
GMT8
Operations in $\mathbb{F}_{p^{24}}$
Operations in $\mathbb{G}_{\phi_6(p^4)}$
Operations in $\mathbb{G}_{\phi_6(p^2)}$
Operations in $\mathbb{F}_{p^{12}}$
Operations in $\mathbb{F}_{p^8}$
Operations in $\mathbb{G}_{\phi_2(p^4)}$
Operations in $\mathbb{F}_{p^4} = \mathbb{F}_p[u](u^2 = i+1)$ or $\mathbb{F}_{p^4} = \mathbb{F}_p[u](u^2 = i+3)$
Operations in $\mathbb{F}_{p^2} = \mathbb{F}_p[i](i^2 = -1)$
Operations in $\mathbb{F}_{p^2} = \mathbb{F}_p[v](v^2 = 5)$
Modular operations

Fig. 3: Operations during final exponentiation on selected curves

4 Hardware implementation of pairings

In this section, we present a lightweight coprocessor design to accelerate modular operations and cut down the additional cost brought by neglected operations. This coprocessor will be called the base field unit in the rest of the paper. Then, we propose a hardware/software co-design architecture to compare pairings on different curves at updated security levels.

4.1 Base field unit

As previously explained, the computation of pairings relies both on elliptic curves and extension fields arithmetic. These arithmetics can be carried out ith sequences of operations in the base field $\mathbb{F}_p$ (also called modular operations). The required operations are modular multiplications, reductions, additions, subtractions, doubles, and divisions by 2. These operations are computed with the base field unit. To limit the number of memory accesses, we implement elementary operations on 64-bit integers. Finally, we choose to use single-port RAM to store intermadiate values. This memory model is more likely to be suitable for light use because its cost is lower compared to a dual port RAM.

Modular multiplication. Given its complexity compared to other operations over the base field, modular multiplication is a key operation. The proposed multiplier, described in Fig. 4, is a variant of the systolic architecture proposed in [21]. It computes an alternative form of the Montgomery algorithm [31]: the Multiple Word Radix-2 Montgomery Multiplication (MWR2-MM, see Algorithm [33]). This architecture is composed of e processing elementary units. One unit (PE_0 on Fig. 4) focuses on the computation of line 3 and the first iteration at line 5 of Algorithm 4 (see below). Then, $e-1$ units (PE_j on Fig. 4) compute other iterations j at line 5.

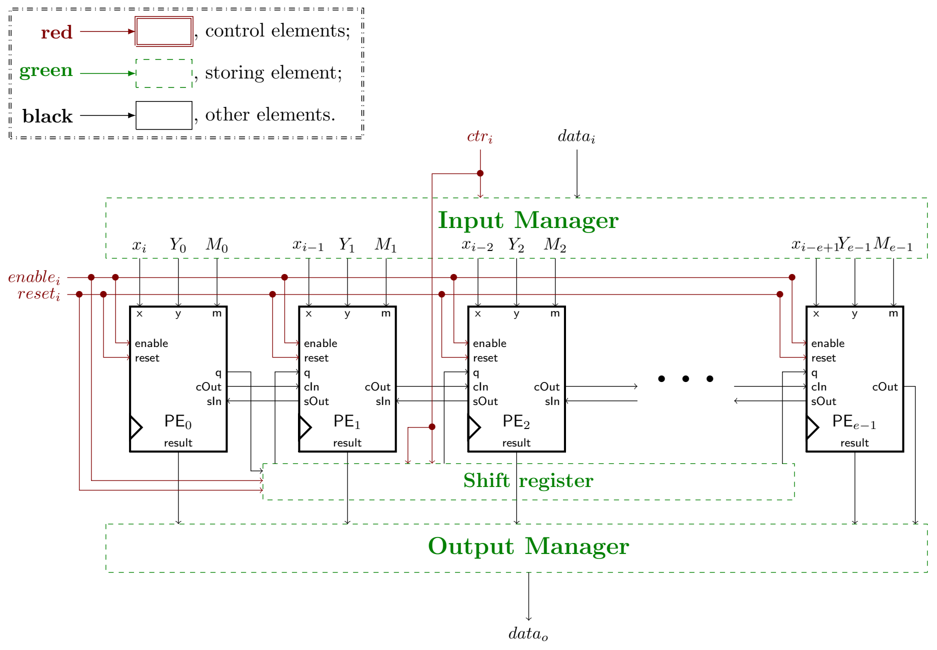

Fig. 4: Hardware design dedicated to compute Modular multiplications

These units compute bitwise and logical operations on 64-bit integers such as shift, addition or xor. Therefore it does not require any DSP in FPGA implementation. This design offers a good performance area trade-off. It distributes the calculation over several small processing elements. Hence, the size of the modulo has a limited impact on hardware frequency. All modular operations are decomposed into 64-bit additions, subtractions or shifts.

Algorithm 4 MWR2-MM [33]

```
Input: $X = \sum_{i=0}^{n-1} x_i.2^i$, $Y = \sum_{i=0}^{e-1} Y^{(j)}.2^{(w.j)}$, $p = \sum_{j=0}^{e-1} p^{(j)}.2^{(w.j)}$
Output: $S = \sum_{j=0}^{e-1} S^j.2^{w.j} = X.Y.2^{-n} \mod (n)$ with $0 \leq S \leq 2.p$
Begin
 1: $S = 0$
 2: for $i = 0, ..., n-1$ do
 3:     $q_i = (x_i.Y_0^{(0)}) \oplus S_0^{(0)}$
 4:     $(C^{(1)}, S^{(0)}) = x_i.Y^{(0)} + q_i.p^{(0)} + S^{(0)}$
 5:     for $j = 1, ..., e$ do
 6:         $(C^{(j+1)}, S^{(j)}) = C^{(j)} + x_i.Y^{(j)} + q_i.p^{(j)} + S^{(j)}$
 7:         $S^{(j-1)} = (S_0^{(j)}, S_{w-1..1}^{j-1})$
 8:     end for
 9:     $S^{(e)} = 0$
10: end for
11: return $S$
```

Additional operations. The dedicated component presented in Fig. 5 is designed to compute additional operations (addition, subtraction, double, division by 2 and reduction). As for modular multiplications, the modulo is loaded once prior to any computation. It is stored on a cyclic register (**CyclReg** on Fig. 5). Two 64-bit adder-subtractors (**Add** and **Red**) are used. One for the addition (or subtraction) and the other to compute the modular reduction. The two possible results are stored in two dedicated registers **RegAdd** and **RegRed**. These registers allow us to chain operations. Thus, we restrict the quantity of memory access to the minimum: load operands and store the final result. Then, the component **Div** is used to compute divisions by 2. The computation of a modular double is considered as a computation of a special modular addition. In this case, the loading of the operand is faster than in classical addition and this enables computing modular doubles faster than modular additions.

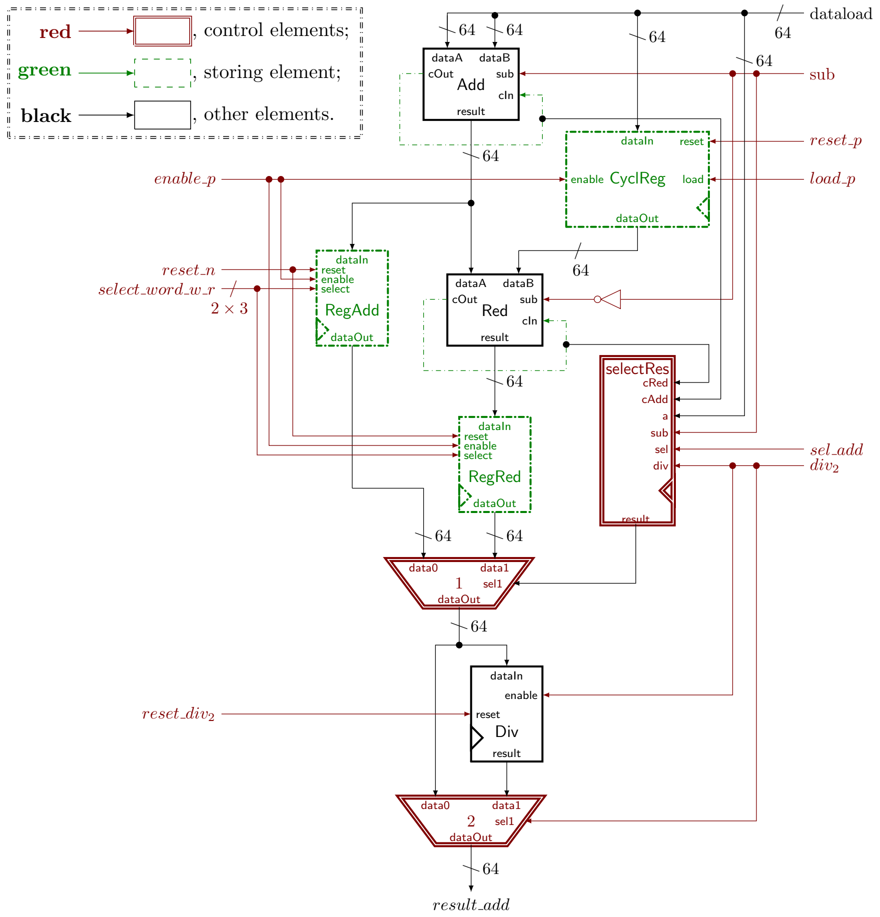
red, control elements;
green, storing element;
black, other elements.
dataload
64
sub
reset_p
load_p
enable_p
reset_n
select_word_w_r
2 × 3
dataA dataB cOut Add sub cIn result
dataIn reset enable CyclReg load dataOut
dataIn reset enable select RegAdd dataOut
dataA dataB cOut Red sub cIn result
selectRes cRed cAdd a sub sel div result
dataIn reset enable select RegRed dataOut
sel_add
div_2
data0 data1 1 sel1 dataOut
dataIn enable reset Div result
reset_div_2
data0 data1 2 sel1 dataOut
result_add

Fig. 5: Hardware design dedicated to compute additional operations

Similarly, the modular division by two is a special subtraction. Our adder computes both $A - 0$ and $A + p$, and then, depending on the parity of A, the results will either be $A/2$ or $(A+p)/2$. Once again, it enables computing modular divisions by two faster than modular subtractions. The number of clock cycles required by each operation in $\mathbb{F}_p$ is expressed in Table 3. The implementation of doubles and divisions by 2 operations allows saving e clock cycles for these operations. Implementation of additional operations (addition, double,...) has less impact on design performances than the implementation of modular multiplication. However, their costs can not be neglected since they are called about 6 times more than multiplications. As an example for GMT8 curves, when $\log_2(p) = 544$ and $e = 9$, the computation time of a modular multiplication is around 18 (resp. 25) times longer than a modular addition (resp. double). Additional operations represent 25% of pairing computation times which is significant.

Table 3: Costs of base field operations

Operation	**Number of clock cycles**
Modular multiplication	$n + 3\lceil n/64 \rceil + 4$
Modular reduction	$2\lceil n/64 \rceil + 6$
Modular addition/subtraction	$3\lceil n/64 \rceil + 6$
Modular double	$2\lceil n/64 \rceil + 6$
Modular division by 2	$2\lceil n/64 \rceil + 7$

Proposed hardware/software architecture for prototyping purposes. The multiplier and the custom adder are both controlled by a Finite State Machine (Scheduler in Fig. 6). This scheduler allows loading operands from the dedicated RAM (CryptoRAM in Fig. 6), launching a modular operation, and saving the final result into the CryptoRAM. The sequence of modular operations can be fixed for a specific pairing implementation, but we choose to maintain flexibility in our design and use a CPU instead. It allows implementing different pairings on the same component. Macro instructions are defined to pilot our base field unit which is connected to the CPU with an AXI interface (Advanced eXtensible Interface). This interface is chosen for its compatibility with a wide variety of processors (including ARM and RISC-V CPUs). To execute complex operations, the CPU must control the base field unit and the cryptoRAM, again to load operands, compute selected operations and save the results.

The chosen CPU for prototyping purposes is the MicroBlaze unit provided by Vivado tools. It is based on a 32-bit architecture. However, it can be easily replaced by any processor. 32-bit instructions of the form: $ins = \{@A, @B, @Z, Code\}$ are used by the CPU to pilot the base field unit. Having 32-bit CPU does not hinder the control of our 64-bit base field unit.

As shown in Fig. 6, instructions are sent by the CPU to the base field unit through the AXI. A decoder (Ins-decoder) stores addresses of perands ($@A, @B$), address of the result ($@Z$), and selects the base field operation corresponding to $Code$. Then, the Scheduler controls the CryptoRAM and either the multiplier or the custom adder computes this operation. In this way, a modular operation can be launched with a single instruction.

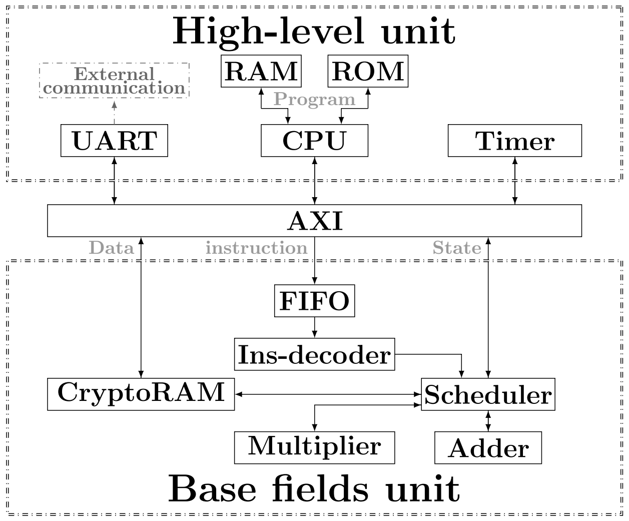

Fig. 6: Proposed hardware/software architecture

Finally, a FIFO stacks several instructions to mask the cost of sending instructions to the base field unit. The CPU can read and write into the CryptoRAM to load operands and to read the result of the computation. Instructions are sent from the CPU to the coprocessor through the AXI as shown in Fig. 6. The CPU also controls a timer to monitor coprocessor computation times and a Universal Asynchronous Receiver Transmitter (UART) component to allow external communication.

Finally, the MicroBlaze CPU runs a software program in C to compute High-level operations. The objective of this design is to minimize the impact of additional operations on implementation performances. Thus, the base field unit is built to be able to launch additional operations while performing modular multiplications.

Common optimization: parallelized multiplication and square in $\mathbb{F}_{p^2}$. The selected curves all rely on $\mathbb{F}_{p^2} = \mathbb{F}_p[v]$ arithmetic. In Section 3, $\mathbb{F}_{p^2} = \mathbb{F}_p[v]$, where v is defined as $v = -1$ for BLS12 and BLS24 curves and $v = 5$ for GMT8 curves. Multiplication by 5 can be computed with two doubles and one addition. Then, multiplications by v can also be parallelized in Algorithm 2 and Algorithm 3. Four additions and one multiplication by v can be computed in parallel during multiplications and three additions, one double and two multiplications by v during squaring.

Additional operations that can be computed in parallel are written with **red** letters in Algorithm 5 and Algorithm 6. Optimized multiplications and squares in $\mathbb{F}_{p^2}$ cut the computation time by 10% with our architecture.

We manage to parallelize more additional operations during multiplications and squares in $\mathbb{F}_{p^4}$. The total gain brought by our parallel implementation is approximately 12% for BLS12 and 15% for GMT8 and BLS24 curves.

Algorithm 5 Optimized multiplication in $\mathbb{F}_{p^2} = \mathbb{F}_p[g], g^2 = v, v \in \mathbb{F}_p$

```
Input: $A = a_0 + a_1 g, B = b_0 + b_1 g \in \mathbb{F}_{p^2}$
Output: $Z \leftarrow AB \in \mathbb{F}_{p^2}$
Cost: $3M_{p^2} + 1A_{p^2}$
Begin
  1: $t_0 \leftarrow a_0 b_0$; $\mathbf{t_1 \leftarrow a_0 + a_1}$ $\mathbf{z_0 \leftarrow b_0 + b_1}$;
  2: $z_1 \leftarrow a_1 b_1$; $t_1 \leftarrow t_1 z_0$; $\mathbf{z_0 \leftarrow t_0 - v z_1}$;
  3: $\mathbf{z_1 \leftarrow t_0 + z_1}$; $z_1 \leftarrow z_1 - t_1$;
  4: return $Z = z_0 + z_1 g$;
End
```

Algorithm 6 Optimized square in $\mathbb{F}_{p^2} = \mathbb{F}_p[g], g^2 = v, v \in \mathbb{F}_p$

```
Input: $A = a_0 + a_1 g \in \mathbb{F}_{p^2}$,
Output: $Z \leftarrow A^2 \in \mathbb{F}_{p^2}$
Cost: $2M_p + 1A_p$
Begin
  1: $z_0 \leftarrow a_0 a_1$; $\mathbf{t_0 \leftarrow a_0 - a_1}$;
  2: $\mathbf{t_1 \leftarrow a_0 - v a_1}$; $t_0 \leftarrow t_0 t_1$;
  3: $\mathbf{z_1 \leftarrow 2 z_0}$; $z_0 \leftarrow t_0 + \mathbf{(v+1) z_0}$;
  4: return $Z = z_0 + z_1 g$;
End
```

Verification and test. The procedure presented in Fig. 7 is used to ensure the correctness of our implementation. First, we use Magma calculator software [10] as a reference implementation to generate P (resp. Q), a generator of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$). Then, a Python script computes the test vector $\{P, [\alpha]P, Q, [\alpha]Q\}$ with α a random element. Subsequently, our design computes both $e(P, [\alpha]Q)$ and $e([\alpha]P, Q)$. Finally, the CPU sends these two values back to the desktop which checks for the equality of: $e(P, [\alpha]Q) = e([\alpha]P, Q)$. Verifying the pairing bilinearity ensures the correctness of our implementations. This method is used to test our design at each level of the development, from operations on the base field to the entire pairing.

4.2 Implementation results

The proposed design is coded in VHDL and implemented on a Kintex-7. Our base field unit is packaged into a custom IP and integrated into a System on Chip with a MicroBlaze CPU. The *time*×*area* metric is chosen to estimate the overall performance of the hardware component. This value gives a complete picture since it does not only take into account the estimated area but also the amount of performance provided. It gives a fairer comparison than the classical one using the equivalent gate metrics. The obtained performances are presented in Table 4 for the old 128-bit and for the updated 128-bit and 192-bit security levels.

In [35], authors build a highly parallel architecture to design a fast and energy-efficient pairing unit for BN curves at the old 128-bit security level. However, their design required thousands of slices and several Digital Signal Processing (DSP) units and would be difficult to fit in lightweight designs.

To our knowledge, the fastest pairing architecture at the old 128-bit security level is the one proposed in [37]. To maximize the benefit of parallelization, the authors use several triple-port RAMs. This memory requires arround four times the area required by classical simple-port ones. As a result, this architecture could also barely fit in lightweight designs.

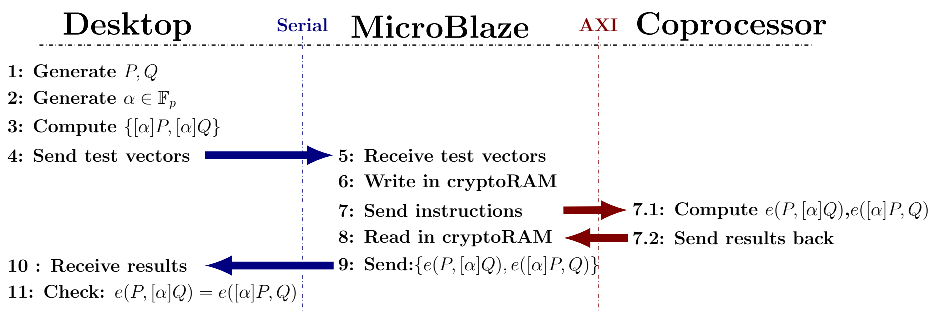
Desktop
Serial
MicroBlaze
AXI
Coprocessor
1: Generate P, Q
2: Generate $\alpha \in \mathbb{F}_p$
3: Compute $\{[\alpha]P, [\alpha]Q\}$
4: Send test vectors
5: Receive test vectors
6: Write in cryptoRAM
7: Send instructions
7.1: Compute $e(P, [\alpha]Q)$,$e([\alpha]P, Q)$
8: Read in cryptoRAM
7.2: Send results back
10 : Receive results
9: Send:$\{e(P, [\alpha]Q), e([\alpha]P, Q)\}$
11: Check: $e(P, [\alpha]Q) = e([\alpha]P, Q)$

Fig. 7: Procedure for a test sequence with our ALU

Authors in [32] propose a lightweight hardware accelerator for modular multiplication and pilot it with a Cortex A9 CPU to compute pairings on BN curves at the old 128-bit security level. They also demonstrate that adding dedicated hardware to compute operations in the base field can decrease both time and energy required to compute a pairing. Our BLS12-381 implementation has a better *time*×*area* than previous hardware implementations on BN curves of [32,35].

Table 4: Performances comparison of pairing hardware implementations

Security	**Ref.**	**Curves**	**Platform**	**Area(slices)**	**DSP**	**Time(ms)**	***time*×*area***
99.7-bit	[37]	BN-254	Virtex-6	5237	64	**0.41**	**2147**
	[35]		Virtex-7	28400	128	3.43	97412
	[32]		Zynq-7020	598	0	134	80132
120.7-bit	This work	BLS12-381	Kintex-7	1006	0	36.14	**36357**
128-bit	This work	BLS12-460	**Kintex-7**	1223	0	48.91	59817
			Virtex-7	1235	0	48.91	60404
			Virtex-6	1446	0	65.21	94294
		BLS24-318	**Kintex-7**	**925**	0	64.78	59922
			Virtex-7	922	0	64.78	**59727**
			Virtex-6	1156	0	86.37	99844
		GMT8-544	**Kintex-7**	1325	0	**42.71**	**56591**
			Virtex-7	1463	0	42.71	62485
			Virtex-6	1654	0	56.94	94195
192-bit	This work	BLS24-518	**Kintex-7**	1325	0	184.23	**244105**

For the sake of fair comparison with future work, we implement our ALU on a Virtex-7 and a Virtex-6 FPGA. These platforms are chosen because they have been widely used in previous work. FPGAs are built with Configurable Logic Blocks (CLB), and 7-series FPGA such as Kintex-7 or Virtex-7, and Virtex-6 FPGA use identical CLB. Each CLB contains two slices and each slice contains four 6-inputs Look Up Table (LUT) and four flip-flops. According to Xilinx, the main difference is that the 7-series FPGAs have more interconnecting routing resources compared to Virtex-6 FPGA. This explains the difference between our design performance on the 7-series and on the Virtex-6 FPGA. The last version of the Xilinx ISE design suite (14.7) is used to implement our coprocessor in Virtex-6. The results presented in Table 4 are given as an indication since the test and development are done on a Kintex-7 FPGA. Virtex-7 implementation is running at 200 MHz and Virtex-6 at 150 MHz. The difference in *time*×*area* between GMT8 and BLS24 is limited to 5% which is not enough to discard a curve. Furthermore, pairings at actual security levels require a consequent amount of memory. With our implementation, GMT8 requires 5976kB, BLS12 7040kB and BLS24 9920kB respectively. Since memory is often the critical resource in constrained devices, GMT8 may be the appropriate choice at the updated 128-bit security level.

Table 4 also shows that the curve ranking in terms of *time*×*area*, not only differs from estimations given in [4], but also differs from one platform to another. Therefore, it becomes interesting to choose the curve according to the desired application and platform.

5 Conclusion and future work

The Kim and Barbulescu attack created a new paradigm as BN curves are no longer undisputed pairing champions. Authors of [4] and [18] have studied new curves. The consequences are that any dedicated hardware implementation ought to support multiple arithmetics to maximize flexibility. Pairing complexity estimates only take into account the complexity of modular multiplications. This paper shows that neglected operations such as modular additions have also a significant impact on implementation performances since they represent 25% of the overall computation time in our architecture. Based on the best curve candidates for implementations, we presented a flexible hardware architecture to support all of them. The proposed lightweight hardware is designed to accelerate modular operations. It has a reconfigurable modulus which enables the support of different curves and allows parallel computing during multiplication. This improves performance by approximately 15% decreasing the cost of additional operations to consider only multiplications.

To the best of our knowledge, this paper presents the first hardware implementation of pairings at the updated 128-bit and 192-bit security levels as proposed in [4] and in [18]. Moreover, the proposed implementations provide promising performances compared to previous work on lightweight implementations since our *time*×*area* product is three times better than the one presented in [32]. Our different FPGA porting results also provide evidence that the best curves have similar complexity at the updated 128-bit security level. It shows that there is no optimal choice of pairings at the 128-bit security level. With our architecture, the GMT8 curve seems to provide the best time and *time*×*area* performances. But it also requires a bigger coprocessor than for BLS24 or BLS12. On the other hand, BLS24 requires much more memory than GMT8. Future work could consider other promising curves such as KSS16, DCC15 or other curves proposed in [18] and evaluate their performance on classical protocols.

References

1. Aranha, D.F., Gouvêa, C.P.L., Markmann, T., Wahby, R.S., Liao, K.: RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`
2. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López-Hernández, J.C.: Faster explicit formulas for computing pairings over ordinary curves. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 6632, pp. 48–68. Springer (2011)
3. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Journal of Cryptology (2018), `https://hal.archives-ouvertes.fr/hal-01534101`
4. Barbulescu, R., El Mrabet, N., Ghammam, L.: A taxonomy of pairings, their security, their complexity. IACR Cryptol. ePrint Arch. **2019**, 485 (2019)
5. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: SCN. Lecture Notes in Computer Science, vol. 2576, pp. 257–267. Springer (2002)
6. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3897, pp. 319–331. Springer (2005)
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: USENIX Security Symposium. pp. 781–796. USENIX Association (2014)
8. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) Advances in Cryptology — CRYPTO 2001. pp. 213–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) Advances in Cryptology — ASIACRYPT 2001. pp. 514–532. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
10. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. **24**(3-4), 235–265 (1997). https://doi.org/10.1006/jsco.1996.0125, `http://dx.doi.org/10.1006/jsco.1996.0125`, computational algebra and number theory (London, 1993)
11. Cocks, C., Pinch, R.: Identity-based cryptosystems based on the Weil pairing (2001)
12. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 6056, pp. 224–242. Springer (2010)
13. Duan, P., Cui, S., Chan, C.: Special polynomial families for generating more suitable elliptic curves for pairing-based cryptosystems. IACR Cryptology ePrint Archive **2005**, 342 (01 2005)
14. El Mrabet, N., Guillermin, N., Ionica, S.: A study of pairing computation for elliptic curves with embedding degree 15. IACR Cryptol. ePrint Arch. **2009**, 370 (2009)
15. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. J. Cryptol. **23**(2), 224–280 (2010)
16. Ghammam, L., Fouotsa, E.: Improving the computation of the optimal Ate pairing for a high security level. Journal of Applied Mathematics and Computing **59** (02 2018). https://doi.org/10.1007/s12190-018-1167-y
17. Granger, R., Scott, M.: Faster squaring in the cyclotomic subgroup of sixth degree extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography – PKC 2010. pp. 209–223. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
18. Guillevic, A., Masson, S., Thomé, E.: Cocks-Pinch curves of embedding degrees five to eight and optimal Ate pairing computation. Cryptology ePrint Archive, Report 2019/431 (2019), `https://eprint.iacr.org/2019/431`

19. Hess, F., Smart, N., Vercauteren, F.: The Eta pairing revisited. Cryptology ePrint Archive, Report 2006/110 (2006), `https://eprint.iacr.org/2006/110`
20. Hess, F.: Pairing lattices. Cryptology ePrint Archive, Report 2008/125 (2008)
21. Huang, M., Gaj, K., El-Ghazawi, T.: New hardware architectures for Montgomery modular multiplication algorithm. IEEE Transactions on Computers **60**(7), 923–936 (July 2011). https://doi.org/10.1109/TC.2010.247
22. John, T.: Duality theorems in Galois cohomology over number fields. International Congress of Mathematicians Stockholm 1962, Djursholm (1963), computational algebra and number theory (London, 1993)
23. Joux, A.: A one round protocol for tripartite diffie-hellman. ANTS-IV : Proceedings of the 4th International Symposium on Algorithmic Number Theory, pages 385394, London, UK (2000)
24. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In: Pairing. Lecture Notes in Computer Science, vol. 5209, pp. 126–135. Springer (2008)
25. Karabina, K.: Squaring in cyclotomic subgroups. Cryptology ePrint Archive, Report 2010/542 (2010), `https://eprint.iacr.org/2010/542`
26. Khandaker, M.A.A., Nanjo, Y., Ghammam, L., Duquesne, S., Nogami, Y., Kodera, Y.: Efficient optimal Ate pairing at 128-bit security level. In: Patra, A., Smart, N.P. (eds.) Progress in Cryptology – INDOCRYPT 2017. pp. 186–205. Springer International Publishing, Cham (2017)
27. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 543–571. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
28. Knuth, D.E.: The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms. Addison Wesley Longman Publishing Co., Inc., USA (1997)
29. Koblitz, N., Menezes, A.: Pairing-based cryptography at high security levels. In: Smart, N.P. (ed.) Cryptography and Coding. pp. 13–36. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
30. Miller, V.S.: The Weil pairing, and its efficient calculation. J. Cryptol. **17**(4), 235–261 (Sep 2004). https://doi.org/10.1007/s00145-004-0315-8
31. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation **44**(170), 519–521 (1985)
32. Salman, A., Diehl, W., Kaps, J.: A light-weight hardware/software co-design for pairing-based cryptography with low power and energy consumption. In: 2017 International Conference on Field Programmable Technology (ICFPT). pp. 235–238 (2017). https://doi.org/10.1109/FPT.2017.8280149
33. Tenca, A., Koc, C.: A scalable architecture for Montgomery multiplication. In: Proc. First Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '99), pp. 94-108. pp. 94–108 (01 1999)
34. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory **56**(1), 455–461 (Jan 2010). https://doi.org/10.1109/TIT.2009.2034881
35. Wang, A.T., Guo, B.W., Wei, C.J.: Highly-parallel hardware implementation of optimal Ate pairing over Barreto-Naehrig curves. Integration **64**, 13 – 21 (2019)
36. Xiong, X., Wong, D., Deng, X.: Tinypairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks. pp. 1 – 6 (05 2010)
37. Yao, G.X., Fan, J., Cheung, R.C.C., Verbauwhede, I.: Faster pairing coprocessor architecture. In: Abdalla, M., Lange, T. (eds.) Pairing-Based Cryptography – Pairing 2012. pp. 160–176. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
38. Zhang, X., Lin, D.: Analysis of optimum pairing products at high security levels (12 2012). https://doi.org/10.1007/978-3-642-34931-7_24