

# NEON implementation of an attribute-based encryption scheme

Ana Helena Sánchez, Francisco Rodríguez-Henríquez\*

Computer Science Department, CINVESTAV-IPN, México  
asanchez@computacion.cs.cinvestav.mx, francisco@cs.cinvestav.mx

**Abstract.** In 2011, Waters presented a ciphertext-policy attribute-based encryption protocol that uses bilinear pairings to provide control access mechanisms, where the set of user’s attributes is specified by means of a linear secret sharing scheme. Some of the applications foreseen for this protocol lie in the context of mobile devices such as smartphones and tablets, which in a majority of instances are powered by an ARM processor supporting the NEON vector set of instructions. In this paper we present the design of a software cryptographic library that implements a 127-bit security level attribute-based encryption scheme over mobile devices equipped with a 1.4GHz Exynos 4 Cortex-A9 processor and a developing board that hosts a 1.7 GHz Exynos 5 Cortex-A15 processor. For the latter platform and taking advantage of the inherent parallelism of the NEON vector instructions, our library computes a single optimal pairing over a Barreto-Naehrig curve approximately 2 times faster than the best timings previously reported on ARM platforms at this level of security. Further, using a 6-attribute access formula our library is able to encrypt/decrypt a text/ciphertext in less than 4.49mS and 15.67mS, respectively.

**Key words:** Attribute Based-Encryption, Optimal-Ate pairing, Barreto-Naehrig curves, ARM processor, pairing-based protocols, elliptic curve scalar multiplication.

## 1 Introduction

It was long assumed that the task of computing a single bilinear pairing was rather expensive, so much so that when assessing the complexity of a given protocol, a designer could safely ignore the computational cost of all the other cryptographic components. Nevertheless, in the last few years we have witnessed a dramatic reduction in the timing required to calculate a single pairing, which has had the side effect that the computation of the other ancillary functions associated to pairing-based protocols have acquired a renewed importance. Some examples of these auxiliary blocks include, fixed/variable point scalar multiplication for elliptic curves defined over finite fields and their extensions, the

---

\* A portion of this work was performed while the author was visiting University of Waterloo.

projection of arbitrary strings to a random point in those elliptic curves, exponentiation in field extensions, *etc.* Furthermore, as pointed out in [8,20], several pairing-based protocols admit further optimizations, such as the computation of fixed-argument pairings and products of pairings.

Unfortunately as of today, very few works have analyzed in detail the complexity and overall computational weight of non-pairing cryptographic operations in a given protocol. This lack of research in the implementation of pairing-based protocols is especially acute for mobile platforms such as the ones using ARM processors.

An important number of major IT players such as Apple, Samsung, Sony, to name just a few, have adopted the ARM Cortex family of processors for powering their tablets, smartphones and other mobile devices. A majority of those devices support the vector set of instructions NEON [6]. In spite of their ever increasing popularity, it is only until recently that some research works have studied the implementation of cryptographic primitives over ARM processor platforms.

Among the research papers reporting pairing implementations in the ARM Cortex family of processors are [1] and [13]. In [1], the authors advocate the use of affine coordinates could be more attractive than the projective ones when implementing pairings in constrained devices, whereas the software library of [13] reports the current record in the computation of a single asymmetric pairing at the 128, 224 and 320-bit security levels. As for the implementation of pairing-based protocols on mobile devices, the only work that we are aware of is [2], where the authors described the design of an attribute-based encryption scheme able to preserve the confidentiality of the medical electronic records generated within a hospital environment.

In this work we present the design of a software library that implements Waters' attribute-based encryption scheme [21], over a set of mobile device platforms equipped with the latest models of the ARM Cortex family of processors and the vectorized set of instructions NEON. Our library was specifically tailored for computing optimal pairings over Barreto-Naehrig curves at the 127-bit security level. When executed on a developing board that hosts a 1.7 GHz Exynos 5 Cortex-A15 processor, our software computes a single optimal pairing in approximately 5.82M clock cycles, which is about two times less than the estimated cycling count reported in [13] for a single pairing computation over a TI 1.2GHz OMAP 4460 Cortex-A9 processor.

Our library also implements single/multi-pairing computations with fixed/-variable input points, as well as other auxiliary functions associated with most pairing-based protocols such as scalar multiplication and the projection of arbitrary strings to elliptic curve points defined over extension finite fields, among others. In particular, when executed on the Cortex-A15 processor mentioned above and when using an access formula composed of six attributes, our library computes the encryption/decryption primitives of Waters' attribute-based encryption protocol in less than 7.63M clock cycles and 26.64M clock cycles, roughly equivalent to 4.49mS and 15.67mS, respectively.

## 2 Mathematical background

Let  $p$  be a prime, and let  $E$  be an elliptic curve defined over the finite field  $\mathbb{F}_p$ . Let  $r$  be a prime with  $r \mid \#E(\mathbb{F}_p)$  and  $\gcd(r, p) = 1$ . The embedding degree  $k$  is the smallest positive integer such that  $r \mid (p^k - 1)$ . In this paper, only the Barreto-Naehrig (BN) pairing-friendly family of elliptic curves [5] were considered for pairing implementation. All BN curves have embedding degree  $k = 12$  and they are defined by the equation  $E : y^2 = x^3 + b, b \in \mathbb{F}_p^*$ , where the characteristic  $p$  of the prime field, the group order  $r$ , and the trace of Frobenius  $t$  are parametrized as,

$$\begin{aligned} p(z) &= 36z^4 + 36z^3 + 24z^2 + 6z + 1; \\ r(z) &= 36z^4 + 36z^3 + 18z^2 + 6z + 1; \\ t(z) &= 6z^2 + 1, \end{aligned} \tag{1}$$

where  $z \in \mathbb{Z}$  is an arbitrary integer known as the BN parameter, and where  $p(z)$  and  $r(z)$  are prime numbers. BN curves admit a sextic twist curve, defined as  $\tilde{E}(\mathbb{F}_{p^2}) : Y^2 = X^3 + b/\xi$ , where  $\xi \in \mathbb{F}_{p^2}$  is neither a square nor a cube in  $\mathbb{F}_{p^2}$ .

Let  $\pi : (x, y) \mapsto (x^p, y^p)$  be the  $p$ -th power Frobenius endomorphism. The trace of the Frobenius is defined as  $t = p + 1 - \#E(\mathbb{F}_p)$ . Let  $\mathbb{G}_1 = \{P \in E[r] : \pi(P) = P\} = E(\mathbb{F}_p)[r]$ , where  $\mathbb{G}_1$  is the 1-eigenspace of  $\pi$  acting on  $E[r]$ . Let  $\Psi : \tilde{E} \rightarrow E$  be the associated twisting isomorphism. Let  $\tilde{Q} \in \tilde{E}(\mathbb{F}_{p^2})$  be a point of order  $r$ ; then  $Q = \Psi(\tilde{Q}) \notin E(\mathbb{F}_p)$ . The group  $\mathbb{G}_2 = \langle Q \rangle$  is the  $p$ -eigenspace of  $\pi$  acting on  $E[r]$ . Let  $\mathbb{G}_T$  denote the order- $r$  subgroup of  $\mathbb{F}_{p^{12}}^*$  [3]. The bilinear pairing studied in this paper is defined as the non-degenerate map  $\hat{a}_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ , corresponding to the optimal *ate* pairing defined as:

$$\begin{aligned} \hat{a}_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 &\rightarrow \mathbb{G}_T \\ (Q, P) &\mapsto [f_{s,Q}(P) \cdot \ell_{(s)Q, \pi(Q)}(P) \cdot \\ &\quad \ell_{(s)Q + \pi(Q), \pi^2(Q)}(P)]^{(p^{12}-1)/r} \end{aligned} \tag{2}$$

where  $f_{s,Q}$  is a Miller function of length  $s = 6z + 2$ , which is a rational function in  $\bar{\mathbb{F}}_p(E)$  with divisor  $\text{div}(f_{s,R}) = s[R] - [sR] - (s-1)[\mathcal{O}]$ , while  $\ell_{Q_1, Q_2}$  is the line equation given by the point addition of  $Q_1 \in \mathbb{G}_2$  and  $Q_2 \in \mathbb{G}_2$ . Algorithm 1 computes the optimal pairing as defined in Eq. (2).

## 3 Tower Extension Field Arithmetic

Efficient arithmetic over extension finite fields is a necessary requirement in the development of high-performance pairing-based schemes. In this work, we represent  $\mathbb{F}_{p^{12}}$  using the same tower extension employed in [4], namely, we first construct a quadratic extension, which is followed by a quadratic and cubic extensions of it and finally by a quadratic one, using the following irreducible binomials,

---

**Algorithm 1** Optimal ate pairing

---

**Require:**  $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$   
**Ensure:**  $g = \tilde{a}_{opt}(Q, P)$

1: Write $s = 6z+2$ as $s = \sum_{i=0}^{l-1} s_i, s_i \in \{-1, 0, 1\}$	8: $f \leftarrow f \cdot \ell_{T,-Q}(P), T \leftarrow T - Q$
2: $T \leftarrow Q, f \leftarrow 1$	9: <b>end if</b>
3: <b>for</b> $i = l-2 \rightarrow 0$ <b>do</b>	10: <b>end for</b>
4: $f \leftarrow f^2 \cdot \ell_{T,T}(P), T \leftarrow 2T$	11: $Q_1 \leftarrow \pi(Q), Q_2 \leftarrow \pi^2(Q)$
5: <b>if</b> $s_i = 1$ <b>then</b>	12: $f \leftarrow f \cdot \ell_{T,Q_1}(P), T \leftarrow T + Q_1, f \leftarrow$ $f \cdot \ell_{T,-Q_2}(P), T \leftarrow T - Q_2$
6: $f \leftarrow f \cdot \ell_{T,Q}(P), T \leftarrow T + Q$	13: $g \leftarrow f^{(p^{12}-1)/r}$
7: <b>else if</b> $s_i = -1$ <b>then</b>	14: <b>return</b> $g$

---

- $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta)$ , where  $\beta = -1$
- $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[V]/(V^2 - \xi)$ , where  $\xi = u + 1$
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[V]/(V^3 - \xi)$ , where  $\xi = u + 1$
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[W]/(W^2 - V)$  or  $\mathbb{F}_{p^4}[T]/(T^3 - V)$

Furthermore, as in [4] we selected  $z = -(2^{62} + 2^{55} + 1)$ , which using Eq. 1 yields a prime  $p \equiv 3 \pmod{4}$ . This allows for a faster arithmetic over  $\mathbb{F}_{p^2}$ , since the multiplication by the constant  $\beta$  reduces to a simple subtraction. For the computation of the pairing final exponentiation, cyclotomic subgroup arithmetic  $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$  [12] was extensively used.

---

**Algorithm 2** Montgomery product

---

**Require:** prime  $p, p', r = 2^k$  and  $\tilde{a}, \tilde{b} \in \mathbb{F}_p$   
**Ensure:**  $\tilde{c} = \text{MontPr}(\tilde{a}, \tilde{b})$

- 1:  $t \leftarrow \tilde{a} \cdot \tilde{b}$
- 2:  $u \leftarrow (t + (t \cdot p' \bmod r) \cdot p) / r$
- 3: **if**  $u > p$  **then**
- 4: **return**  $u - p$
- 5: **else**
- 6: **return**  $u$
- 7: **end if**

---

### 3.1 Field multiplication over $\mathbb{F}_p$

The single most important base field arithmetic operation is modular multiplication, which is defined as  $c = a \cdot b \bmod p$ , with  $a, b, c \in \mathbb{F}_p$ . Since in general BN primes are not suitable for fast reductions, this operation was performed via the Montgomery multiplication algorithm. The Montgomery product is defined as,  $\tilde{c} = \tilde{a} \cdot \tilde{b} \cdot r^{-1} \bmod p$ , where  $\tilde{a}, \tilde{b} \in \mathbb{F}_p$  are given as,  $\tilde{a} = a \cdot r \bmod p$  and  $\tilde{b} = b \cdot r \bmod p$ , respectively. This formulation allows trading the costly division by  $p$  with divisions by  $r$ , where  $r = 2^k$  with  $k-1 < |p| < k$ . If required, the modular product  $c$ , can be easily recovered using  $c = \tilde{c} \cdot r^{-1} \bmod p$ . Algorithm 2 shows the classical version of the Montgomery product, which expects as input parameter an integer  $p'$  that can be precomputed before hand using Bezout's identity for two co-prime integers, namely,  $r \cdot r^{-1} - p \cdot p' = 1$ .

In our library both, the Separated Operand Scanning (SOS) and the Coarsely Integrated Operand Scanning (CIOS) multi-precision Montgomery product variants as described in [17] were implemented. The SOS method computes first the integer product  $t = a \cdot b$ , followed by the Montgomery reduction step that calculates  $u$  such that  $u = (t + m \cdot p)/r$ , where  $m = t \cdot p' \pmod{r}$ . In this case  $r = 2^{\omega n}$ , where  $\omega$  is the wordsize of the processor and  $n = \lceil (\log_2 p) / \omega \rceil$ . The CIOS method interleaves the calculations corresponding to the integer product with the ones required for getting  $u$ . Since in both methods, the reduction is implemented word by word, then the operation  $m = t \cdot p' \pmod{r}$  can be performed replacing  $p'$  by  $p'_0 = p' \pmod{2^\omega}$ , which redounds in a more efficient computation. According to [17], the CIOS variant is more efficient than the SOS one. Nevertheless, the later method allows lazy reduction, which was the reason why we implemented both variants.

### 3.2 Extension field arithmetic computational cost

Denote by  $(a, m, s, i)$  and  $(\tilde{a}, \tilde{m}, \tilde{s}, \tilde{i})$  the computational cost of the addition, multiplication, squaring and inversion operations over  $\mathbb{F}_p$  and  $\mathbb{F}_{p^2}$ , respectively. The field arithmetic procedures used in this work extensively exploits lazy reduction, which closely resembles the approaches adopted in [4, 13]. Let  $\tilde{m}_E, \tilde{s}_E$  and  $\tilde{r}_E$  denote integer multiplication, integer squaring and reduction over  $\mathbb{F}_{p^2}$ , respectively, where  $\tilde{m} = \tilde{m}_E + \tilde{r}_E$  and  $\tilde{s} = \tilde{s}_E + \tilde{r}_E$ . The rationale behind the costs given in Table 1, can be summarized as follows.

The cost of reductions over  $\mathbb{F}_{p^2}$  is twice the cost of reduction over  $\mathbb{F}_p$ , *i.e.*,  $\tilde{r}_E = 2r_E$ . At a field extension  $\mathbb{F}_{p^d}$ ,  $d = 2^i 3^j$ ,  $i, j \in \mathbb{Z}^+$ ; the product  $c = a \cdot b$ ,  $a, b, c \in \mathbb{F}_{p^d}$  can be computed with  $3^i 6^j$  integer multiplications and  $2^i 3^j$  reductions modulo  $p$  (Theorem 1 of [4]). Field inversion was based on the procedure described in [13]. In the case of the quadratic and twelfth field extensions  $\mathbb{F}_{p^2}$  and  $\mathbb{F}_{p^{12}}$ , field squaring was computed using the complex method at a cost of 2 multiplications. The inversion of an element  $A = a_0 + a_1 u \in \mathbb{F}_{p^2}$  was obtained through the identity  $(a_0 + a_1 u)^{-1} = (a_0 - a_1 u)/(a_0^2 - \beta a_1^2)$ . In  $\mathbb{F}_{p^4}$  the squaring was implemented at a cost of  $3\tilde{s}$ . This operation is required for computing squarings in the cyclotomic group  $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ , having a cost of 3 squarings over  $\mathbb{F}_{p^4}$ . The asymmetric squaring formula for cubic extensions of [7] was used in the field  $\mathbb{F}_{p^6}$  at a cost of  $2\tilde{m} + 3\tilde{s}$ . Inversion in  $\mathbb{F}_{p^6}$  has a computational complexity of  $9\tilde{m} + 3\tilde{s} + i$  [15]. Notice that  $m_\xi$  stands for a multiplication by the constant  $\xi$ .

### 3.3 Field arithmetic implementation using NEON

The performance of a field arithmetic library is strongly influenced by the processor micro-architecture features, the size of the operands and the algorithms and programming techniques associated to them. In our case, the size of the operands is of 254 bits, which conveniently allows the usage of lazy reduction. The word size in the ARM processors is of 32 bits and the processors considered in this work include the NEON vector set of instructions.

Field	Addition	Multiplication	Squaring	Inversion
$\mathbb{F}_{p^2}$	$\tilde{a} = 2a$	$\tilde{m} = 3m_E + 2r_E + 8a + m_\beta$	$\tilde{s} = 2m_E + 2r_E + 3a$	$\tilde{i} = 2m_E + r_E + 2m + 2a + i$
$\mathbb{F}_{p^4}$	$2\tilde{a}$		$3\tilde{s} + m_\xi + 4\tilde{a}$	
$\mathbb{F}_{p^6}$	$3\tilde{a}$	$6\tilde{m}_E + 3\tilde{r}_E + 2m_\xi + 24\tilde{a}$	$2\tilde{m} + 3\tilde{s} + 2m_\xi + 9\tilde{a}$	$9\tilde{m}_E + 3\tilde{s}_E + 7\tilde{r}_E + 4m_\xi + 10\tilde{a} + \tilde{i}$
$\mathbb{F}_{p^{12}}$	$6\tilde{a}$	$18\tilde{m}_E + 6\tilde{r}_E + 7m_\xi + 96\tilde{a}$	$12\tilde{m}_E + 6\tilde{r}_E + 6m_\xi + 63\tilde{a}$	$25\tilde{m}_E + 9\tilde{s}_E + 16\tilde{r}_E + 13m_\xi + 79\tilde{a} + \tilde{i}$
$\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$			$9\tilde{s} + 4m_\xi + 30\tilde{a}$	$3\tilde{a}$

**Table 1.** Computational cost of the tower extension field arithmetic

NEON is a 128-bit Single Instruction Multiple Data (SIMD) architecture extension for the ARM Cortex family of processors. NEON architecture has 32 registers of 64 bits (*doubleword*), which can be viewed as 16 registers of 128 bits (*quadword*). Our library mostly made use of two intrinsic instructions:

```
uint64x2_t vmull_u32 (uint32x2_t, uint32x2_t);
uint64x2_t vmlal_u32 (uint64x2_t, uint32x2_t, uint32x2_t).
```

The first one performs two 32-bit integer multiplications storing the corresponding result into two 64-bit registers. The second one performs a multiplication that is accumulated with the addition of a 64-bit scalar. Algorithm 3 illustrates the usage of NEON for computing the double integer product  $F = a \cdot b$ ,  $G = c \cdot d$ . This is the core operation for the SOS Montgomery multiplication variant. Each field element  $a, b, c, d$  is represented with two 32-bit words. Figure 1 depicts the NEON dataflow of this algorithm.

---

**Algorithm 3** Computing double integer product with NEON

---

**Require:**  $a = (a_0, a_1)$ ,  $b = (b_0, b_1)$ ,  $c = (c_0, c_1)$  y  $d = (d_0, d_1)$   
**Ensure:**  $F = a \cdot b$ ,  $G = c \cdot d$

- 1:  $F \leftarrow 0$ ,  $G \leftarrow 0$
- 2: **for**  $i = 0 \rightarrow 1$  **do**
- 3:    $C_1 \leftarrow 0$ ,  $C_2 \leftarrow 0$
- 4:   **for**  $j = 0 \rightarrow 1$  **do**
- 5:      $(C_1, S_1) \leftarrow F_{i+j} + a_j \cdot b_i + C_1$ ,  $(C_2, S_2) \leftarrow G_{i+j} + c_j \cdot d_i + C_2$ ,
- 6:      $F_{i+j} = S_1$ ,  $G_{i+j} = S_2$
- 7:   **end for**
- 8:    $F_{i+n} = C_1$ ,  $G_{i+n} = C_2$
- 9:   **return**  $F, G$
- 10: **end for**

---

Because of their ability to perform two multiplications at once, NEON instructions are very useful for accelerating arithmetic computations. However, data loading and storing is in general costly since the NEON registers have to be fed by storing data into consecutive 32-bit ARM registers. Hence, in order to take a real advantage of NEON, load/store instructions should be avoided as much as possible, which is easier to accomplish if the arithmetic algorithms are

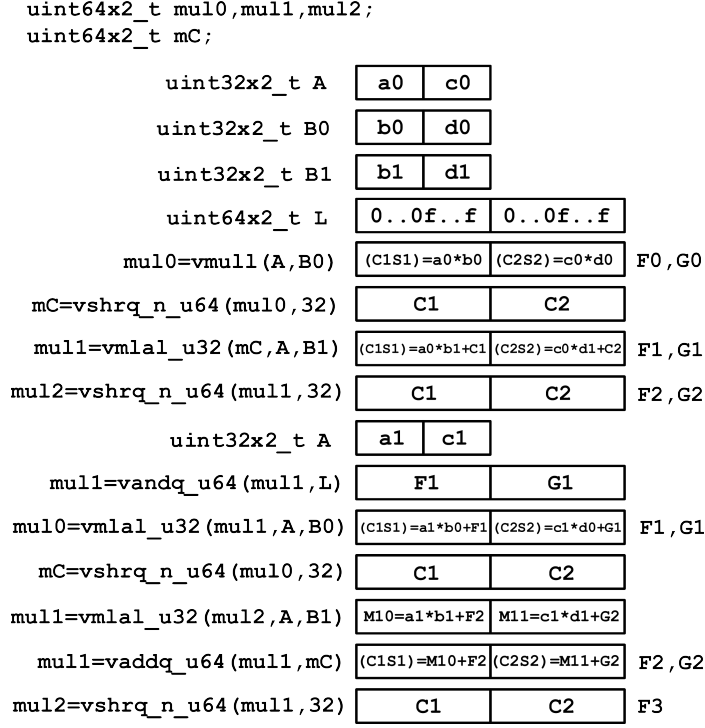


Fig. 1. NEON Implementation of Algorithm 3

specified with little data dependency among the multiplier operands. In the case of  $\mathbb{F}_{p^2}$  arithmetic, two independent multiplications over  $\mathbb{F}_p$  were implemented using NEON as follows. Let us consider  $|p| = 254$  bits and define the following three functions:  $mul_{NEON}$ ,  $mule_{NEON}$  and  $red_{NEON}$ . The first one performs two independent multiplications in  $\mathbb{F}_p$  using the CIOS method, *i.e.* given  $a, b, c, d, f, g \in \mathbb{F}_p$ , define  $mul_{NEON}$  as  $(f, g) \leftarrow mul_{NEON}(a, b, c, d)$  where  $f = a \cdot b \bmod p$  and  $g = c \cdot d \bmod p$ . The second function  $mule_{NEON}$  performs two integer multiplications:  $(F, G) \leftarrow mule_{NEON}(a, b, c, d)$ , with  $a, b, c, d \in \mathbb{F}_p$ ,  $F = a \cdot b$  and  $G = c \cdot d$ , where  $|G| = |F| = 508$  bits. Finally, the third function  $red_{NEON}$  implements the Montgomery reduction defined as  $(f, g) \leftarrow red_{NEON}(F, G)$ , where  $f, g \in \mathbb{F}_p$  and  $|G| = |F| = 512$  bits. Making use of the aforementioned functions, Algorithms 4 and 5 compute a multiplication and a squaring in  $\mathbb{F}_{p^2}$ , respectively. Notice that In step 4 of Alg. 4 the function  $mul256$  stands for a single integer multiplication.

---

**Algorithm 4** NEON multiplication over  $\mathbb{F}_{p^2}$ 

---

**Require:**  $A = a_0 + a_1u$ ,  $B = b_0 + b_1u \in \mathbb{F}_{p^2}$

**Ensure:**  $C = A \cdot B \in \mathbb{F}_{p^2}$

1:  $s \leftarrow a_0 + a_1$   
2:  $t \leftarrow b_0 + b_1$   
3:  $(d_0, d_1) \leftarrow \text{mul}_{\text{NEON}}(s, t, a_0, b_0)$   
4:  $d_2 \leftarrow \text{mul}_{256}(a_1, b_1)$   
5:  $d_0 \leftarrow d_0 - d_1 - d_2$   
6:  $d_1 \leftarrow d_1 - d_2$   
7:  $(c_1, c_0) \leftarrow \text{red}_{\text{NEON}}(d_0, d_1)$   
8: **return**  $C = c_0 + c_1u$

---

---

**Algorithm 5** NEON Squaring over  $\mathbb{F}_{p^2}$ 

---

**Require:**  $A = a_0 + a_1u \in \mathbb{F}_{p^2}$

**Ensure:**  $C = A^2 \in \mathbb{F}_{p^2}$

1:  $c_0 \leftarrow a_0 - a_1$   
2:  $c_2 \leftarrow a_0 + a_1$   
3:  $(c_1, c_0) \leftarrow \text{mul}_{\text{NEON}}(a_0, a_1, c_0, c_2)$   
4:  $c_1 \leftarrow 2c_1$   
5:  $c_0 \leftarrow c_0 - c_1$   
6: **return**  $C = c_0 + c_1u$

---

## 4 Elliptic curve arithmetic

Elliptic curve points were representing using projective Jacobian coordinates, where a BN elliptic curve  $E$  is given as  $Y^2 = X^3 + BZ^6$ . A point  $(X_1 : Y_1 : Z_1)$  in  $E$  corresponds to the affine point  $(X_1/Z_1^2, Y_1/Z_1^3)$ , with  $Z_1 \neq 0$ . The point at infinity  $\mathcal{O}$  is represented as  $(1, 1, 0)$ , whereas the additive inverse of  $(X_1 : Y_1 : Z_1)$  is  $(X_1 : -Y_1 : Z_1)$ .

**Point doubling.** Given  $P = (X_1 : Y_1 : Z_1)$ , the point  $2P = (X_3 : Y_3 : Z_3)$  can be calculated with 4 squarings and 3 multiplications according to the next sequence of operations,

$$\begin{aligned} t_1 &\leftarrow Y_1^2, t_2 \leftarrow 4X_1 \cdot t_1, t_3 \leftarrow 8t_1^2, t_4 \leftarrow 3X_1^2, \\ X_3 &\leftarrow t_4^2 - 2t_2, Y_3 \leftarrow t_4 \cdot (t_2 - X_3) - t_3, Z_3 \leftarrow 2Y_1 \cdot Z_1 \end{aligned}$$

**Mixed point addition.** Let  $P = (X_1 : Y_1 : Z_1)$  with  $Z_1 \neq 0$  and  $Q = (X_2 : Y_2 : 1)$ , for  $P \neq \pm Q$ , the addition  $R = P + Q = (X_3 : Y_3 : Z_3)$  can be obtained at a cost of 3 squarings and 8 multiplications according to the next sequence,

$$\begin{aligned} t_1 &\leftarrow Z_1^2, t_2 \leftarrow Z_1 \cdot t_1, t_3 \leftarrow X_2 \cdot t_1, t_4 \leftarrow Y_2 \cdot t_2, t_5 \leftarrow t_3 - X_1, \\ t_6 &\leftarrow t_4 - Y_1, t_7 \leftarrow t_5^2, t_8 \leftarrow t_7 \cdot t_5, t_9 \leftarrow X_1 \cdot t_7, \\ X_3 &\leftarrow t_6^2 - (t_8 + 2t_9), Y_3 \leftarrow t_6 \cdot (t_9 - X_3) - Y_1 \cdot t_8, Z_3 \leftarrow Z_1 \cdot t_5 \end{aligned}$$

Taking advantage of the inherent parallelism of the above sequences, a NEON implementation of the point addition and point doubling operations is shown in Algorithms 6 and 7, respectively.

### 4.1 Scalar Multiplication

The elliptic curve scalar multiplication operation computes the multiple  $R = [\ell]P$ , with  $\ell \in \mathbb{Z}_r$ ,  $P, R \in E(\mathbb{F}_p)$ , which corresponds to the point resulting of



---

**Algorithm 6** Point doubling with Jacobian coordinates

---

**Require:**  $P = (X_1 : Y_1 : Z_1) \in \mathbb{G}_1$   
**Ensure:**  $2P = (X_3 : Y_3 : Z_3) \in \mathbb{G}_1$   
1:  $(t_1, t_4) \leftarrow \text{mul}_{NEON}(Y_1, Y_1, 3X_1, X_1)$   
2:  $(t_2, t_3) \leftarrow \text{mul}_{NEON}(4X_1, t_1, 4t_1, 2t_1)$   
3:  $(X_3, Z_3) \leftarrow \text{mul}_{NEON}(t_4, t_4, 2Y_1, Z_1)$   
4:  $X_3 \leftarrow X_3 - 2t_2$   
5:  $Y_3 \leftarrow t_4 \cdot (t_2 - X_3) - t_3$   
6: **return**  $(X_3 : Y_3 : Z_3)$

---

---

**Algorithm 7** Mixed point addition

---

**Require:**  $P = (X_1 : Y_1 : Z_1)$  and  $Q = (X_2 : Y_2 : 1) \in \mathbb{G}_1$   
**Ensure:**  $R = P + Q = (X_3 : Y_3 : Z_3) \in \mathbb{G}_1$   
1:  $t_1 \leftarrow Z_1^2$   
2:  $(t_2, t_3) \leftarrow \text{mul}_{NEON}(Z_1, t_1, X_2, t_1)$   
3:  $t_5 \leftarrow t_3 - X_1$   
4:  $(t_4, t_7) \leftarrow \text{mul}_{NEON}(Y_2, t_2, t_5, t_5)$   
5:  $(t_8, t_9) \leftarrow \text{mul}_{NEON}(t_7, t_5, t_7, X_1)$   
6:  $t_6 \leftarrow t_4 - Y_1$   
7:  $(X_3, Z_3) \leftarrow \text{mul}_{NEON}(t_6, t_6, Z_1, t_5)$   
8:  $X_3 \leftarrow X_3 - (t_8 + 2t_9)$   
9:  $(Y_3, t_0) \leftarrow \text{mul}_{NEON}(t_6, t_9 - X_3, Y_1, t_8)$   
10:  $Y_3 \leftarrow Y_3 - t_0$   
11: **return**  $(X_3 : Y_3 : Z_3)$

---

adding  $P$  to itself  $\ell$  times. The average cost of computing  $[\ell]P$  by a random  $n$ -bit scalar  $\ell$  using the customary double-and-add method is of about,  $nD + \frac{n}{2}A$ , where  $A$  is the cost of a point addition, and  $D$  is the cost of a point doubling.

The customary method to speed up this operation reduces the Hamming weight of the scalar  $\ell$  by representing it in its non-adjacent form (NAF). The technique can be easily extended to the  $w$ -NAF representation, namely,  $\ell = \sum_{i=0}^{n-1} \ell_i 2^i$ ,  $|\ell_i| \leq 2^{w-1}$ , and at most one of any  $w$  consecutive digits is non-zero, with  $\ell_{n-1} \neq 0$ , where the length  $n$  is at most one bit larger than the bitsize of the scalar  $n$  and the resulting Hamming weight is approximately  $1/(w+1)$ . The estimated cost of the scalar multiplication reduces to,  $nD + \frac{n}{w+1}A$ , plus the cost of the precomputation of the multiples  $P_i = [i]P$ , for  $i \in [1, 3, \dots, 2^{w-1} - 1]$ . Due to this exponential penalty, in most applications a rather conservative value of  $w \in [3, 5]$  is selected.

When the point  $P$  is fixed, some form of the comb method is usually preferred. Given an  $w$ -window size and a known-point  $P$ , one can pre-compute for all of the possible bit strings  $(a_{w-1}, \dots, a_0)$  the following  $2^w$  multiples of  $P$ :  $[a_{w-1}, \dots, a_2, a_1, a_0]P = a_{w-1}2^{(w-1)d}P + \dots + a_22^{2d}P + a_12^dP + a_0P$ . Then, the scalar  $\ell$  is scanned column-wise by recoding it into  $d$  blocks each one with a bit length of  $w$  bits, where  $d = \lceil r/w \rceil$ . The computational and storage costs of the comb method is of  $d(A+D)$ , and a look-up table of  $2^w$  points, respectively. Notice that the storage cost can be reduced to a half by using a signed representation of the scalar  $\ell$ , as presented by Hamburg in [14]. A further speedup in the computation of the scalar multiplication can be achieved if there exists an efficient-computable endomorphism  $\psi$  over  $E/\mathbb{F}_p$  such that  $\psi(P) = \lambda P$  [11]. In the case of BN curves, given a cube root of unity  $\beta \in \mathbb{F}_p$ , one has that the mapping

$\psi : E_1 \rightarrow E_1$  defined as,  $(x, y) \rightarrow (\beta x, y)$  and  $\mathcal{O} \rightarrow \mathcal{O}$ , is an endomorphism over  $\mathbb{F}_p$  with a characteristic polynomial given as  $\lambda^2 + \lambda \equiv -1 \pmod{r}$ ,  $\lambda = 36z^4 - 1$ . The scalar  $\ell$  can be rewritten as  $\ell \equiv \ell_0 + \ell_1 \lambda \pmod{r}$ , where  $|\ell_i| < \sqrt{r}$ , which allows to compute the scalar multiplication as,  $[\ell]P = [\ell_0]P + [\ell_1]\psi(P)$ , at an approximate cost of  $[D + (2^{\omega-2} - 1)A] + \left[\frac{n}{\omega+1}A + \frac{n}{2}D\right]$ .

In the case that the scalar multiplication in  $\mathbb{G}_2$ , *i.e.* the computation of the multiple  $S = [\ell]Q$ , with  $\ell \in \mathbb{Z}_r$ ,  $Q, S \in E(\mathbb{F}_{p^2})$ , is of interest, one can take advantage of the Frobenius endomorphism to extend the two-dimensional GLV method to a four dimension version using the GS approach [10]. Let  $E$  be a BN elliptic curve over  $\mathbb{F}_p$  with embedding degree  $k = 12$  and let  $\tilde{E}(\mathbb{F}_{p^2})$  be the sixth degree twist of  $E$ . Let  $\pi_p$  be the Frobenius operator in  $E$ , then  $\psi = \phi^{-1}\pi_p\phi$  is an endomorphism on  $\tilde{E}$  such that  $\psi : \tilde{E}(\mathbb{F}_{p^2}) \rightarrow \tilde{E}(\mathbb{F}_{p^2})$ . Then for  $Q \in \tilde{E}(\mathbb{F}_{p^2})$ , it holds that  $\psi^k(Q) = Q$ ,  $\psi(Q) = pQ$ , and  $\psi$  satisfies  $\psi^4 - \psi^2 + 1 = 0$ . Since  $p \equiv t - 1 \pmod{r}$ , the scalar  $\ell$  can be decomposed as  $\ell = \ell_0 + \ell_1 \lambda + \ell_2 \lambda^2 + \ell_3 \lambda^3$  with  $\lambda = t - 1$ , which allows to compute the scalar multiplication in  $\mathbb{G}_2$ , as,  $[\ell]Q = [\ell_0]Q + [\ell_1]\psi(Q) + [\ell_2]\psi^2(Q) + [\ell_3]\psi^3(Q)$ , at an approximate cost of  $[D + (2^{\omega-2} - 1)A] + \left[\frac{n}{\omega+1}A + \frac{n}{4}D\right]$ .

Likewise, in the case of the exponentiation in  $\mathbb{G}_T$ , the operation  $f^e$  with  $f \in \mathbb{G}_T, e \in \mathbb{Z}_r$  can then be accomplished by rewriting the exponent  $e$  in base  $p$  as,  $e = e_0 + e_1 \cdot p + e_2 \cdot p^2 + e_3 \cdot p^3$ , with  $|e_i| \approx |p|$ , followed by the computation,  $f^e = f^{e_0} \cdot f^{e_1 p} \cdot f^{e_2 p^2} \cdot f^{e_3 p^3}$ . Notice that the Frobenius mapping  $e^{p^i}$  for  $i = 0, \dots, 3$ , has a negligible computational cost. Notice also that the identity  $f^p = f^\lambda = f^{6x^2}$ , holds.

## 5 Bilinear Pairing Arithmetic

In this section we briefly describe the Miller's loop and final exponentiation computations as well as the algorithm utilized to perform multi-pairing computations.

**Miller Loop** The main operations of Algorithm 1 are the evaluation of the tangent line  $\ell_{T,T}$  and the doubling of the point  $T$ ; as well as the secant line evaluation and the computation of the point addition  $T + P$ . The most efficient way to perform above operations is through the usage of standard projective coordinates, where the projective point  $(X_1 : Y_1 : Z_1)$  in the elliptic curve  $E$  corresponds to the affine point  $(X_1/Z_1, Y_1/Z_1)$ . Given the curve  $\tilde{E}/\mathbb{F}_{p^2}$  defined as  $\tilde{E} : y^2 = x^3 + b'$  whose projective form is  $Y^2Z = X^3 + bZ^3$ , one can calculate  $2T = (X_3 : Y_3 : Z_3) \in E'(\mathbb{F}_{p^2})$  using the formulas [4]:

$$\begin{aligned} X_3 &= \frac{X_1 Y_1}{2} (Y_1^2 - 9b' Z_1^2) \\ Y_3 &= \left[ \frac{1}{2} (Y_1^2 + 9b' Z_1^2) \right]^2 - 27b'^2 Z_1^4 \\ Z_3 &= 2Y_1^3 Z_1 \end{aligned}$$

whereas the tangent line  $\ell_{T,T}$  evaluated on  $P = (x_P, y_P) \in E(\mathbb{F}_p)$  is given as,

$$\ell_{T,T}(P) = -2Y_1Z_1y_P + 3X_1x_Pw + (3b'Z_1^2 - X_1^2)w^3 \in \mathbb{F}_{p^{12}}$$

In the same way, given the points  $T = (X_1, Y_1, Z_1), Q = (X_2, Y_2, 1) \in E'(\mathbb{F}_{p^2})$  and  $P = (x_P, y_P) \in E(\mathbb{F}_p)$  one can calculate the point addition  $R = T + Q = (X_3, Y_3, Z_3)$  and  $\ell_{T,Q}(P)$  as [4]:

$$\begin{aligned} \ell_{T,Q}(P) &= \lambda y_P - \theta x_P w + (\theta X_2 - \lambda Y_2) w^3, \\ X_3 &= \lambda(\lambda^2 + Z_1\theta^2 - 2X_1\lambda^2) \\ Y_3 &= \theta(3X_1\lambda^2 - \lambda^3 - Z_1\theta^2) - Y_1\lambda^3 \\ Z_3 &= Z_1\lambda^3 \end{aligned}$$

where  $\theta = Y_1 - Y_2Z_1$  and  $\lambda = X_1 - X_2Z_1$ .

Another important aspect of the Miller's algorithm is the multiplication of the Miller variable  $f$  by the line (either tangent or secant) evaluation. However, the evaluation of the lines  $\ell_{Q,Q}$  and  $\ell_{T,Q}$  produce a sparse element in the group  $\mathbb{F}_{p^{12}}^*$  with half of its coefficients having a zero value, which motivates the idea that any product of  $f \in \mathbb{F}_{p^{12}}$  with  $\ell_{Q,Q}$  or  $\ell_{T,Q}$  should be performed using a procedure specially tailored for computing sparse multiplications.

**Final Exponentiation** The exponent  $e = (p^k - 1)/r$  in the BN final exponentiation can be broken into two parts as,

$$(p^{12} - 1)/r = [(p^{12} - 1)/\Phi_{12}(p)] \cdot [\Phi_{12}(p)/r],$$

where  $\Phi_{12}(p) = p^4 - p^2 + 1$  denotes the twelfth cyclotomic polynomial evaluated in  $p$ . Computing the map  $f \mapsto f^{(p^{12}-1)/\Phi_{12}(p)}$  is relatively inexpensive, costing only a few multiplications, inversions, and inexpensive  $p$ -th exponentiations in  $\mathbb{F}_{p^{12}}$ . Raising to the power  $d = \Phi_{12}(p)/r = (p^4 - p^2 + 1)/r$  is considered more difficult. This part was computed using a multiple  $d'$  of  $d$  where  $r \nmid d$  as discussed in [9], which allowed a lower number of operations. Using the BN parameter  $z$ , the exponentiation of  $g^{d'(z)}$  requires the calculation of the following addition chain,

$$fz \mapsto f^{2z} \mapsto f^{4z} \mapsto f^{6z} \mapsto f^{6z^2} \mapsto f^{12z^2} \mapsto f^{12z^3},$$

which requires 3 exponentiations by  $z$ , 3 squarings and one multiplication over  $\mathbb{F}_{p^{12}}$ . Finally, given the variables  $a = g^{12z^3} \cdot g^{6z^2} \cdot g^{6z}$  and  $b = a \cdot (g^{2z})^{-1}$ , the exponentiation of  $g^{d'(z)}$  is calculated as follows:

$$g^{d'(z)} = [a \cdot g^{6z^2} \cdot g] \cdot [b]^p \cdot [a]^{p^2} \cdot [b \cdot g^{-1}]^{p^3} \in \mathbb{F}_{p^{12}}^\times$$

Since  $g \in \mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ , the cost of  $g^{d'(z)}$  is 3 Frobenius operators, 3 exponentiations by  $z$ , 10 multiplications in  $\mathbb{F}_{p^{12}}$  and 3 squarings in  $\mathbb{G}_{\Phi_6}(\mathbb{F}_{p^2})$ . It should be noted that we use the Karabina compressed squaring formulas [16] for performing the exponentiation-by- $z$  step.

**Multipairing** Products of pairings are part of the required computations in the ABE protocol. For this, one can make use of the pairing bilinear property to group pairings sharing one of the input parameters. If all the pairings share a common input point, then one can exchange  $n$  pairing products by  $n - 1$  point additions and a single pairing using the identity,

$$\prod_{i=0}^{n-1} e(Q, P_i) = e(Q, \sum_{i=0}^{n-1} P_i),$$

If a product of pairings is still needed, and the previous method was already exploited, there is still room for obtaining significant speedups. For instance, one can compute this product by performing a multi-pairing (or simultaneous product of pairings), by exploiting the well-known techniques employed in the multi-exponentiation setting. In essence, in a multipairing computation not only the costly final exponentiation step can be shared, but also, one can share both the Miller variable  $f$ , and the squaring computations performed in the step 4 of Algorithm 1. A further performance improvement can be achieved if the point  $Q$  in  $\mathbb{G}_2$  is known in advance, since in this case one can pre-compute some of the line evaluation operations, which allows to compute the lines on the point  $P \in \mathbb{G}_1$  as,  $\ell_{\cdot, \cdot}(P) = l_0 y_P + l_1 x_P w + l_2 w^3$ . Algorithm 8 computes the product of  $n$  optimal BN pairings as discussed above.

---

**Algorithm 8** Multipairing for BN curves

---

<p><b>Require:</b> <math>P_1, P_2, \dots, P_n \in \mathbb{G}_1,</math>  <math>Q_1, Q_2, \dots, Q_n \in \mathbb{G}_2</math></p> <p><b>Ensure:</b> <math>g = \prod_{i=1}^n e(Q_i, P_i)</math></p>	<p>11: <math>f \leftarrow f \cdot \ell_{T_i, Q_i}(P_i), T_i \leftarrow T_i + Q_i</math></p> <p>12: <b>else if</b> <math>s_j = -1</math> <b>then</b></p> <p>13: <math>f \leftarrow f \cdot \ell_{T_i, -Q_i}(P_i), T_i \leftarrow T_i - Q_i</math></p> <p>14: <b>end if</b></p> <p>15: <b>end for</b></p> <p>16: <b>end for</b></p> <p>17: <b>for</b> <math>i = 1 \rightarrow n</math> <b>do</b></p> <p>18: <math>R_1 \leftarrow \pi(Q_i), R_2 \leftarrow \pi^2(Q_i)</math></p> <p>19: <math>f \leftarrow f \cdot \ell_{T_i, R_1}(P), T_i \leftarrow T_i + R_1,</math>  <math>f \leftarrow f \cdot \ell_{T_i, -R_2}(P), T_i \leftarrow T_i - R_2</math></p> <p>20: <b>end for</b></p> <p>21: <math>g \leftarrow f^{(p^{12}-1)/r}</math></p> <p>22: <b>return</b> <math>g</math></p>
<p>1: Write <math>s = 6z + 2</math> as <math>s = \sum_{j=0}^{l-1} s_j</math> with  <math>s_j \in \{-1, 0, 1\}</math></p> <p>2: <math>f \leftarrow 1</math></p> <p>3: <b>for</b> <math>i = 1 \rightarrow n</math> <b>do</b></p> <p>4: <math>T_i \leftarrow Q_i</math></p> <p>5: <b>end for</b></p> <p>6: <b>for</b> <math>j = l - 2 \rightarrow 0</math> <b>do</b></p> <p>7: <math>f \leftarrow f^2</math></p> <p>8: <b>for</b> <math>i = 0 \rightarrow n</math> <b>do</b></p> <p>9: <math>f \leftarrow f \cdot \ell_{T_i, T_i}(P_i), T_i \leftarrow 2T_i</math></p> <p>10: <b>if</b> <math>s_j = 1</math> <b>then</b></p>	<p>11: <math>f \leftarrow f \cdot \ell_{T_i, Q_i}(P_i), T_i \leftarrow T_i + Q_i</math></p> <p>12: <b>else if</b> <math>s_j = -1</math> <b>then</b></p> <p>13: <math>f \leftarrow f \cdot \ell_{T_i, -Q_i}(P_i), T_i \leftarrow T_i - Q_i</math></p> <p>14: <b>end if</b></p> <p>15: <b>end for</b></p> <p>16: <b>end for</b></p> <p>17: <b>for</b> <math>i = 1 \rightarrow n</math> <b>do</b></p> <p>18: <math>R_1 \leftarrow \pi(Q_i), R_2 \leftarrow \pi^2(Q_i)</math></p> <p>19: <math>f \leftarrow f \cdot \ell_{T_i, R_1}(P), T_i \leftarrow T_i + R_1,</math>  <math>f \leftarrow f \cdot \ell_{T_i, -R_2}(P), T_i \leftarrow T_i - R_2</math></p> <p>20: <b>end for</b></p> <p>21: <math>g \leftarrow f^{(p^{12}-1)/r}</math></p> <p>22: <b>return</b> <math>g</math></p>

---

## 6 Attribute Based-Encryption

Attribute-Based Encryption (ABE) is a relatively new encryption scheme where an identity is seen as a set of attributes. In this scheme a user can access to

some resources only if she had a set of privileges (called attributes) satisfying a control access policy previously defined. The policy is described through a boolean formula, which can be represented by an access structure and can be implemented using a linear secret-sharing scheme (LSSS) [19, 21]. The LSSS structure is described by the pair  $(M, \rho)$ , where  $M \in \mathbb{F}_r$  is an  $u \times t$  matrix, where  $u, t$  are the number of required attributes and the access policy threshold, respectively; whereas  $\rho$  is a label function that according to the policy links each row of the matrix  $M$  to an attribute. For the sake of efficiency and as Scott did in [20], we reformulate the protocol from its original symmetric setting to an asymmetric one where some scheme parameters are conveniently defined in  $\mathbb{G}_1$  whereas others are in  $\mathbb{G}_2$ . The ABE scheme is made up of four algorithms [21]: Setup, Encrypt, Key Generation y Decrypt, as described next.

**Setup** This algorithm takes as input the security parameter  $\lambda$  and the set of  $U$  attributes. The security parameter becomes the main criterion to select the groups  $\mathbb{G}_1$  and  $\mathbb{G}_2$  of order  $r$  and the generators  $P \in \mathbb{G}_1$  and  $Q \in \mathbb{G}_2$ . The points  $H_1, \dots, H_U \in \mathbb{G}_1$  are generated from the attribute universe and the algorithm also chooses two random integers  $a, \alpha \in \mathbb{F}_r$ . The public key is published as,  $\text{PK} = \{P, Q, e(Q, P)^\alpha, [a]P, H_1, \dots, H_U\}$ . Additionally, the authority establishes  $\text{MSK} = [\alpha]P$  as her master secret key. This algorithm has a cost of one pairing, two scalar multiplications and  $U$  MapToPoint functions. Notice that it is assumed that the elements  $P, Q, [a]P$  and  $e(Q, P)^\alpha$  are all known in advance. On the contrary, the points  $H_1, \dots, H_U$  were not considered as fixed points since the attribute universe has a variable length, a fact that is also reflected in the storage cost.

**Encrypt** The algorithm for encryption takes as input the public key PK, the message  $\mathcal{M}$  to be encrypted, and the LSSS access structure  $(M, \rho)$ , where  $M \in \mathbb{F}_r$  is an  $u \times t$  matrix as described above. The algorithm starts by randomly selecting a column vector  $v = (s, y_2, \dots, y_t)^T \in \mathbb{F}_r^n$  that will be used to securely share the secret exponent  $s$ . For  $i = 1$  to  $u$ , it calculates  $\lambda_i = M_i \cdot v$  where  $M_i$  is the  $1 \times t$  vector corresponding to the  $i$ -th row of  $M$ . The scalars  $r_1, \dots, r_u \in \mathbb{F}_r$  are also randomly chosen. Then, the cipher CT is published as follows:

$$\text{CT} = \begin{cases} C = \mathcal{M}e(Q, P)^{\alpha s}, C' = [s]Q, \\ (C_1 = [\lambda_1]([a]P) - [r_1]H_{\rho(1)}, D_1 = [r_1]Q), \\ \vdots \\ (C_u = [\lambda_u]([a]P) - [r_u]H_{\rho(u)}, D_u = [r_u]Q), \end{cases}$$

which is sent along with the LSSS access structure  $(M, \rho)$ .

The comb method was applied to compute scalar multiplications involving the fixed points  $Q, P$  and  $[a]P$ . In the same way, one can apply this method to obtain the powering of  $e(Q, P)^\alpha$  by the exponent  $s$ . The GLV method was used to compute scalar multiplications with the points  $H_i \in \mathbb{G}_1$ . Hence the cost of encryption is one multiplication and one fixed exponentiation in  $\mathbb{G}_T$ ,  $u$  fixed point multiplications in  $\mathbb{G}_1$ ,  $u + 1$  fixed point multiplications in  $\mathbb{G}_2$  and  $u$  point multiplication in  $\mathbb{G}_1$ .

**Key Generation** This algorithm takes as input the master secret key  $\text{MSK} = \alpha P$  and a set of attributes  $S$ . First the algorithm selects a random number  $t \in \mathbb{F}_r$ , then it generates the attribute-based private key as follows,

$$\text{SK} = \{ K = [\alpha]P + [t]([a]P), L = [t]Q, \forall x \in S K_x = [t]H_x \}$$

Let  $N$  be the number of attributes on  $S$ , since the points  $aP$  and  $Q$  are known, the cost of this algorithm is one fixed point multiplication in  $\mathbb{G}_1$ , one fixed point in  $\mathbb{G}_2$  and  $N$  point multiplications in  $\mathbb{G}_1$ . For the first two scalar multiplications the *comb* method was used, and the GLV method for the rest.

**Decrypt** This algorithm takes as inputs the cipher CT with the access structure  $(M, \rho)$  and the private key SK for a set  $S$ . Suppose  $S$  satisfies the access structure and define  $I \subset \{1, 2, \dots, u\}$  as  $I = \{i : \rho(i) \in S\}$ . Let  $\{\omega_i \in \mathbb{Z}\}_{i \in I}$  be the set of constants such that if  $\lambda_i$  is a valid share of a secret  $s$  according to  $M$ , then  $\sum_{i \in I} \omega_i \lambda_i = \Delta s$  with  $\Delta \in \mathbb{F}_r$ . The decryption algorithm first computes:

$$\left( e(L, \sum_{i \in I} [\omega_i] C_i) \prod_{i \in I} e(D_i, [\omega_i] K_{\rho(i)}) \right)^{\frac{1}{\Delta}} / e(C', K) = e(P, Q)^{-\alpha s}, \quad (3)$$

Followed by the multiplication of this value by  $C$  as defined in Eq. (3). If  $S$  satisfies the access structure this should recover the message  $\mathcal{M}$ .

The variable  $\Delta$  guarantees low size constants  $\omega_i$ , i.e.,  $|\omega_i| < 64$  bits, which allows us to perform the scalar multiplications involving these constants using an  $\omega$ -NAF method. We called this operation *short scalar multiplication*. Let  $N < u$  be the number of elements of  $I$ , then the computational cost of Eq. (3) is of  $2N$  short multiplications in  $\mathbb{G}_1$ , an  $N + 2$  multipairing computation,  $N$  point additions in  $\mathbb{G}_1$  and one exponentiation in  $\mathbb{G}_T$  which can be computed using the GS method. Also, since  $L \in \mathbb{G}_2$  is a known point, its lines evaluations were precomputed.

## 7 Implementation Results

This section presents our main implementation results classified into three subsections: bilinear pairings, scalar multiplication and the ABE scheme timings.

### 7.1 Pairing costs

The total cost of the pairing is split into two parts: the Miller Loop and the final exponentiation. In the case of the Miller Loop, Table 2 shows the estimated cost of its main operations.

The length of the optimal pairing Miller loop in Alg. 1 is  $s = 6z + 2$ . Choosing the BN parameter as  $z = -2^{62} - 2^{55} - 1$  implies that the signed binary representation of  $s$  has a length of 65 bits with a Hamming weight of 5. Hence, the total cost of the Miller Loop is:

$$\text{M.L.} = 238\tilde{m} + 266\tilde{s} + 1694\tilde{m}_E + 126\tilde{s}_E + 875\tilde{r}_E + 588m_\xi + 8739\tilde{a} + 288m$$

Operation	Cost
Sparse Multiplication $f \cdot \ell_{\cdot, \cdot}(P)$	$13\tilde{m}_E + 6\tilde{r}_E + 3m_\xi + 50\tilde{a}$
Doubling and line evaluation. $\ell_{T,T}(P)$ y $2T$	$3\tilde{m} + 4\tilde{s} + 2\tilde{s}_E + 1\tilde{r}_E + 19\tilde{a} + 4m$
Adding and line evaluation. $\ell_{T,Q}(P)$ y $T + Q$	$7\tilde{m} + 2\tilde{s} + 4\tilde{m}_E + 2\tilde{r}_E + 10\tilde{a} + 4m$
Frobenius operator on $Q$ $\pi(Q) \circ \pi^2(Q)$	$4m$

**Table 2.** Costs of Miller's loop main operations

As for the final exponentiation step, we estimate the cost of the exponentiation by  $z$  over  $\mathbb{F}_{p^{12}}$  as,  $9\tilde{m} + 378\tilde{s} + 36\tilde{m}_E + 12\tilde{r}_E + 204m_\xi + 1704\tilde{a} + \tilde{i}$ , which implies that the cost of the final exponentiation is approximately,  
E.F. =  $38\tilde{m} + 1161\tilde{s} + 346\tilde{m}_E + 9\tilde{s}_E + 121\tilde{r}_E + 721m_\xi + 8m + 6442\tilde{a} + 4\tilde{i}$

In the case of the multipairing computation, we observe that when  $n = 1$ , its cost is equivalent to a pairing, whereas for  $n > 1$  we have that for each extra pairing the final exponentiation and the cost of the squaring in the Miller Loop is saved as shown in Table 3.

# Pairings	Appr. Cost in $\mathbb{F}_{p^2}$	Appr. cost in $\mathbb{F}_p$
single pairing	$280\tilde{m} + 1436\tilde{s} + 2148\tilde{m}_E + 135\tilde{s}_E + 996\tilde{r}_E + 1309m_\xi + 15175\tilde{a} + 288m + 4\tilde{i}$	$10652m + 32968a + 4i$
each extra pairing	$238\tilde{m} + 266\tilde{s} + 938\tilde{m}_E + 126\tilde{s}_E + 497\tilde{r}_E + 210m_\xi + 4767\tilde{a} + 288m$	$4474m + 9954$
each extra fixed point pairing	$910\tilde{m}_E + 420\tilde{r}_E + 210m_\xi + 3500\tilde{a} + 280m$	$2100m + 7420a$

**Table 3.** Costs of the multipairing computation

Table 4 presents a comparison against the works [1] and [13]. In [1] a pairing library using affine coordinates was presented, whereas [13] reports an assembler optimized pairing library using standard projective coordinates. Notice that the authors in [13] did not consider the usage of NEON.

## 7.2 Costs of the scalar multiplication and field exponentiation

Table 5 shows the timings obtained for the computation of scalar multiplication in  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , and field exponentiation in the group  $\mathbb{G}_T$ . Scalar multiplication using  $w$ -NAF only was employed for small 64-bit scalars, with  $w = 3$ . The comb method was the choice for computing fixed point scalar multiplication with a window size of  $w = 8$ . We stress that the  $w$ -NAF was used in combination with both the GLV and GS methods.

Work	Processor	$10^3$ clock cycles for 254 bits						
		$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML	FE	$O-\hat{a}(\cdot)$
[1]	Tegra 2 <sup>a</sup>	1.42	8.18	5.20	26.61	26,320	24,690	51,010
[13]	Apple A5 <sup>b</sup>	0.25	3.48	2.88	19.19	8,338	5,483	13,821
	TI OMAP <sup>c</sup>	0.16	3.37	2.53	16.86	8,231	5,258	13,489
	(ASM)	0.12	2.95	2.48	16.60	7,376	4,510	11,886
This Work	Tegra 2 <sup>a</sup>	0.17	3.41	2.41	39.25	8,313	5,269	13,582
	Exynos <sup>d</sup> (NEON)	0.17	3.42	2.41	39.21	8,348	4,607	13,618
		0.16	2.29	2.00	60.37	5,758	3,794	9,477
	Exynos <sup>e</sup>	0.14	1.36	0.86	29.01	3,388	2,353	5,838

a. NVidia Tegra 2 (ARM v7) Cortex-A9 a 1.0 GHz (C)

b. iPad 2 (ARM v7) Apple A5 Cortex-A9 a 1.0 GHz (C)

c. Galaxy Nexus (ARM v7) TI OMAP 4460 Cortex-A9 a 1.2 GHz (Two versions: C y ASM)

d. Galaxy Note (ARM v7) Exynos 4 Cortex-A9 a 1.4 GHz (Two versions: C y NEON)

e. Arndaleboard (ARM v7) Exynos 5 Cortex-A15 a 1.7 GHz (NEON)

**Table 4.** Clock cycle comparison for Single pairing computations

Processor	$\mathbb{G}_1$ Mult.			$\mathbb{G}_2$ Mult.			$\mathbb{G}_T$ Exp.		
	$w$ -NAF	GLV	Comb	$w$ -NAF	GS	Comb	$w$ -NAF	GS	Comb
Tegra 2	779	1977	626	2059	4190	1745	2643	5998	2727
Exynos 4 (NEON)	785	1973	627	2096	4189	1742	2633	4777	2155
	676	1698	556	1493	2933	1214	1827	4102	1863
Exynos 5	337	822	251	797	1571	636	1125	2522	1121

**Table 5.** Scalar mult. and exponentiation timings (in  $10^3$  clock cycles)

### 7.3 Attribute-based encryption costs

We could not compare our ABE scheme timings against [2], because this work only implements the decryption algorithm and it does not presents the exact timings. Table 6 reports the timings obtained when a 6-attribute policy is employed in the ABE protocol. Note that for the decryption algorithm we present the cases when  $\Delta = 1$  and  $\Delta > 1$  (see Eq. 3).

## 8 Conclusion

We presented a cryptographic library that implements Waters’ attribute encryption scheme in mobile devices operated with ARM processors. The main primitives developed were bilinear pairings and scalar multiplications in different flavors. Our library uses four different scalar multiplications according to the group, scalar size and the type of the point (either fixed or variable), providing a 127-bit security level and achieving record timings for the computation of a single bilinear pairing at this level of security when implemented on the Exynos-5 Cortex-A15 processor.

A key factor that helps us to achieve faster timings than previously reported works was the usage of the NEON technology that allows a better exploitation



Processor	Key Generator	Encryption	Decryption ( $\Delta = 1$ )	Decryption ( $\Delta \geq 1$ )
Tegra 2	18,340	31,830	63,870	74,140
Exynos 4	18,270	29,480	63,810	73,930
Exynos 4 (NEON)	15,333	24,167	43,980	50,808
Exynos 5	7,617	7,634	26,638	31,161

**Table 6.** ABE scheme with 6 attributes (Timings in  $10^3$  clock cycles)

of the inherent parallelism present in several field and elliptic curve arithmetic operations. It is illustrative to analyze the Exynos-4 scalar multiplication timings shown in Table 5. where NEON produces savings of about 14%, 30% and 15% in the computations over the  $\mathbb{G}_1$ ,  $\mathbb{G}_2$  and  $\mathbb{G}_T$  groups, respectively. Notice that the significant better performance of NEON in the computations over  $\mathbb{G}_2$  are a consequence of the rich parallelism extracted for the field squaring and multiplication over  $\mathbb{F}_{p^2}$  as it was explained in Section 4.

Another interesting aspect to remark is the performance comparison of our work against [13] for the single pairing computation at the 127 bit security level. As shown in Table 4, *without using* NEON, the two libraries perform essentially the same when implemented in the Tegra 2 and Apple A5 processors, respectively. However, taking advantage of NEON, our library outperforms the library in [13] by approximately 20% when implemented in the Exynos 4 and TI OMAP processors, respectively. Moreover, when deployed in the Exynos 5 processor, our library computes a bilinear pairing 40% faster than our Exynos 4 processor implementation; and performs a bit more than two times faster than the software in [13].

The Cortex A-15 micro-architecture is equipped with an out-of-order super-scalar pipeline, a lower latency level-2 cache unit and an improved NEON unit that according to the manufacturer’s documentation is expected to “...*deliver over five times the performance of today’s advanced smartphones*” [18].

The experimental results presented in this paper lead us to conclude that the Cortex A-15 micro-architecture and its improved NEON unit indeed offer a significantly superior performance when implementing cryptographic applications. Because of that, any future developments of pairing-based protocols on ARM processors should not omit the redesign of the (extension) field and elliptic curve arithmetic procedures so that the parallelism offered by the NEON set of instructions can be effectively used.

## Acknowledgments

We wish to thank Peter Schwabe for explaining us how to perform accurate clock cycle counts on ARM processors and for giving us feedback on the first draft of the paper and Armando Faz-Hernández for benchmarking our software in the Exynos 5 Cortex-A15 processor. We also thank Alfred Menezes for commenting on the earlier draft.

## References

1. T. Acar, K. Lauter, M. Naehrig, and D. Shumow. Affine Pairings on ARM. *IACR Cryptology ePrint Archive*, 2011:243, 2011.
2. J. A. Akinyele, C. Lehmann, M. Green, M. Pagano, Z. Peterson, and A. Rubin. Self-Protecting Electronic Medical Records Using Attribute-Based Encryption. In A. Bhattacharya, P. Dasgupta, and W. Enck, editors, *SPSM '11 Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 75–86, Chicago, Illinois, USA, October 17-21, 2010. ACM.
3. D. F. Aranha, L. Fuentes-Castañeda, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In *Pairing 2012*, volume 7708 of *Lecture Notes in Computer Science*, pages 177–195. Springer, 2013.
4. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In K. G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68, Tallinn, Estonia, May 15-19, 2011. Springer.
5. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography - SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331, 2006.
6. D. J. Bernstein and P. Schwabe. NEON crypto. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 320–339. Springer, 2012.
7. J. Chung and M. A. Hasan. Asymmetric Squaring Formulas. In P. Kornerup and J.-M. Muller, editors, *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 113–122, Montpellier, France, June 25-27, 2007. IEEE Computer Society.
8. C. Costello and D. Stebila. Fixed argument pairings. In M. Abdalla and P. S. L. M. Barreto, editors, *Progress in Cryptology - LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2010.
9. L. Fuentes-Castañeda, E. Knapp, and F. Rodríguez-Henríquez. Faster hashing to  $\mathbb{G}_2$ . In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011*, volume 7118 of *Lecture Notes in Computer Science*, pages 412–430, Toronto, ON, Canada, August 11-12, 2011. Springer.
10. S. D. Galbraith and M. Scott. Exponentiation in Pairing-Friendly Groups Using Homomorphisms. In S. D. Galbraith and K. G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 211–224, Egham, UK, September 1-3, 2008. Springer.
11. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200, Santa Barbara, California, USA, August 19-23, 2001. Springer.
12. R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In P. Q. Nguyen and D. Pointcheval, editors, *In Public Key Cryptography PKC'10*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223, Paris, France, May 26-28, 2010. Springer.
13. G. Grewal, R. Azarderakhsh, P. Longa, S. Hu, and D. Jao. Efficient implementation of bilinear pairings on ARM processors. *IACR Cryptology ePrint Archive*, 2012:408, 2012.
14. M. Hamburg. Fast and compact elliptic-curve cryptography. *Cryptology ePrint Archive*, Report 2012/309, 2012. <http://eprint.iacr.org/>.

15. D. Hankerson, A. Menezes, and M. Scott. Software implementation of pairings (Chapter 12). In M. Joye and G. Neven, editors, *Identity-based Cryptography*, volume 2 of *Cryptology and Information Security*, page 188–206, Amsterdam, The Netherlands, December 2009. IOS Press.
16. K. Karabina. Squaring in cyclotomic subgroups. *IACR Cryptology ePrint Archive*, 2010:542, 2010.
17. C. K. Koc, T. Acar, and B. S. K. Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June, 1996.
18. A. Limited. Cortex-a15 processor, 2013. <http://arm.com/products/processors/cortex-a/cortex-a15.php>.
19. Z. Liu and Z. Cao. On efficiently transferring the linear secret-sharing scheme matrix in ciphertext-policy attribute-based encryption. *IACR Cryptology ePrint Archive*, 2010:374, 2010.
20. M. Scott. On the Efficient Implementation of Pairing-Based Protocols. In L. Chen, editor, *Cryptography and Coding - 13th IMA International Conference, IMACC 2011*, volume 7089 of *Lecture Notes in Computer Science.*, pages 296–308, Oxford, UK, December 12-15, 2011. Springer.
21. B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography 2011 PKC 2011*, volume 6571 of *Lecture Notes in Computer Science.*, pages 53–70, Taormina, Italy, March 6-9, 2011. Springer.