Software implementation of an Attribute-Based Encryption scheme

Eric Zavattoni, Luis J. Dominguez Perez, Shigeo Mitsunari, Ana H. Sánchez-Ramírez, Tadanori Teruya, and Francisco Rodríguez-Henríquez, *Member, IEEE*

Abstract

A ciphertext-policy attribute-based encryption protocol 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. In this paper we present the design of a software cryptographic library that achieves record timings for the computation of a 126-bit security level attribute-based encryption scheme. We developed all the required auxiliary building blocks and compared the computational weight that each of them adds to the overall performance of this protocol. In particular, our single pairing and multi-pairing implementations achieve state-of-the-art time performance at the 126-bit security level.

Index Terms

attribute-based-encryption, pairing-based protocols, bilinear pairings, scalar multiplication

I. INTRODUCTION

A bilinear pairing, or pairing for short, can be defined as a non-degenerate bilinear mapping, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, are finite cyclic groups of prime order r. Pairings

E. Zavattoni was a visiting fellow at CINVESTAV-IPN from the Université Claude Bernard, Lyon 1, France.

L. J. Dominguez Perez is currently with CINVESTAV-LTI Unidad Tamaulipas, Cd. Victoria, TAM, 87130 MEX.

Shigeo Mitsunari is with the Cybozu Labs, Inc., Tokyo, Japan.

A. H. Sánchez-Ramírez, and. F. Rodríguez-Henríquez are with the Computing Department, CINVESTAV-IPN, Mexico City, DF, 07360 MEX email: francisco@cs.cinvestav.mx. A portion of this work was performed while F. Rodríguez-Henríquez was visiting University of Waterloo.

Tadanori Teruya is with the Research Institute for Secure Systems, National Institute of Advanced Industrial Science and Technology, Japan. (Part of this work was done while the author was a researcher at the University of Tsukuba)

are classified according to the structure of their underlying groups. When $\mathbb{G}_1 = \mathbb{G}_2$, the pairing is said to be of Type 1. Otherwise, if $\mathbb{G}_1 \neq \mathbb{G}_2$, and no efficient computable homomorphism to map elements between these two groups is known, the pairing is said to be of Type 3 [21].

In 2000, Sakai-Ohgishi-Kasahara [39], and Joux [29], independently discovered constructive cryptographic properties of bilinear pairings. Since then, several researchers have devised ingenious algorithmic improvements that have yielded substantial savings in pairing computations. In the past few years alone, the speed records for computing pairings in hardware and software platforms at or around the 128-bit security level, were broken several times [3], [52]. Furthermore, the timings reported in [2], [15], [44] for computing pairings at the 192-bit and 256-bit levels of security are reasonably fast for a wide range of protocols running on high-end processors.

As a consequence of this continued reduction in the timing required to calculate a pairing, the computation of the other customary cryptographic primitives included in pairing-based protocols have acquired a renewed importance. Examples of these primitives are, fixed-point and variable-point scalar multiplication for elliptic curves defined over finite extension fields, the mapping of arbitrary strings to a random point in those elliptic curves, exponentiation in field extensions, point compression, point membership in a given subgroup, among others. Moreover, as pointed out by Scott [44] and some other authors (see for example [16]), a good number of pairing-based protocols admit further optimizations, such as a faster pairing computation when one of its two arguments is fixed, and a specialized computation of products of pairings.

Since the functionality and security proofs of a scheme tend to be simpler, the vast majority of pairing-based protocols reported so far use Type 1 pairings (see [17] for a comprehensive list). However, Type 1 pairings do not scale up well when going from medium to high security levels. Moreover, it is usually possible to convert a Type 1 pairing-based protocol into a Type 3 pairing one, although it becomes uncertain if the security guarantees would get compromised or not by making this change [12]. As an additional bonus, the migration to Type 3 pairings can sometimes simplify the protocol's complexity [37] and improve its performance [44].

As of today, a relatively small number of works have examined the complexity and overall computational weight of non-pairing cryptographic primitives in a given protocol [10], [11], [13], [40], [44]. In [13], the authors presented a high-level analysis of the computational expenses associated to several pairing-based building blocks. In [11], Chatterjee *et al.* carefully estimated the relatively cost of the main cryptographic building blocks of two pairing-based aggregate signature schemes. However, the work in [11] preceded some of the important pairing computation improvements mentioned above. For example, the authors in [11] estimated that the cost of computing a single pairing was roughly the same as 15,000 field multiplications. Nevertheless, the optimal ate pairing computation reported in [3] required a little more than 10,000 field multiplications. Hence, we believe that it is worthwhile to revisit the authors' conclusions on the relative cost of the different protocol components.

Attribute-based encryption. In 2004, Sahai and Waters presented in [38] a novel encryption scheme where an entity must possess a minimum number of attributes to be declared valid. This concept can be applied to a class of protocols called Attribute-Based Encryption (ABE).

The idea of having a variable number of attributes as an identity comes from the need of sharing a secret in an orderly manner. For instance, in the application of this scheme to health record protection, a patient may be transported to a different hospital, and a different general practitioner may need to read some, but not all, the details of the patient's confidential health record.

For a concrete example of this situation, let us suppose that the following set of attributes have been defined in a hospital: "Cardiologist Hospital", "Cardiologist Surgeon", "Anesthesiologist", "Technician", "Patient". If the access policy of a patient's health record is defined so that it can only be read by anesthesiologists and technicians affiliated to the cardiologist hospital, and at the same time the patient's cardiologist surgeon and the patient can read it at any time, then the following boolean expression would capture that policy:

$$(CardiologistSurgeon \ \textbf{OR} \ Patient) \ \ \textbf{OR} \ \ [(Anesthesiologist \ \textbf{OR} \ Technician) \ \ \textbf{AND} \ Cardiologist \ Hospital]$$

The medical record can also be associated with other attributes such as *electrocardiogram* or *surgery status report*. The access to this document will only be granted to those private key holders having enough attributes to satisfy the access policy.

The attributes that are initially described as alphanumeric strings can be uniquely mapped to group elements by means of a special hash function that maps arbitrary strings to elements in either the group $\mathbb{G}_1$ or $\mathbb{G}_2$. As it was described above, the access policy is specified through a boolean formula, which is transformed to an access structure that can be implemented using a linear secret-sharing scheme (LSSS) as described by Waters in [51].

Several implementations of different ABE protocol's variants have been reported in [7], [40], [44]. In [7], Bethencourt *et al.* implemented for the first time a Type 1 pairing version of this protocol using a supersingular elliptic curve that provided around 70 bits of security. Scott presented in [44] a Type 3 pairing version of the ABE scheme at the 128-, 192- and 256-bit levels of security, where the protocol's attributes were all defined in the group $\mathbb{G}_2$. Finally, the authors of [40] reported a full implementation of the ABE protocol running over mobile device platforms equipped with the latest models of the ARM Cortex family of processors and the vectorized set of instructions NEON.

Our Contributions. By carefully considering practical applications and using the most efficient known algorithms, we provide a comparison of the computational costs associated to the main cryptographic blocks included in customary pairing-based protocols. To this end, we carefully crafted a software library that supports several types of elliptic curve scalar multiplications classified according to the size of the scalar, cyclic group membership of the points (either $\mathbb{G}_1$ or $\mathbb{G}_2$), and the type of the point (either fixed or variable). Using our software, single pairing and multi-pairing computations achieve state-of-the-art time performance at the 126-bit security level.[1]

Furthermore, we reformulate Waters' attribute-based protocol from its original Type 1 setting to a Type 3 one, where the attributes can be defined either in the group $\mathbb{G}_1$, or in the group $\mathbb{G}_2$. We report efficient implementations of the setup, encryption, key generation, decryption and delegation primitives of these two protocol versions.

The remainder of this paper is organized as follows. In Section II relevant mathematical concepts and definitions used throughout this paper are given. Sections III and IV present several implementation notes about the best methods for computing auxiliary building blocks for pairing-based protocols and the computation of bilinear pairings in different settings, respectively. Section V describes a Type 3 version of Water's attribute-based encryption scheme [51]. The computational timings achieved by our software are reported in Section VI, and conclusions are drawn in Section VII.

[1] An open source code for benchmarking our software library is available at: http://sandia.cs.cinvestav.mx/Site/CPABE

II. Mathematical background

A. Elliptic curves: basic definitions

Let p be a prime number, and $\mathbb{F}_p$ a finite field of integers modulo p, with $p \neq 2, 3$. Then, the Weierstrass equation $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ defines an elliptic curve E, provided that the constants $a, b \in \mathbb{F}_p$ are chosen such that $4a^3 + 27b^2 \not\equiv 0 \bmod p$. The $\mathbb{F}_p$-rational points $E(\mathbb{F}_p)$ are defined as the set of points $(x, y) \in \mathbb{F}_p$ that satisfy the above equation, together with the point at infinity denoted by $\mathcal{O}$. It is known that $E(\mathbb{F}_p)$ forms an additive Abelian group with respect to the elliptic point addition operation [50]. The j-invariant $j(E)$ of an elliptic curve E is given as: $j(E) = -1728\frac{4a^3}{4a^3+27b^2}$. Any two non-isomorphic curves defined over an extension field $\mathbb{F}_{p^m}$, $m \in \mathbb{Z}^+$ having the same j-invariant are called *twists* of each other. The number of points in $E(\mathbb{F}_p)$ is denoted as $\#E(\mathbb{F}_p)$, and the integer $t = p + 1 - \#E(\mathbb{F}_p)$, known as the trace of Frobenius, satisfies $|t| \leq 2\sqrt{p}$.

Let r be a large prime with $r \mid \#E(\mathbb{F}_p)$ and $\gcd(r, p) = 1$. The embedding degree k is the smallest positive integer such that $r \mid (p^k - 1)$. We will assume that $k > 1$ is even, which implies that the torsion subgroup $E[r]$ is a proper subset of $E(\mathbb{F}_{p^k})$. For embedding degrees of the form $k = 2^i 3^j$, with $i, j > 0$, there exist curves $\tilde{E}$ defined over $\mathbb{F}_{p^d}$, with $d = \frac{k}{e}$ and $e \in \{2, 3, 4, 6\}$ that are isomorphic to $E(\mathbb{F}_{p^k})$. Let P be a point in $E(\mathbb{F}_p)$ with prime order r (so, $[r]P = \mathcal{O}$). The cyclic subgroup of $E(\mathbb{F}_p)$ generated by P is denoted as, $\langle P \rangle = \{\mathcal{O}, P, [2]P, [3]P, \ldots, [r-1]P\}$.

The elliptic curve scalar multiplication operation, or scalar multiplication for short, computes the multiple $R = [n]P$, with $n \in \mathbb{Z}_r$, which corresponds to the point resulting of adding P to itself $n - 1$ times. The average cost of computing $[n]P$ by a random scalar n using the customary double-and-add method is $\ell D + \frac{\ell}{2}A$, where ℓ is the size in bits of the subgroup, that is to say, $\ell = |r| = \lceil \log_2(r) \rceil$, D is the cost of doubling a point (i.e. the operation of computing $R = 2S = S + S$, with $S \in \langle P \rangle$) and A is the cost of a point addition (i.e. the operation of computing $R = S + T$, with $S, T \in \langle P \rangle$).

Given r and $P, R \in \langle P \rangle$, the Elliptic Curve Discrete Logarithm Problem (ECDLP) consists of finding the unique integer $n \in [0, r - 1]$, such that $R = [n]P$ holds. It is widely believed that for a sensibly-chosen elliptic curve, the complexity of solving the ECDLP needs $O(\sqrt{r})$ steps, roughly equivalent to $2^{\frac{\ell}{2}}$ bit operations [18].

In practice, since field inversion is considerably more expensive than field multiplication, it becomes advantageous to represent points using a projective coordinate system. For example, Jacobian projective coordinates represent an affine point $P = (x, y)$ as the triplet (X, Y, Z), where $x = \frac{X}{Z^2}$ and $y = \frac{Y}{Z^3}$, with $Z \neq 0$.

B. Barreto-Naehrig elliptic curves

Barreto-Naehrig (BN) elliptic curves were presented in [5], as a family of elliptic curves with embedding degree $k = 12$ defined by the equation $E/\mathbb{F}_p : y^2 = x^3 + b, b \neq 0$, where the prime p, the group order $r = \#E(\mathbb{F}_p)$, and the trace of Frobenius t are parametrized as,

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

and where $x \in \mathbb{Z}$, is an arbitrary integer, known as the BN parameter, that must be selected so that $p = p(x)$ and $r = r(x)$, are both 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 $\mathbb{F}_{p^2}$. Notice that for BN curves, $j(E) = 0$, and that all the nonzero points in $E(\mathbb{F}_p)$ have prime order r. From Eq. (1), it follows that choosing a BN parameter x of around 64-bit length such that p and r are prime numbers, implies $\ell = |r| \approx 256$ bits. Consequently, the security level achieved for such a BN curve would be of approximately 128 bits.

C. Bilinear pairings

The reduced Tate pairing on ordinary elliptic curves maps two linearly independent rational points defined over the order-r groups $\mathbb{G}_2, \mathbb{G}_1 \subseteq E(\mathbb{F}_{p^k})$ to the group of r-th roots of unity of the finite field $\mathbb{F}_{p^k}$. In practice the Tate pairing is computed using the iterative algorithm proposed by Victor Miller in 1986 [32], [33] that produces a result f that belongs to the quotient group $\mathbb{F}^*_{p^k}/(\mathbb{F}^*_{p^k})^r$. In order to obtain a unique representative useful for cryptographic purposes, a final exponentiation step consisting of raising f to the power $\frac{p^k-1}{r}$ must be performed [14], [50].

The BN-curve version of the optimal ate pairing introduced by Vercauteren in [49], is defined as follows. Let $\pi : (x, y) \mapsto (x^p, y^p)$ be the p-th power Frobenius endomorphism. Let $\mathbb{G}_1 = \{P \in E[r] : \pi(P) = P\} = E(\mathbb{F}_p)[r]$, be $\mathbb{G}_1$ is the 1-eigenspace of π acting on $E[r]$. Let

$\Psi : \tilde{E} \to E$ 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 π acting on $E[r]$. Let $\mathbb{G}_T$ denote the order-r subgroup of $\mathbb{F}^*_{p^{12}}$. The optimal ate pairing studied in this paper is defined as the non-degenerate map $\hat{a}_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$, defined as,

$$\begin{array}{rcl} \hat{a}_{\text{opt}} : \mathbb{G}_2 \times \mathbb{G}_1 & \longrightarrow & \mathbb{G}_T \\ (Q, P) & \longmapsto & \big(f_{6x+2,Q}(P) \cdot l_{[6x+2]Q,\pi_p(Q)}(P) \cdot \\ & & \qquad l_{[6x+2]Q+\pi_p(Q),-\pi_p^2(Q)}(P)\big)^{\frac{p^{12}-1}{r}}, \end{array}$$

where the Miller function $f_{s,Q}$ of length $s = 6x + 2$, is a normalized rational function [35] in $\bar{\mathbb{F}}_p(E)$ with divisor $(f_{s,Q}) = s(Q) - ([s]Q) - (s-1)(\mathcal{O})$, and where l_{Q_1,Q_2} is the equation of the line corresponding to the addition of $Q_1 \in \mathbb{G}_2$ with $Q_2 \in \mathbb{G}_2$ evaluated at the point $P \in \mathbb{G}_1$ [50].

An efficient algorithm to compute the optimal ate pairing as described above is presented in §IV.

III. Main building blocks for pairing-based cryptography

This section presents implementation notes of some of the most common building blocks used in pairing-based protocols including, map-to-point hash functions to the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, known-point scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$, field exponentiation in $\mathbb{G}_T$ and the signed comb method for unknown-point scalar multiplications.

A. *Hash to* $\mathbb{G}_1$

Algorithm 1 BN-curve indifferentiable Map-to-Point function to $\mathbb{G}_1$ [19]

```
Input: a randomly chosen $\tau \in \mathbb{F}_p$
Output: $P \in E(\mathbb{F}_p)$
$w \leftarrow \sqrt{-3} \cdot \frac{\tau}{1+b+t^2}$
$x_1 \leftarrow \frac{-1+\sqrt{-3}}{2} - \tau \cdot w$
$x_2 \leftarrow -1 - x_1$
$x_3 \leftarrow 1 + 1/w^2$
$r_1, r_2, r_3 \xleftarrow{\$} \mathbb{F}_p^*$
$\alpha \leftarrow \chi_p(r_1^2 \cdot (x_1^3 + b))$
$\beta \leftarrow \chi_p(r_2^2 \cdot (x_2^3 + b))$
$i \leftarrow [(\alpha - 1) \cdot \beta \bmod 3] + 1$
return $P \leftarrow (x_i, \chi_p(r_3^2 \cdot \tau) \cdot \sqrt{x_i^3 + b})$
```

Boneh and Franklin defined in [9] the Map-to-point hash function H_2 to the group $\mathbb{G}_1$ as, $H_2 : \{0,1\}^* \to \mathbb{G}_1^*$, where, as pointed out by the authors, building a direct hash function to

$\mathbb{G}_1^*$ is rather difficult. In the case of BN curves, one can first use a standard cryptographic hash function to map an arbitrary string to a field element $\tau \in \mathbb{F}_p$, followed by the indifferentiable BN hash procedure reported by Fouque & Tibouchi in [19] that is shown in Algorithm 1. Notice that $\chi_p(a)$ denotes the quadratic residuosity test for an arbitrary element $a \in \mathbb{F}_p$. If $\chi_p(a) = 1$, then the element a is said to be a quadratic residue in $\mathbb{F}_p$. The computation of the Legendre symbol can determine $\chi_p(a)$ at a cost similar to that of computing the greatest common divisor of a and p [1].

Furthermore, in Algorithm 1 it is assumed that the prime p of Eq. (1) is selected so that $p \equiv 3 \bmod 4$, which is in fact the most common choice for this parameter [36]. The cost of Algorithm 1 includes the computation of two field inversions, three quadratic residuosity tests, one square root, plus some few field multiplications. The selection $p \equiv 3 \bmod 4$ has the added benefit that the square root computation of step 9 can be performed as the exponentiation $(x_i^3+b)^{\frac{p+1}{4}}$, where $b \in \mathbb{F}_p$ is the BN elliptic curve constant. Notice also that the per-field constants $\sqrt{-3}$ and $\frac{-1+\sqrt{-3}}{2}$, can be precomputed offline.

B. Hash to $\mathbb{G}_2$

In the case of BN pairings, $\mathbb{G}_2$ consists of a group of points of order r that belong to the twist curve $\tilde{E}(\mathbb{F}_{p^2})$. Hence, the standard procedure to hash to $\mathbb{G}_2$, is composed of two phases: first an arbitrary string is hashed to a random point $Q \in \tilde{E}(\mathbb{F}_{p^2})$, followed by the scalar multiplication $[c]Q$, where the cofactor c is defined as, $c = \#\tilde{E}(\mathbb{F}_{p^2})/r$ [46]. By virtue of Lagrange's group theorem, the point $[c]Q$ has order r as desired.

For the first phase, one can use a procedure analogous to the one shown in Algorithm 1, except that in this case all the arithmetic operations must be performed in the quadratic extension $\mathbb{F}_{p^2}$, and the constant b should be replaced by $\bar{\xi} = b/\xi$, where $\xi \in \mathbb{F}_{p^2}$ was defined in section II as the constant of the twist curve $\tilde{E}$. Once again, the square root computation of Step 9 is the single most costly operation. Scott describes in [43] a square root procedure that has a computational cost of just two square roots, one quadratic residue test and one field inversion over $\mathbb{F}_p$ and that is recommended in [1] as the best approach when $p \equiv 3 \bmod 4$. Once a random point $Q \in \tilde{E}(\mathbb{F}_{p^2})$ has been obtained, a scalar multiplication by the cofactor c must be performed. This last step can be accomplished efficiently using the lattice basis reduction approach described in [20], at a cost of one scalar multiplication $[x]Q$, where x is the BN parameter of Eq. (1), plus four point additions and one point doubling.

C. Unknown-point scalar multiplication in $\mathbb{G}_1$

In the following, we discuss how to compute the scalar multiplication $R = [n]P$, with $P \in \mathbb{G}_1$, $|n| \in \mathbb{Z}_r$ and when the point P is not known in advanced. This case will be referred in the remainder of this paper as the *unknown point* scalar multiplication scenario.

The customary method to speed up this operation as described in [28], reduces the Hamming weight of the scalar n by representing it in its w-non-adjacent form (w-NAF) as, $n = \sum_{i=0}^{\ell-1} n_i 2^i$, where each non-zero n_i coefficient is odd, less than 2^{w-1}, and at most one of any w consecutive digits is non-zero, with $n_{\ell-1} \neq 0$, where the length ℓ is at most one bit larger than the bit size of the scalar n. The estimated cost of the scalar multiplication is, $\ell D + \frac{\ell}{\omega+1} A$, plus the expense of precomputing the multiples $P_i = [i]P$, for $i \in \{1, 3, \ldots, 2^{w-1} - 1\}$. Roughly speaking, the cost of the precomputation step is of about one doubling and $(2^{w-2} - 1)$ point additions along with the storage of 2^{w-2} points. The scalar multiplication in $\mathbb{G}_1$ can be further accelerated by means of the GLV method as briefly described next.

Gallant *et al.* introduced in [23] a method to speed up the scalar multiplication $[n]P$ for several families of elliptic curves. In particular the method applies for all curves with j-invariant equal to zero, such as the BN curves. The two-dimensional GLV method breaks the scalar n into 2 smaller pieces $n \equiv n_1 + n_2\lambda \pmod r$ for some λ, such that $|n_i| \approx |\sqrt{r}|$. In the case that there exists an efficient endomorphism $\psi(P) = [\lambda]P$, the scalar multiplication $[n]P = [n_1]P + [n_2]\psi(P)$ can be efficiently performed by means of simultaneous exponentiation techniques that effectively reduce the number of point doubling computations to a half. In the case of BN curves, the map $\psi : E \to E$ is defined by, $\psi : (x, y) \to (\beta x, y)$, where $\beta \in \mathbb{F}_p$ is an element of order three, *i.e.*, $\beta^3 = 1$, which implies that the computation of the point $\psi(P)$ has a negligible cost of just one field multiplication.

Algorithm 2 shows a procedure that performs a two-dimensional GLV scalar multiplication using $m = 2$, $P_0 = P$ and $P_1 = \psi(P)$. The computational cost of Algorithm 2 is of about $\frac{\ell}{2} D + \frac{\ell}{w+1} A$, plus the cost of the precomputation. An efficient method for obtaining a two-dimensional decomposition of the scalar n is discussed in Appendix A.

Algorithm 2 m-dimensional w-NAF scalar multiplication for BN curves

```
Input: $m \in \{2,4\}$, $w$, $n_i$, points $P_i \in E(\mathbb{F}_{p^s})$, $0 \le i \le m-1$, $s = m/2$.
Output: $Q = [n]P = \sum_{i=0}^{m-1} n_i P_i$
Compute $[j]P_i$ for $j \in \{1,3,5,\ldots 2^{w-1}-1\}$, $0 \le i \le m-1$
Compute the $w$-NAF of each scalar $n_i = \sum_{j=0}^{t-1} n_{i,j} 2^j$, $0 \le i \le m-1$
$Q \leftarrow \infty$
for $j = t-1$ down to $0$ do
    $Q \leftarrow [2]Q$
    for $i = 0$ to $m-1$ do
        if $n_{i,j} > 0$ then
            $Q \leftarrow Q + [n_{i,j}]P_i$
        else
            $Q \leftarrow Q - [n_{i,j}]P_i$
        end if
    end for
end for
return $Q$
```

D. *Unknown-point scalar multiplication in $\mathbb{G}_2$ and exponentiation in $\mathbb{G}_T$*

Using endomorphisms with degree-four characteristic polynomials that operate in the $\mathbb{G}_2$ and $\mathbb{G}_T$ groups, the method presented by Galbraith and Scott in [22] generalizes the GLV approach described above to a four-dimensional version.

In the case of BN curves, the homomorphism that applies for the group $\mathbb{G}_2$ is given as, $\psi^i = \phi\pi^i\phi^{-1}$, where π^i is the p-power Frobenius map on the curve E, and $\phi : \tilde{E}(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^{12}})$, and its inverse map ϕ^{-1}, are the endomorphisms that permit to map a point from the twisted elliptic curve $\tilde{E}$ to E and *vice versa*. On the other hand, the homomorphism that applies for the $\mathbb{G}_T$ group is simply the Frobenius endomorphism, since $p \equiv t-1 \bmod r$, which implies that $f^p = f^{t-1}$, with $f \in \mathbb{G}_T$. For both $\mathbb{G}_2$ and $\mathbb{G}_T$ groups, their homomorphisms satisfy characteristic polynomials of degree four, thus allowing four-dimensional GLV methods of the form,

$$n \equiv n_0 + n_1\lambda + n_2\lambda^2 + n_3\lambda^3 \pmod{r},$$

with $\lambda = t - 1 = 6x^2$ and with $|n_i| \approx |r|/4$, for $i = 0,\ldots,3$.

In the case of the scalar multiplication in $\mathbb{G}_2$, the operation $[n]Q$ with $Q \in \mathbb{G}_2, n \in \mathbb{Z}_r$ can then be accomplished as, $[n]Q = [n_0]Q + [n_1]\psi(Q) + [n_2]\psi^2(Q) + [n_3]\psi^3(Q)$, where the mapping $\psi^i(Q)$ for $i = 0,\ldots,3$, has a negligible computational cost. Notice also that the identity $\psi(Q) = [\lambda]Q = [6x^2]Q$, holds. To perform the scalar multiplication in $\mathbb{G}_2$, Algorithm 2 is invoked with input points $Q, \psi(Q), \psi^2(Q), \psi^3(Q) \in \mathbb{G}_2$. The cost associated to the procedure shown in Algorithm 2 is of, $\frac{\ell}{4}D + \frac{\ell}{w+1}A$, plus the precomputation expense.

Likewise, in the case of the exponentiation in $\mathbb{G}_T$, the operation f^n with $f \in \mathbb{G}_T, n \in \mathbb{Z}_r$ can then be accomplished as, $f^n = f^{n_0} \cdot f^{n_1^p} \cdot f^{n_2^{p^2}} \cdot f^{n_3^{p^3}}$, where the Frobenius mapping f^{p^i} for

$i = 0, \ldots, 3$, has a negligible computational cost as explained for example in [8].

Algorithm 2 can be readily adapted to perform exponentiations in $\mathbb{G}_T$, by receiving as input parameters the field elements $f, f^p, f^{p^2}, f^{p^3} \in \mathbb{F}_{p^{12}}$ and by substituting the point doubling computation of step 5 by a fast Granger-Scott squaring [24], and the point addition computations of steps 8 and 10 by field multiplications over $\mathbb{F}_{p^{12}}$. Notice that the exponentiation in $\mathbb{G}_T$ admits almost-free field multiplicative inversions via conjugations [43]. Hence, the addition-subtraction chains generated by the w-NAF scalar expansions of the scalars in step 2 apply just as well to the exponentiation in this group. Leaving aside the precomputation effort, the cost of this exponentiation is $\frac{\ell}{4}S + \frac{\ell}{w+1}M$, where M, S stand for field multiplications and squarings over $\mathbb{F}_{p^{12}}$, respectively.[2]

An efficient method to compute a four-dimensional decomposition of the scalar n is discussed in Appendix A.

E. Known-point scalar multiplication

The case where one wants to compute the scalar multiplication $R = [n]P$, with $P \in \mathbb{G}_1$, $|n| \in \mathbb{Z}_r$, and when the point P is known in advanced, will be referred in the remainder of this paper as the *known point* scalar multiplication. In this scenario, the scalar multiplication operation can be performed efficiently using the comb method provided that some extra memory is available [28].

Given a window size w and a known-point P, the comb method computes the scalar multiplication $Q = [n]P$ with $|n| \approx |r| = \ell$, by first precomputing for all of the possible bit strings $(a_{w-1}, \ldots, a_0)$ the following 2^w multiples of P,

$$[a_{w-1}, \ldots, a_2, a_1, a0]P = [a_{w-1}2^{(w-1)d}]P + \ldots + [a_2 2^{2d}]P + [a_1 2^d]P + [a_0]P, \qquad (2)$$

Then, the scalar n is scanned column-wise by recoding it into d blocks each one with a bit length of w bits, where $d = \ell/w$, and where w is the selected window size. In order to compute the scalar multiplication, a double & add-always sequential procedure of d iterations is performed. Each one of the d blocks, serves as an index to select the right base point multiple from the precomputed look-up table described above.

[2] A faster squaring method was presented by Karabina in [30]. However, GS scalar decompositions lead to high Hamming-weight exponents, which makes Karabina squaring computation less attractive.

In total, the comb method requires the storage of 2^w points. This storage expense can be reduced to a half by using a signed representation of the scalar n as presented by Hamburg in [27]. The signed representation of the scalar is defined as, $n = \sum_{i=0}^{t-1} \sigma_i 2^i \bmod r$, with $\sigma_i \in [-1, 1]$ and $t = |r|$. Using the identity $\sum_{i=0}^{t-1} 2^i = 2^t - 1$ one has that,

$$\frac{n + 2^t - 1}{2} = \left(\sum_{i=0}^{t-1} \sigma_i 2^i + \sum_{i=0}^{t-1} 2^i \right) /2 = \sum_{i=0}^{t-1} \frac{\sigma_i + 1}{2} 2^i = \sum_{i=0}^{t-1} b_i 2^i \bmod r,$$

where $b_i \in [0, 1]$, which implies that the i-th signed bit of n is given as $\sigma_i = 2b_i - 1$. By exploiting the symmetry of the signed multiples of the point P, this scalar representation reduces the memory expense of the precomputation phase to the cost of storing 2^{w-1} points.

Algorithm 3 shows the computation of the scalar multiplication using the signed comb method.

Algorithm 3 Single table signed comb scalar multiplication

```
Input: Positive integer $n$, point $P \in E(\mathbb{F}_{p^s})$, $s \in \{1, 2\}$, a window
    size $w$, $d = \lceil |r|/w \rceil$
Output: $Q = [n]P$
Precompute the signed multiples $[a_{w-1}, \dots, a_1, a0]P$ for all
    possible strings $(a_{w-1}, \dots, a_0)$
Recode the scalar $n$ column-wise dividing it into $d$, $w-$bit blocks
    $\ell_i$, for $i = 0, \dots, d-1$.
for $i = d-1$ down to $0$ do
    $Q \leftarrow [2]Q$
    if the MSB of $\ell_i = 1$ then
        $Q \leftarrow Q - [\ell_i \& (2^{w-1} - 1)]P$
    else
        $Q \leftarrow Q + [\ell_i]P$
    end if
end for
return $Q$
```

The estimated computational complexity of this scalar multiplication is of $d(A + D)$, where A is the cost of a point addition, and D is the cost of a point doubling. Algorithm 3 can also be adapted for computing exponentiations in the $\mathbb{G}_T$ group, using the Granger-Scott squaring, and regular field multiplications/divisions over $\mathbb{F}_{p^{12}}$.[3] Alternative optimizations include the usage of multiple tables and/or combining an m-dimensional GLV with the comb method, at the price of adding more precomputation tables. In this work none of these additional optimizations were considered.

In the case that the storage required by the comb scalar multiplication or exponentiation becomes excessive, one can optionally use a wNAF recoding of the scalar/exponent with $w = 7$, which obtains competitive computational timings.

[3] Field division of two elements $a, b \in \mathbb{F}_{p^{12}}$, is implemented as, $a/b = a \cdot \bar{b}$, where $\bar{b}$ is the conjugate of b in the field $\mathbb{F}_{p^{12}}$ that is seen as the quadratic extension of the field $\mathbb{F}_{p^6}$.

IV. Efficient computation of pairings

As was discussed in § II-C, the computation of the reduced Tate pairing consists of two main steps: the Miller loop and the final exponentiation. In this Section, several relevant algorithmic aspects of the computation of the Miller loop and the final exponentiation steps are briefly reviewed. Then, a specialized procedure for the efficient computation of a product of pairings is presented.

A. Single Pairing computation

Algorithm 4 Optimal ate pairing as computed in [3]

```
Input: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$
Output: $f = a_{\text{opt}}(Q, P)$
 1: $f \leftarrow 1$, $T \leftarrow Q$, $s \leftarrow$ abs$(6x+2)$
 2: Write $s$ as, $s = \sum_{i=0}^{m-1} s_i 2^i$ with $s_i \in \{-1, 0, 1\}$
 3: for $i = m-2$ down to $0$ do
 4:     $f \leftarrow f^2 \cdot l_{T,T}(P)$, $T \leftarrow 2T$
 5:     if $s_i = 1$ then
 6:         $f \leftarrow f \cdot l_{T,Q}(P)$, $T \leftarrow T + Q$
 7:     else if $s_i = -1$ then
 8:         $f \leftarrow f \cdot l_{T,-Q}(P)$, $T \leftarrow T - Q$
 9:     end if
10: end for
11: $f \leftarrow f^{p^6}$
12: $R \leftarrow \pi(Q)$; $f \leftarrow f \cdot l_{-T,R}(P)$; $T \leftarrow T + R$
13: $R \leftarrow \pi^2(Q)$; $f \leftarrow f \cdot l_{-T,-R}(P)$; $T \leftarrow T - R$
14: $f^{(p^{12}-1)/r}$
15: return $f$
```

Algorithm 4 shows a slightly adapted version of the optimal ate pairing over BN curves proposed in [3].

The most costly operations in the main loop of Algorithm 4 are the evaluation of the tangent line $l_{T,T}$ and the doubling of the point T in step 4, as well as the evaluation of the line through T and Q, and the computation of the point addition $T + P$ in steps 6 and 8. Additionally, these steps require the multiplication of the Miller variable f by the lines $l_{T,T}$ and $l_{T,Q}$, respectively.

The final exponentiation (see step 14 in Algorithm 4), can be accomplished using the lattice basis reduction approach described in [20] at a cost of 3 Frobenius operators, 3 exponentiations by the BN parameter x, 12 multiplications, 3 squarings and one inversion in $\mathbb{F}_{p^{12}}$.

The arithmetic operations for computing the Miller loop and the final exponentiation are performed using field towering techniques, where the field $\mathbb{F}_{p^{12}}$ is represented as the tower,

$$\mathbb{F}_p \subset \mathbb{F}_{p^2} \subset \mathbb{F}_{p^6} \subset \mathbb{F}_{p^{12}}.$$

Furthermore, taking advantage of the sextic twist of the BN curves, it results convenient to specify the cost of the pairing computation in terms of the number of field arithmetic operations performed in the quadratic extension field $\mathbb{F}_{p^2}$, being field multiplication and squaring the single two most dominant operations.

The Montgomery multiplier procedure can compute a field multiplication over the quadratic extension field $\mathbb{F}_{p^2}$ by performing two main steps, the integer multiplication of two 256-bit integers (producing a 512-bit integer), denoted as m_E, followed by the Montgomery reduction from a 512-bit integer to a 256-bit integer, an operation denoted as r_E.

The lazy reduction technique opportunistically exploits the homomorphic properties of modular reduction, namely, $((a \bmod p)+(b \bmod p)) \bmod p = (a+b) \bmod p$. By carefully selecting prime numbers p with bitlength slightly less than 256 bits, lazy reduction can significantly accelerate the field arithmetic computations. Under this condition, it was shown in [8] that field multiplication and squaring over $\mathbb{F}_{p^2}$ can be computed at a cost of $3m_E + 2r_E$ and $2m_E + 2r_E$, respectively. This technique can also be applied to the other extension fields included in the field towering described above. We refer the reader to [3] for further details of how to apply the lazy reduction to accelerate the arithmetic operations in other field extensions.

Notice that the main loop of Algorithm 4 performs a total of $|s|$ iterations, where $s = 6x + 2$. Choosing the BN parameter as $x = -2^{62} - 2^{55} - 1$ [36], implies that the signed binary representation of s has a length of 65 bits and a Hamming weight of 5.[4] Using that standard choice for the BN parameter, the total costs of the Miller Loop and the final exponentiation can be estimated as [34], M. L. $= 6785m_E + 3022r_E$ and F. E. $= 3526m_E + 1932r_E$, respectively.[5]

B. Fixed argument pairing computation

If the point $Q \in \mathbb{G}_2$ is known in advance then, as was discussed in [43], a speedup can be attained in the computation of the optimal ate pairing at the price of more storage. If the point Q is fixed and known, the line functions of the pairing algorithm can be partially precomputed off-line. At running time, the stored values of the lines are recovered and evaluated at the variable

[4] This BN parameter selection can be subject to small subgroup attacks in $\mathbb{G}_T$, as discussed in [45, section 8.3], where an alternative BN parameter choice was suggested.

[5] An optimized version of our library that was especially tailored for the newest Intel processor Haswell, achieves the fastest pairing computation time reported in software platforms [34].

point P. If Algorithm 4 is executed with the BN parameter choice previously mentioned, then 70 elements in $\mathbb{F}_{p^6}$ per pairing must be stored, which corresponds to 64 tangent lines that are required at each iteration in step 4, plus 6 lines through the points T and Q that are used in the case that $s_i = 1$, in steps 6 and 8. If more than a pairing computation is required, this improvement can be combined with the multi-pairing setting presented next.

C. Computation of products of pairings

In many protocols, some of their computations include a product of pairings. In this case, 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 the n pairing products by $n-1$ point additions and a single pairing using the identity,

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

which saves a significant amount of operations, namely, $n-1$ pairings, and $n-1$ multiplications in $\mathbb{G}_T$. 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 that product by exploiting similar strategies as the ones proposed in the multi-exponentiation setting. These techniques have already been discussed by Solinas [48], Scott [42], and Granger and Smart in [25]. In essence, in a multi-pairing computation not only the costly final exponentiation step can be shared, but also, one can share both the accumulator f, and the squaring computations performed in the step 4 of Algorithm 4.

Algorithm 5 presents an explicit multi-pairing version of Algorithm 4 that computes the product of n optimal ate pairings specialized for BN curves.

Note that during a protocol flow, most of the pairing input points are either known in advance, or they are the result of a scalar multiplication, which for efficiency reasons are represented in Jacobian projective coordinates. In that case, a *multi-normalization*, *i.e.* the operation of performing the conversion from Jacobian projective to affine coordinates, of n points can be performed efficiently using Montgomery's simultaneous inversion trick [28] at a cost of $(6n-3)$ multiplications, n squarings and one inversion.

Algorithm 5 Explicit multi-pairing version of Algorithm 4

```
Input: $P_1, P_2 \dots P_n \in \mathbb{G}_1$, $Q_1, Q_2 \dots Q_n \in \mathbb{G}_2$
Output: $f = \prod_{i=1}^{n} e(Q_i, P_i)$
Write $s$ in binary signed form, $s = \sum_{i=0}^{m-1}$ with $s_i \in \{-1, 0, 1\}$
$f \leftarrow 1$, $s \leftarrow$ abs$(6x + 2)$
for $j = 1$ to $n$ do
    $T_j \leftarrow Q_j$
end for
for $i = m - 2$ down to $0$ do
    $f \leftarrow f^2$
    for $j \leftarrow 1$ to $n$ do
        $f \leftarrow f \cdot L_{T_j,T_j}(P_j)$, $T_j \leftarrow 2T_j$
        if $s_i = 1$ then
            $f \leftarrow f \cdot L_{T_j,Q_j}(P_j)$, $T_j \leftarrow T_j + Q_j$
        else if $s_i = -1$ then
            $f \leftarrow f \cdot L_{T_j,-Q_j}(P_j)$, $T_j \leftarrow T_j - Q_j$
        end if
    end for
end for
$f \leftarrow f^{p^6}$
for $j = 1$ to $n$ do
    $R \leftarrow \pi(Q_j)$; $f \leftarrow f \cdot L_{-T_j,R}(P_j)$; $T_j \leftarrow T_j + R$
    $R \leftarrow \pi^2(Q_j)$; $f \leftarrow f \cdot L_{-T_j,-R}(P_j)$; $T_j \leftarrow T_j - R$
end for
$f^{(p^{12}-1)/r}$
return $f$
```

V. CASE STUDY: ATTRIBUTE-BASED ENCRYPTION

In an ABE protocol, a participant encrypts data that can only be decrypted by users able to satisfy an access policy previously agreed. The access policy is specified as a boolean formula over a set of *attributes*.

The boolean formula describing the access policy can be represented by an access structure using a linear secret-sharing scheme (LSSS) [51], which in turn can be implemented efficiently using the approach of Liu and Cao reported in [31]. The description of the Liu and Cao conversion algorithm is presented in Appendix B, where the simple access policy mentioned in §I is used to illustrate the process of converting a control access policy specified as a boolean formula to an LSSS matrix. During the setup phase of the protocol, the attributes, initially specified as arbitrary strings, are mapped to points in the group $\mathbb{G}_1$ or in the group $\mathbb{G}_2$.

An ABE scheme consists of five main primitives as described next [7],

1) **Setup(**κ**,** U**)**. The input parameters of this primitive are the security parameter κ and the universe of attributes U. The primitive generates a master key MK along with the domain parameters PK.
2) **Encryption(**$PK, M, \mathbb{A}$**)**. This primitive takes as input the set of domain parameters PK, the message M and the access policy $\mathbb{A}$ specified as a boolean formula whose operands are a subset of the universe of attributes. Then, this primitive encrypts M as the ciphertext

C_T in such a way that only those users that have the set of attributes required to satisfy the access policy $\mathbb{A}$, will be able to decrypt it. It is assumed that the ciphertext and the access policy $\mathbb{A}$ must be transmitted together as a pair.

3) **Key generation(**MK, S**)**. This primitive takes as input the master key MK along with a set of attributes S. Then, this primitive generates a private key SK with the prescribed set of attributes. Usually, this primitive is executed by a "trusted third party" that has the crucial role of generating for each one of the participants a customized private key with specific access privileges.
4) **Decryption(**PK, C_T, SK**)**. This primitive takes as input the domain parameters PK along with the ciphertext C_T and its corresponding access policy $\mathbb{A}$, and the private key SK, which contains the set of attributes S. Only in the case that the set of attributes S satisfies the policy $\mathbb{A}$, this primitive will be able to recover the message M from the ciphertext C_T.
5) **Delegate(**$SK, \tilde{S}$**)**. This primitive takes as input a private key SK with some set of attributes S and a subset $\tilde{S} \subset S$. It outputs a valid private key $\tilde{SK}$ for the subset of attributes $\tilde{S}$.

Converting a Type 1 pairing-based protocol into a Type 3 setting has the challenge of selecting the groups where the pairing inputs belong in an accurate way from the security and performance points of view. In the case of an ABE protocol, if the access policy attributes are defined in $\mathbb{G}_1$, then the computation of the key generation and encryption primitives can be boosted via multi-pairing computation. Moreover, the size of the ciphertext, which is proportional to the number of attributes and their bit-length, would be smaller. On the other hand, the scenario where the attributes are defined in $\mathbb{G}_2$ will benefit from a fast decryption algorithm.

In the following we describe the details of the ABE protocol in the $\mathbb{G}_1$ setting. The ABE protocol in the $\mathbb{G}_2$ setting is similar and therefore its description is omitted.

A. *Attribute-based encryption with variable attributes defined in* $\mathbb{G}_1$

In this subsection, an asymmetric setting of the ABE protocol with a variable number of attributes mapped to elements in $\mathbb{G}_1$, is presented. Explicit algorithms for the five main primitives of the ABE protocol can be found in Appendix C.

1) Setup: Let us assume that a set of attributes U composed of N distinct strings, has been previously defined. Two generating points $P \in \mathbb{G}_1$, and $Q \in \mathbb{G}_2$ are randomly selected so that $|\mathbb{G}_1| = |\mathbb{G}_2| = r$, with r a large prime whose size is defined by the security parameter κ. Then, the randomly chosen scalars α, $\delta \in \mathbb{F}_r$, are used in Algorithm 7 to compute the multiples, $P_\alpha = [\alpha]P$ and $P_\delta = [\delta]P$, along with the pairing $\gamma = e(P,Q)^\alpha$. Choose a hash function H_1 which hashes an element in $\mathbb{G}_T$ to a string of m bits.[6] All the attribute strings are mapped to points in $\mathbb{G}_1$ by means of a special hash function $H_2(\cdot)$ that maps arbitrary strings to the group elements $\{\mathcal{H}_1, \ldots \mathcal{H}_N\}$ belonging to $\mathbb{G}_1$. The public domain parameters computed by Algorithm 7 are $\{P, Q, P_\delta, \gamma\}, \{\mathcal{H}_1, \ldots \mathcal{H}_N\}$, and the secret master key is $\{P_\alpha\}$.

2) Encryption: Let us recall that the access policy $\mathbb{A}$ is initially specified as a boolean formula over a subset of attributes using the **AND** and **OR** operators. Let us assume that the number of distinct attributes in that boolean formula is u, with $u \leq N$.

The first step of this primitive uses Algorithm 6 presented in Appendix B, to convert the boolean formula describing the access policy into a LSSS matrix $\mathcal{S} \in \mathbb{F}_r$ of size $u \times t$, along with a function ρ that associates rows of $\mathcal{S}$ to attributes in $\mathcal{H}$, namely, $I \subset \{1, 2, \ldots, u\}$ is defined as $I = \{i : \rho(i) \in \mathcal{H}\}$.

Next, the encryption exponent $s \in \mathbb{F}_r$ is picked at random, along with $t-1$ random secrets $y_i \in \mathbb{F}_r$ to form the column vector $\bar{\mathbf{u}} = (s, y_2, \ldots, y_t)$ that is used to compute the secret share vector $\bar{\lambda}$ as,

$$\bar{\lambda} = \mathcal{S}\bar{\mathbf{u}}. \tag{3}$$

Thereafter, a total of u secrets in $\mathbb{F}_r$ are randomly selected to form the vector $\bar{\mathbf{x}} = (x_1, \ldots, x_u)$. The message M is then hidden as as $C = M \oplus H_1(\gamma^s)$, whereas the encryption exponent is protected as, $C_d = [s]Q$. All the coefficients of the secret share vector $\bar{\lambda}$ and the vector x are hidden as, $C_i \leftarrow [\lambda_i]P_\delta - [x_i]\mathcal{H}_{\rho(i)}$ and $D_i \leftarrow [x_i]Q$, for $i = 1, \ldots, u$. Algorithm 8 performs the procedure just described given as output the ciphertext, $C_T = \{\mathcal{S}, C, C_d, (C_1, D_1), \ldots, (C_u, D_u)\}$.

3) Key Generation: Let us denote by $\mathcal{H}$ the entity's set of $v_\mathcal{H}$ attributes. A field element $\tau \in \mathbb{F}_r$ is randomly selected to compute $K = P_\alpha + [\tau]P_\delta$, and $L = [\tau]Q$. For each of the entity's attributes $\mathcal{H}_i \in \mathcal{H}, i = 1, \ldots, v_\mathcal{H}$, the points $K_i = [\tau]\mathcal{H}_i$, are computed. Algorithm 9 performs the procedure just described to generate the private key SK as, $SK = \{K, L, K_1, \ldots, K_{v_\mathcal{H}}\}$.

4) Decryption: Algorithm 10 takes as inputs the ciphertext C_T with its corresponding access structure represented by the LSSS matrix $\mathcal{S} \in \mathbb{F}_r$, and the private key SK associated to a set of attributes $\mathcal{H}$. Let us suppose that S satisfies the access structure.

[6] In our implementation we used $m = 256$ bits.

As a first step, the $u \times t$ LSSS matrix $\mathcal{S}$ from the ciphertext C_T, is reduced to a square matrix by removing the rows and columns that are unrelated to the set of attributes $\mathcal{H}$. We denote the resulting reduced $v \times v$ matrix as $\tilde{\mathcal{S}} \in \mathbb{F}_r$, with $v \leq u$. Let ρ be a function that associates rows of $\tilde{\mathcal{S}}$ to attributes in $\mathcal{H}$ as, $I = \{i : \rho(i) \in \mathcal{H}\}$ with $I \subset \{1, 2, \ldots, v\}$.

Let us recall that the secret share vector $\bar{\lambda}$ was computed in Eq. (3) as, $\bar{\lambda} = \mathcal{S}\bar{\mathbf{u}}$. For the sake of simplicity assume that the size of the vectors $\bar{\mathbf{u}}$ and $\bar{\lambda}$ is equal to v. Then, it follows that, $\tilde{\mathcal{S}}^{-1}\bar{\lambda} = \bar{\mathbf{u}}$, which implies that,

$$\Omega \cdot \bar{\lambda} = \sum_{i \in I} \omega_i \lambda_i = \Delta \cdot s, \tag{4}$$

where $\Omega = \{\omega_i \in \mathbb{Z}\}_{i \in \mathcal{I}}$ is the first row of the inverse of the matrix $\tilde{\mathcal{S}}$, and Δ is the determinant of the reduced matrix $\tilde{\mathcal{S}}$ in $\mathbb{F}_r$ [44]. Having computed the vector Ω as defined above, Algorithm 10 recovers the plaintext M as,

$$M = C \oplus H_1 \left(\left(e(C_d, -[\Delta]K) \cdot e\left(L, \sum_{i \in \mathcal{I}} [\omega_i] C_i\right) \cdot \prod_{i \in \mathcal{I}} e(D_i, [\omega_i] K_{\rho(i)}) \right)^{\frac{1}{\Delta}} \right) \tag{5}$$

Notice that keeping the determinant Δ in Eq. (4) helps us to guarantee low size constants in the vector Ω. This allows us to compute v short scalar multiplications in $\mathbb{G}_1$ (as opposed to v full scalar multiplications), with an associated penalty in Eq. (5) of only one single exponentiation in $\mathbb{G}_T$.

In our experiments we have always observed that $\mid \omega_i \mid < 20$ bits. In this way, the scalar multiplications where these constants are involved can be computed using a simple and fast 2-NAF method, an operation that in the remainder of this paper is called *short scalar multiplication*.

5) Delegate: Given a set of attributes $\mathcal{H}$ matching a given policy, Algorithm 11 takes as input a subset $\tilde{\mathcal{H}}$ of $v_{\tilde{\mathcal{H}}}$ attributes, such that $\tilde{\mathcal{H}} \subseteq \mathcal{H}$, and a private key SK. A private key $\tilde{SK}$ that contains the set of attributes $\tilde{\mathcal{H}}$ is produced as output. Notice that the delegation step is essentially a copy of the Key Generation procedure with a different random secret τ, denoted as τ'.

B. Operation count

For the sake of simplicity let us assume that the number of attributes specified in the control access policies of the setup and the encryption primitives are both equal to v. Then, Table I summarizes the number of operations in the $\mathbb{G}_1$ and $\mathbb{G}_2$ settings associated to the three main ABE primitives, namely, encryption, key generation and decryption.

It is noticed that for those applications where the attributes are not known in advance, both the encryption and key generation primitives require v applications of the *Map-To-Point* hash function.

Primitive	Scalar Mult. in $\mathbb{G}_1$		Scalar Mult. in $\mathbb{G}_2$		Exp. in $\mathbb{G}_T$		Pairing	
	K	**small**	**K**	**small**	**K**	**U**	**F**	**U**
ABE protocol with attributes defined in $\mathbb{G}_1$								
encryption:	$2v$	–	$v+1$	–	1	–	–	–
key generation:	$v+1$	–	1	–	–	–	–	–
decryption $\Delta = 1$	–	$2v$	–	–	–	–	1	$v+1$
decryption $\Delta \neq 1$	1	$2v$	–	–	–	1	1	$v+1$
ABE protocol with attributes defined in $\mathbb{G}_2$								
encryption:	$v+1$	–	$2v$	–	1	–	–	–
key generation:	1	–	$v+1$	–	–	–	–	–
decryption $\Delta = 1$	–	v	–	v	–	–	$v+1$	1
decryption $\Delta \neq 1$	–	v	1	v	1	–	$v+1$	1

TABLE I

ABE protocol operation count with v attributes. **K** stands for known point/element using the signed comb exponentiation with a window size $w = 8$, **small** stands for a scalar multiplication by a *small* scalar, **U** stands for an unknown-point scalar multiplication, unknown-element exponentiation, or unknown-point pairing, **F** stands for a pairing computation with a fixed parameter.

VI. Implementation results

This section presents our main implementation results classified into two subsections. First in §VI-A, the performance timings achieved by all the ABE protocol building blocks, are given. This subsection also includes a performance comparison against recently reported works on scalar multiplication and exponentiation in pairing groups. Then in §VI-B, the timings achieved by our software for computing the ABE protocol in the $\mathbb{G}_1$ and $\mathbb{G}_2$ settings are presented.

All our timings were measured on an Intel Core i7-4770 equipped with the micro-architecture

Haswell running at 3.4GHz with the TurboBoost technology disabled, using a Linux kernel 3.8.0 and GCC version 4.7.3.

A. Cryptographic primitive timings

Table II shows the performance achieved by single/multi pairing computations and the implementation of all the auxiliary functions as they were described in §III, which includes the exponentiation operations in the three pairing groups, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, and the map-to-point hash function. For each operation, its approximate cost in terms of number of field multiplications over $\mathbb{F}_p$ is given.

As shown in Eq. 5, the decryption primitive of the ABE protocol studied in this paper requires a multi-pairing computation that involves at least three pairings. Within a multi-pairing computation, each extra pairing amounts for $483,000$ and $280,000$ clock cycles in the unknown and fixed point scenarios, respectively (see rows three and four of Table II).

The scalar multiplications with small scalars of less than 20 bits in the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ were implemented using a 2-NAF recoding at a cost of about $26,800$ and $61,800$ clock cycles, respectively.

Additionally, Table II also reports a performance comparison of each block against the cost of computing a scalar multiplication with a non-fixed scalar in the group $\mathbb{G}_1$ denoted as $\mathrm{M}_{\mathbb{G}_1}$ for short. This comparison aims to give a quick reference of the relative weight that each of the main cryptographic blocks adds to the total computational complexity of the ABE protocol. For example, Table II reports that one single pairing computation costs approximately 5.95 $\mathrm{M}_{\mathbb{G}_1}$. However, any extra pairing computation in the unknown-point and known point scenarios would cost just 2.48 and 1.44 $\mathrm{M}_{\mathbb{G}_1}$, respectively. Similarly, one can see that the unknown point multiplication in $\mathbb{G}_2$ and exponentiation in $\mathbb{G}_T$ cost about 1.82 and 2.86 $\mathrm{M}_{\mathbb{G}_1}$, respectively.

As shown in Table III, the performance achieved by our library for scalar multiplication and exponentiation in pairing groups, clearly outperforms all the related results recently published. However, it is worth to mention that the library presented in [26] utilized a random prime p to compute all the arithmetic operations. Hence, the comparison against our specialized library is a little unfair in that sense.

Operation	w	Type	Operation count	10^3 Clock cycles	op/M$_{\mathbb{G}_1}$
single pairing	–	U	$10,312m_E + 4,954r_E$	$1,162$	5.95
	–	K	$8,598m_E + 3,652r_E$	980	5.03
one more pairing	–	U	$4,604m_E + 2,301r_E$	483	2.48
	–	K	$3,011m_E + 1,125r_E$	280	1.44
$\mathbb{G}_1$ scalar mult.	3	U	$\approx 1,654m$	195	1.00
	8	K	$\approx 576m$	61	0.31
$\mathbb{G}_2$ scalar mult.	3	U	$\approx 3,036m$	354	1.82
	8	K	$\approx 1,472m$	161	0.83
$\mathbb{G}_T$ Exp.	3	U	$\approx 5,070m$	557	2.86
	8	K	$\approx 2,496m$	260	1.33
Map-To-Point $\mathbb{G}_1$	–	–	$\approx 750m$	72	0.37
Map-To-Point $\mathbb{G}_2$	–	–	$\approx 2,760m$	262	1.34

TABLE II

COMPUTATIONAL COSTS OF THE ABE PROTOCOL BUILDING BLOCKS. TYPE U AND TYPE K STAND FOR THE UNKNOWN-POINT AND KNOWN-POINT SCENARIOS, RESPECTIVELY. m_E, r_E AND m DENOTE 256-BIT INTEGER MULTIPLICATION, 512-BIT MONTGOMERY REDUCTION AND FIELD MULTIPLICATION OVER $\mathbb{F}_p$, RESPECTIVELY. ALL THE SCALAR MULTIPLICATIONS/EXPONENTIATIONS IN THE GROUPS $\mathbb{G}_1, \mathbb{G}_2$ AND $\mathbb{G}_T$, PROCESS THE SCALAR/EXPONENT USING A WINDOW SIZE w AS INDICATED. FINALLY, M$_{\mathbb{G}_1}$ STANDS FOR UNKNOWN SCALAR MULTIPLICATION OVER $\mathbb{G}_1$.

B. Protocol timings

In this subsection we report the ABE protocol implementation timings when the attributes are defined in the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We do not include the setup primitive timings because in most applications this primitive can be performed offline, and hence its influence in the overall performance of the protocol is limited. Likewise, since the delegation timings are essentially the same as the ones of key generation they are not reported either. For comparison purposes, we include the timings reported by Scott in [44].

Table IV shows the timings obtained for the encryption, key generation and decryption primitives in the $\mathbb{G}_1$ and $\mathbb{G}_2$ settings using an access policy of 6 and 20 attributes. In the $\mathbb{G}_2$ setting, a substantial speedup in the decryption step is achieved at the cost of larger precomputation tables and ciphertext sizes. The extra penalty in memory is due to the cost of storing the precomputation of the lines for fixed-point pairing calculations. In contrast, the $\mathbb{G}_1$ setting is significantly more

Op	Type	Scott [41]	Bos *et al.* [10]	Guillevic [26]	This work
		Intel i5-520M	Intel i7-3520	Intel Celeron	Intel i7-4770
		@2.4GHz	@2.9GHz	@2.6GHz	@3.4 GHz
$\mathbb{G}_1$ scalar mult.	U	528	900	1,430	192
	K	168	–	–	60
$\mathbb{G}_2$ scalar mult.	U	1,056	1,800	4,966	367
	K	456	–	–	152
$\mathbb{G}_T$ Exp.	U	2,280	3,100	13,416	558
	K	1,032	–	–	262
single pairing	U	5,568	7,000	13,130	1,162
	K	5,016	–	–	980
one more pairing	U	1,800	–	–	485

TABLE III

Scalar multiplication and exponentiation in pairing groups (all timings are given in 10^3 clock cycles). Type U/K, stand for unknown/known point/element operation.

	10^3 clock cycles					
LSSS ABE Protocol	Attributes in $\mathbb{G}_1$		Attributes in $\mathbb{G}_2$		Scott's Attributes in $\mathbb{G}_2$ [44]	
	Six attributes	Twenty attributes	Six attributes	Twenty attributes	Six attributes	Twenty attributes
encryption	2,384	7,150	2,921	9,129	16,704	23,832
key generation	652	1,699	1,326	3,994	3,408	–
decryption ($\Delta = 1$)	4,606	12,776	3,515	9,528	14,832	31,320
overall cost	7,642	21,625	7,762	22,651	31,083	–
pairing cost	4,378	11,168	3,123	7,043	–	–
Pairing cost (%)	57.3	51.6	40.2	31.1	–	–

TABLE IV

Performance of the ABE protocol primitives (all the timings are given in 10^3 clock cycles)

economical for the computation of the encryption and key generation primitives. The overall result is that both settings achieve roughly the same performance with a 6-attribute access policy, whereas the $\mathbb{G}_1$ setting is 4.6% faster than the $\mathbb{G}_2$ one, when using a 20-attribute access policy.

Table IV also presents the estimated CPU cycles required by Scott's implementation of the ABE scheme studied in this paper. It is noticed that our 6-attribute encryption, key generation and decryption timings presented in Table IV for the $\mathbb{G}_2$ setting, are $5.7, 2.6$ and 4.2 times faster that those reported in [44], respectively, and significantly faster than the ones reported in [7], where a scheme providing just 70 bits of security was reported.

VII. Conclusion

In this paper, we presented two variants of Waters' attribute-based encryption protocol that make use of Type 3 asymmetric pairings allowing a variable number of access policy attributes, which can be defined either in the $\mathbb{G}_1$ or in the $\mathbb{G}_2$ pairing groups. Although both variants achieve roughly the same timing performance, the $\mathbb{G}_1$ setting has smaller memory requirements and produces smaller ciphertexts than the $\mathbb{G}_2$ setting.

Table II presents a performance comparison of several relevant cryptographic blocks against the cost of computing a scalar multiplication with a non fixed scalar in the group $\mathbb{G}_1$. It is interesting to note that after performing the first pairing in a multi-pairing operation, the computational cost of calculating any additional pairing is roughly the same of performing one exponentiation in the $\mathbb{G}_T$ group. Furthermore, the computational cost of unknown point/element exponentiation in the groups $\mathbb{G}_2$, and $\mathbb{G}_T$ is about 1.8 and 2.9 times larger than the cost of one unknown point scalar multiplication in $\mathbb{G}_1$, respectively. Notice also that the map-to-point mapping has a non-negligible cost, especially when performing a mapping to $\mathbb{G}_2$. Hence, this operation should not be overlooked when assessing the computational cost of a pairing-based cryptographic scheme.

As shown in Table IV our timings for the encryption, key generation and decryption primitives when using 6 attributes defined in the group $\mathbb{G}_2$ are $5.7, 2.6$ and 4.2 times faster that those reported in [44], respectively.

As a general conclusion, our study shows that the cost of a single pairing is still the most expensive primitive in a typical attribute-based protocol. However, it is noticed that the costs of the pairing computations reported in Table IV amount for roughly 50% and 35% of the total cost of the ABE protocol in the $\mathbb{G}_1$ and $\mathbb{G}_2$ settings, respectively. This result suggests that the extra cryptographic blocks typically used in pairing-based protocols may contribute to more than half of the total scheme's computational cost.

Future work will attempt a timing-attack protected version of this protocol.

References

[1] G. Adj and F. Rodríguez-Henríquez. Square root computation over even extension fields. Cryptology ePrint Archive, Report 2012/685, 2012. http://eprint.iacr.org/.

[2] 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.

[3] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in cryptology*, EUROCRYPT'11, pages 48–68, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, March, 1986.

[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] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, June, 1996.

[7] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[8] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In M. Joye, A. Miyaji, and A. Otsuka, editors, *Pairing*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.

[9] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001.

[10] J. W. Bos, C. Costello, and M. Naehrig. Exponentiating in pairing groups. Cryptology ePrint Archive, Report 2013/458, 2013. To appear in SAC 2013, available at: http://eprint.iacr.org/.

[11] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes. Comparing two pairing-based aggregate signature schemes. *Des. Codes Cryptography*, 55(2-3):141–167, 2010.

[12] S. Chatterjee, D. Hankerson, and A. Menezes. On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In M. A. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields, WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 114–134. Springer, 2010.

[13] L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *Int. J. Inf. Sec.*, 6(4):213–241, 2007.

[14] H. Cohen and G. Frey. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2006.

[15] C. Costello, K. Lauter, and M. Naehrig. Attractive subfamilies of BLS curves for implementing high-security pairings. In D. J. Bernstein and S. Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011*, volume 7107 of *Lecture Notes in Computer Science*, pages 320–342. Springer, 2011.

[16] 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.

[17] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols : A survey. Cryptology ePrint Archive, Report 2004/064, 2004. http://eprint.iacr.org/.

[18] ECRYPT II. ECRYPT II yearly report on algorithms and keysizes (2011-2012). European Network of Excellence in Cryptology II, August 2013. available at: http://sac2013.irmacs.sfu.ca/slides/s1.pdf.

[19] P.-A. Fouque and M. Tibouchi. Indifferentiable hashing to Barreto-Naehrig curves. In A. Hevia and G. Neven, editors, *Progress in Cryptology - LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2012.

[20] 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*, volume 7118 of *Lecture Notes in Computer Science*, pages 412–430. Springer, 2011.

[21] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156(16):3113–3121, Sept. 2008.

[22] 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. Springer-Verlag, 2008. Pairing 2008.

[23] R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.

[24] R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2010.

[25] R. Granger and N. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006. http://eprint.iacr.org/.

[26] A. Guillevic. Comparing the pairing efficiency over composite-order and prime-order elliptic curves. In M. J. Jacobson Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2013.

[27] M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. http://eprint.iacr.org/.

[28] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[29] A. Joux. A one round protocol for tripartite Diffie-Hellman. In W. Bosma, editor, *Algorithmic Number Theory – ANTS IV*, number 1838 in Lecture Notes in Computer Science, pages 385–394. Springer, 2000.

[30] K. Karabina. Squaring in cyclotomic subgroups. *Math. Comput.*, 82(281), 2013.

[31] Z. Liu and Z. Cao. On efficiently transferring the linear secret-sharing scheme matrix in ciphertext-policy attribute-based encryption, 2010.

[32] V. Miller. Short programs for functions on curves. Available at http://crypto.stanford.edu/miller, 1986.

[33] V. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, Sept. 2004.

[34] S. Mitsunari. A fast implementation of the optimal ate pairing over BN curve on Intel Haswell processor. Cryptology ePrint Archive, Report 2013/362, 2013. http://eprint.iacr.org/.

[35] N. Ogura, S. Uchiyama, N. Kanayama, and E. Okamoto. A note on the pairing computation using normalized miller functions. *IEICE Transactions*, 95-A(1):196–203, 2012.

[36] G. C. C. F. Pereira, M. A. Simplício, Jr., M. Naehrig, and P. S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *J. of Systems and Software*, 84(8):1319–1326, Aug. 2011.

[37] S. C. Ramanna, S. Chatterjee, and P. Sarkar. Variants of Waters' dual system primitives using asymmetric pairings - (extended abstract). In M. Fischlin, J. Buchmann, and M. Manulis, editors, *Public Key Cryptography - PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2012.

[38] A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 557–557. Springer Berlin / Heidelberg, 2005.

[39] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan*, pages 26–28, Jan. 2000.

[40] A. H. Sánchez and F. Rodríguez-Henríquez. NEON implementation of an attribute-based encryption scheme. In M. J. Jacobson Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *Applied Cryptography and Network Security - ACNS 2013*, volume 7954 of *Lecture Notes in Computer Science*, pages 322–338. Springer, 2013.

[41] M. Scott. Benchmark for MIRACL – Multiprecision Integer and Rational Arithmetic C/C++ Library - Certivox. http://certivox.org/display/EXT/Benchmarks+and+Subs. Accessed: 2013-10-15.

[42] M. Scott. Computing the Tate Pairing. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer Berlin / Heidelberg, 2005.

[43] M. Scott. Implementing cryptographic pairings. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, editors, *Pairing-Based Cryptography – Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 177–196. Springer-Verlag, 2007. Invited Talk. ftp://ftp.computing.dcu.ie/pub/resources/crypto/pairings.pdf.

[44] M. Scott. On the efficient implementation of pairing-based protocols. In L. Chen, editor, *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 296–308. Springer, 2011.

[45] M. Scott. Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Report 2013/688, 2013. http://eprint.iacr.org/.

[46] M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez, and E. Kachisa. Fast Hashing to G_2 on Pairing-Friendly Curves. In H. Shacham and B. Waters, editors, *Pairing-Based Cryptography – Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 102–113. Springer Berlin / Heidelberg, 2009.

[47] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[48] J. Solinas. ID-based digital signature algorithms, 2003. http://www.cacr.math.uwaterloo.ca/conferences/2003/ecc2003/solinas.pdf.

[49] F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, jan. 2010. http://www.cosic.esat.kuleuven.be/publications/article-1039.pdf.

[50] L. C. Washington. *Elliptic Curves: Number Theory and Criptograhpy*. Chapman & Hall/CRC, University of Maryland, College Park, USA, 2 edition, 2008.

[51] B. Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *Proceedings of the 14th international conference on Practice and theory in public key cryptography conference on Public key cryptography*, PKC'11, pages 53–70, Berlin, Heidelberg, 2011. Springer-Verlag.

[52] G. X. Yao, J. Fan, R. C. C. Cheung, and I. Verbauwhede. Faster pairing coprocessor architecture. In M. Abdalla and T. Lange, editors, *Pairing-Based Cryptography - Pairing 2012 - 5th International Conference, Cologne, Germany, May 16-18, 2012, Revised Selected Papers*, volume 7708 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2013.

APPENDIX

APPENDIX A: EFFICIENT SCALAR DECOMPOSITION FOR THE GROUPS $\mathbb{G}_1, \mathbb{G}_2$ AND G_T

A. Scalar decomposition for $\mathbb{G}_1$

The two-dimensional decomposition of the scalar n for the group $\mathbb{G}_1$ can be accomplished using the Galbraith-Scott method [22] as discussed next.

In the case of the GLV method, the task is to find a two-dimensional partition n_0, n_1, such that $n \equiv n_0 + n_1\lambda$ (mod r) with $|n_0| = |n_1| \approx |\sqrt{r}|$. In the case of BN curves, $\lambda = 36x^4 - 1$, which implies that, $(6x^2+2x) - (2x+1)\lambda \equiv 0 \bmod r$ and also, $-(2x+1) - (6x^2+4x+1)\lambda \equiv 0 \bmod r$. This can be used to find an optimal scalar partition as follows. Define the 2×2 matrix associated to λ as,

$$B = \begin{pmatrix} 6x^2 + 2x, & -2x - 1 \\ -2x - 1, & -6x^2 - 4x - 1 \end{pmatrix}$$

We stress that $B \cdot [1, \lambda]^T \equiv [0, 0]^T \bmod r$, holds for any BN curve [22].

The next step is to compute the 2-dimension vector $v = [n, 0]B^{-1}$, an operation that implies a division by r (since r divides the determinant of the matrix B). In order to guarantee that the resulting coefficients of the vector v are integers, this integer division must be followed by Babai's rounding method [4]. Notice that the costly integer division by r can be replaced with divisions by 2^m, using the following trick. Consider the vector $w' = [2^m, 0]$ with $m \geq \lceil \log_2 r \rceil$, then precompute the per-curve constant vector $v' = w'B^{-1}$ given as,

$$v' = w'B^{-1} = (v'_0, v'_1) = \left(\left\lfloor \frac{-2^m(6x^2 + 4x + 1)}{r} \right\rceil, \left\lfloor \frac{2^m(2x + 1)}{r} \right\rceil \right).$$

The vector v can then be obtained as,

$$v = \left(\left\lfloor nv'_0/2^m \right\rceil, \left\lfloor nv'_1/2^m \right\rceil \right),$$

In a final step one computes the vector $[n_0, n_1] = [n, 0] - vB$, which has the desired scalar partition form, with $|n_0|, |n_1| \approx \sqrt{r}$.

B. Scalar decomposition for $\mathbb{G}_2$ *scalar multiplication and* $\mathbb{G}_T$ *exponentiation*

Similarly to the case of $\mathbb{G}_1$ exponentiation, the task of decomposing the scalar n in these two groups is achieved by solving the *closest vector problem* in the 4-dimension lattice defined as,

$L = \{z \in \mathbb{Z}^4 : \sum_{i=0}^{3} z_i \lambda^i \equiv 0 \ (mod \ r)\}$, where λ is given as, $\lambda = t - 1 = 6x^2$. The reduced base matrix for the lattice L is then defined as, [22]

$$B = \begin{pmatrix} x+1 & x & x & -2x \\ 2x+1 & -x & -(x+1) & -x \\ 2x & 2x+1 & 2x+1 & 2x+1 \\ x-1 & 4x+2 & -(2x-1) & x-1 \end{pmatrix},$$

where $B \cdot [\lambda^0, \lambda^1, \lambda^2, \lambda^3]^T \equiv [0,0,0,0]^T \bmod r$, holds for any BN curve.

Using the same procedure described in the precedent subsection, the vector $v \approx wB^{-1}$ can be found by precomputing the per-curve constant vector $v' = w'B^{-1} = [v'_0, v'_1, v'_2, v'_3]$, with $w' = [2^m, 0, 0, 0]$, $m \geq \lceil \log_2 r \rceil$ as,

$$v' = \left(\left\lfloor \frac{2^m(2x^2+3x+1)}{r} \right\rceil, \left\lfloor \frac{2^m(12x^3+8x^2+x)}{r} \right\rceil, \left\lfloor \frac{2^m(6x^3+4x^2+x)}{r} \right\rceil, \left\lfloor \frac{2^m(-x^2-x)}{r} \right\rceil \right)$$

The vector v can then be obtained as,

$$v = \left(\left\lfloor kv'_0/2^m \right\rceil, \left\lfloor kv'_1/2^m \right\rceil, \left\lfloor kv'_2/2^m \right\rceil, \left\lfloor kv'_3/2^m \right\rceil \right),$$

In a final step one computes the vector $[n_0, n_1, n_2, n_3] = [n, 0, 0, 0] - vB$, which has the desired scalar partition form, namely, $[n_0, n_1, n_2, n_3][\lambda^0, \lambda^1, \lambda^2, \lambda^3]^T \equiv n \bmod r$, with $|n_i| \approx |r|/4$.

Appendix B: Linear secret sharing scheme

In [47], Shamir famously presented a *threshold scheme* in which given k out of n pieces of a secret D it is easy to recover D, while having less than k pieces leaves the secret D completely undetermined. Shamir pointed out that a practical realization of this scheme can be achieved via polynomial interpolation. In the same vein, an attribute-based encryption scheme, first proposed in [38], requires that an entity meets a minimal number of attributes in order to pass a given control access policy. Formally, a (monotone) access structure is defined as follows,

Definition 1 (Monotone access structure [6], [51])**.** *Let $\{P_1, P_2, \ldots, P_n\}$ be a set of attributes. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. A monotone access structure is a monotone collection $\mathbb{A}$ of non-empty subsets of attributes $\{P_1, P_2, \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets of attributes, and the sets not in $\mathbb{A}$ are called the unauthorized sets of attributes.*

In practice, the access policy can be specified via a boolean formula seen as an expression tree, where each internal node is a boolean logic gate “OR” or a logic boolean gate “AND”, with associated thresholds of 1 and 2, respectively, and where the leaves represent the attributes. In this way, any boolean formula describing a control access policy, can be transformed to a recursive arrangement of (m, s)-gates with leaves $\mathcal{H}_1, \ldots, \mathcal{H}_m, s$ denoted as, $(\mathcal{H}_1, \ldots, \mathcal{H}_m, s)$ [31]. For example the boolean formulas $\mathcal{H}_1 \wedge \mathcal{H}_2 \wedge \mathcal{H}_3$, $\mathcal{H}_1 \vee \mathcal{H}_2 \vee \mathcal{H}_3$, can be converted into $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, 3)$, $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, 1)$, respectively. Similarly, the boolean formula: $\mathcal{H}_1 \wedge (\mathcal{H}_2 \vee \mathcal{H}_3)$, which requires having $\mathcal{H}_1$, AND any of the $\mathcal{H}_2$ OR $\mathcal{H}_3$ attributes, is transformed into $(\mathcal{H}_1, (\mathcal{H}_2, \mathcal{H}_3, 1), 2)$.

A Linear Secret-Sharing Scheme (LSSS) is an efficient mechanism to decide if the minimum set of attributes required to grant access have been satisfied or not by the attribute boolean formula presented by the entity.

Definition 2 (Linear secret-sharing scheme [6], [51])**.** *A secret-sharing scheme Π over a set of attributes $\mathcal{P}$ is called linear (over $\mathbb{F}_r$) if the shares for each attribute form a vector over $\mathbb{F}_r$, and if there exists an $u \times t$ matrix M, called the share-generating matrix for Π, where u, t are the number of shares and the access policy threshold, respectively. For all $i \in \{1, \ldots, u\}$, M_i, the i^{th} row of M is labeled by an attribute $\rho(i)$, where ρ is a function from $\{1, \ldots, u\}$ to $\mathcal{P}$.*

Consider the randomly chosen vector $\nu = (s, r_2, \ldots, r_t)^T \in \mathbb{F}_r^t$, where $s \in \mathbb{F}_r$ is the secret to *be shared. Then, $\Lambda = M\nu$ is the vector of u shares of the secret s according to Π. The share $\lambda_i = M_i \cdot \nu$ belongs to the attribute $\rho(i)$.*

It was proved in [6] that each LSSS has the property of lineal reconstruction that can be defined as follows. Let $S \in \mathbb{A}$ be an authorized set, and define $I \subset \{1, 2, \ldots, u\}$ as $I = \{i : \rho(i) \in S\}$. Then, there exists a constant vector $\Omega = (\omega_1, \ldots, \omega_u) \in \mathbb{F}_r^u$ such that for any valid share vector Λ it holds that $\sum_{i \in I} \omega_i \lambda_i = s$, with

$$\begin{cases} \omega_i \neq 0 & \text{if } i \in I, \\ \omega_i = 0 & \text{otherwise.} \end{cases}$$

The constant vector Ω can then be found via standard linear algebra. It is noticed that given the LSSS security property, a constant vector $\Omega \in \mathbb{F}_r^u$ satisfying the above conditions exists if and only if Λ corresponds to an authorized set S.

Algorithm 6 converts a boolean access Formula F specified as a recursive arrangement of (m, t)-gates with leaves $\mathcal{H}_1, \ldots, \mathcal{H}_m, t$; into a $t \times u$ LSSS matrix M representing the access structure corresponding to F, where the i-th row of M is labeled by the i-th attribute of F.

Algorithm 6 Construction of the LSSS matrix [31]

```
Input: A boolean access Formula $F$ specified as a recursive arrangement of $(m,t)$-gates with leaves $\mathcal{H}_1,\ldots,\mathcal{H}_m,t$
Output: A $u \times t$ LSSS matrix $M \in \mathbb{F}_r$ representing the access structure corresponding to $F$, where the $i$-th row of $M$ is labeled by the $i$-th attribute of $F$.
$M \leftarrow (1)$, $L \leftarrow (F)$
$u \leftarrow 1$, $t \leftarrow 1$
while there exist boolean formulas in $L$ do
    Represent $L$ as $L = (L_1, L_2, \ldots, L_m)$ and $M$ as an $u \times t$ matrix.
    Find the first coordinate $L_z$ where $L_z = F_z = (F_{z_1}, \ldots, F_{z_n}, t)$ is a boolean formula
    Extract the $n$ children of $L_z$, i.e., $F_{z_1}, F_{z_2}, \ldots, F_{z_n}$ along with its threshold $t_z$.
    Insert the MSP-$(n, t_z)$ matrix on the $z$-th row of $M$, obtaining an updated $(u-1+n) \times (t-1+t_z)$ matrix.
    $L \leftarrow (L_1, L_2, \ldots, L_{z-1}, F_{z_1}, F_{z_2}, \ldots, F_{z_n}, L_{z+1}, \ldots L_m)$
    $u \leftarrow u - 1 + n$
    $t \leftarrow t - 1 + t_z$
end while
return $M$
```

Example 1. *Consider again the access structure described in §I, where the following five attributes were described,*

- *$A =$ "Patient"*
- *$B =$"Cardiologist Surgeon"*

- $C =$*“Anesthesiologist”*
- $D =$ *“Technician”*
- $E =$ *“Cardiologist Hospital”*

Using the above definitions, the access policy was described by the following boolean function,

$$(A \textbf{\textit{ OR }} B) \textbf{\textit{ OR }} ((C \textbf{\textit{ OR }} D) \textbf{\textit{ AND }} E)$$

That boolean function can be transformed to the arrangement,

$$(A, B, ((C, D, 1), E, 2), 1),$$

Using the above formulation, Algorithm 6 will generate the following sequence,

1) $M \;=\; (1) \quad (A, B, ((C, D, 1), E, 2), 1)$

2) $$M = \begin{array}{cl} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} & \begin{array}{l} A \\ B \\ ((C, D, 1), E, 2) \end{array} \end{array}$$

3) $$M = \begin{array}{cl} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} & \begin{array}{l} A \\ B \\ (C, D, 1) \\ E \end{array} \end{array}$$

4) $$M = \begin{array}{cl} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} & \begin{array}{l} A \\ B \\ C \\ D \\ E \end{array} \end{array}$$

APPENDIX C: MAIN PRIMITIVES FOR ATTRIBUTE-BASED ENCRYPTION WITH VARIABLE ATTRIBUTES DEFINED IN $\mathbb{G}_1$

Algorithm 7 Setup with attributes defined in $\mathbb{G}_1$

```
Input: $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, a set of attributes $\mathcal{H}$
Output: Public Key $PK(\mathbb{G}_1, \mathbb{G}_2, P, Q, P_\delta, \gamma)$, $\{\mathcal{H}_1 \dots \mathcal{H}_N\}$, Master private Key MK($P_\alpha$)
Choose at random : $\alpha$ and $\delta \in \mathbb{Z}_r$
$P_\delta \leftarrow [\delta]P$    {Scalar mult. in $\mathbb{G}_1$}
$P_\alpha \leftarrow [\alpha]P$    {Scalar mult. in $\mathbb{G}_1$}
$\gamma \leftarrow e_{opt}(Q, P)^\alpha$    {single pairing, exp in $\mathbb{G}_T$}
for $i \leftarrow 1$ to $\#\mathcal{H}$ do
    Generate a point $\mathcal{H}_i \in \mathbb{G}_1$ {$H_2(.)$ application}
end for
$PK \leftarrow (\mathbb{G}_1, \mathbb{G}_2, P, Q, P_\delta, \gamma), \{\mathcal{H}_1 \dots \mathcal{H}_N\}$
$MK \leftarrow (P_\alpha)$
return $PK, MK$
```

Algorithm 8 Encryption with attributes defined in $\mathbb{G}_1$

```
Input: A message $M$, $PK$, an access structure $\mathcal{S}$ given as a $u \times t$ LSSS Matrix and $I \subset \{1, 2, \dots, u\}$ as $I = \{i : \rho(i) \in \mathcal{H}\}$
Output: Ciphertext $C_T = \{\mathcal{S}, C, C_d, (C_1, D_1), \dots, (C_u, D_u)\}$
Generate a random vector $\bar{\mathbf{u}} = (s, y_2, \dots, y_t) \in \mathbb{Z}_r$.
Calculate the column vector $\bar{\lambda} = \mathcal{S}\bar{\mathbf{u}}^t$
Generate another random vector $\bar{\mathbf{x}} = (x_1, \dots, x_u) \in \mathbb{Z}_r$.
$C = M \oplus H_1(\gamma^s)$    {mult. and exp. in $\mathbb{G}_T$}
$C_d = [s]Q$    {Scalar mult. in $\mathbb{G}_2$}
for $i = 1$ to $u$ do
    $C_i \leftarrow [\lambda_i]P_\delta - [x_i]\mathcal{H}_{\rho(i)}$    {Scalar mult. point add in $\mathbb{G}_1$}
    $D_i \leftarrow [x_i]Q$    {Scalar mult. in $\mathbb{G}_2$}
end for
$C_T \leftarrow \{\mathcal{S}, C, C_d, (C_1, D_1), \dots, (C_u, D_u)\}$
return $C_T$
```

Algorithm 9 Key generation with attributes defined in $\mathbb{G}_1$

```
Input: $MK$ and a set of user's attributes $\mathcal{H}$
Output: A private key $SK = \{K, L, K_1, \dots, K_{v_\mathcal{H}}\}$
Choose at random $\tau \in \mathbb{F}_r$
$K \leftarrow P_\alpha + [\tau]P_\delta$    {Scalar mult., point add in $\mathbb{G}_1$}
$L \leftarrow [\tau]Q$    {Scalar mult. in $\mathbb{G}_2$}
for $i = 1$ to $v_\mathcal{H}$ do
    $K_i \leftarrow [\tau]\mathcal{H}_i$    {Scalar mult. in $\mathbb{G}_1$}
end for
$SK \leftarrow \{K, L, K_1, \dots, K_{v_\mathcal{H}}\}$
return $SK$
```

Algorithm 10 Decryption with attributes defined in $\mathbb{G}_1$

```
Input: $C_T$ and its LSSS matrix $S$, $SK$ and its set of attributes $\mathcal{H}$
Output: Plaintext $M$ (if the attributes in $SK$ satisfy the ciphertext's policy)
$\tilde{\mathcal{S}} \leftarrow$ Reduce the LSSS matrix $\mathcal{S}$ by removing the rows and columns unrelated with the attributes in $\mathcal{H}$
Find the determinant $\Delta \leftarrow Det(\tilde{\mathcal{S}}) \in \mathbb{F}_r$
Calculate the vector $\bar{\omega}$ as the first row of $\tilde{\mathcal{S}}^{-1}$
for $i = 1$ to $v$ do
    $C_i^{\omega_i} \leftarrow [\omega_i]C_i$    {Scalar mult. in $\mathbb{G}_1$}
    $K_{\rho(i)}^{\omega_i} \leftarrow [\omega_i]K_{\rho(i)}$    {Scalar mult. in $\mathbb{G}_1$}
end for
$M = C \oplus H_1 \left( \left( e(C_d, -[\Delta]K) \cdot e(L, \sum_{i\in\mathcal{H}} C_i^{\omega_i}) \cdot \prod_{i\in\mathcal{H}} e(D_i, K_{\rho(i)}^{\omega_i}) \right)^{\frac{1}{\Delta}} \right)$
    {Scalar mult. in $\mathbb{G}_1$, Point add in $\mathbb{G}_1$, Mult in $\mathbb{G}_T$, multi-pairing, Inversion in $\mathbb{G}_T$}
return $M$
```

Algorithm 11 Delegate with parameters in $\mathbb{G}_1$

```
Input: $SK$ and a subset $\tilde{\mathcal{H}}$ of attributes corresponding to a user
Output: A private key $\tilde{SK} = \{K', L', K'_1, \ldots, K'_{v_{\tilde{\mathcal{H}}}}\}$
Choose at random : $\tau' \in \mathbb{F}_r$
$K' \leftarrow K + [\tau']P_\delta$    {Scalar mult., point add. in $\mathbb{G}_1$}
$L' \leftarrow L + [\tau']Q$    {Scalar mult., point add in $\mathbb{G}_2$}
for $i = 1$ to $v_{\tilde{\mathcal{H}}}$ do
    $K'_i \leftarrow K_i + [\tau']\tilde{\mathcal{H}}_i$    {Scalar mult., point add. in $\mathbb{G}_1$}
end for
$\tilde{SK} \leftarrow \{K', L', K'_1, \ldots, K'_{v_{\tilde{\mathcal{H}}}}\}$
return $\tilde{SK}$
```