IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 2, pp. 192–239. DOI:10.46586/tches.v2022.i2.192-239

ABE Squared: Accurately Benchmarking Efficiency of Attribute-Based Encryption

Antonio de la Piedra[1], Marloes Venema[2] and Greg Alpár[2,3]

[1] Kudelski Security Research Team, Cheseaux-sur-Lausanne, Switzerland
[2] Radboud University, Nijmegen, the Netherlands
[3] Open University of the Netherlands, Heerlen, the Netherlands
firstname.lastname@cs.ru.nl

Abstract. Measuring efficiency is difficult. In the last decades, several works have contributed in the quest to successfully determine and compare the efficiency of pairing-based attribute-based encryption (ABE) schemes. However, many of these works are limited: they use little to no optimizations, or use underlying pairing-friendly elliptic curves that do not provide sufficient security anymore. Hence, using these works to benchmark ABE schemes does not yield accurate results. Furthermore, most ABE design papers focus on the efficiency of one important aspect. For instance, a new scheme may aim to have a fast decryption algorithm. Upon realizing this goal, the designer compares the new scheme with existing ones, demonstrating its dominance in this particular aspect. Although this approach is intuitive and might seem fair, the way in which this comparison is done might be biased. For instance, the schemes that are compared with the new scheme may be optimized with respect to another aspect, and appear in the comparison consequently inferior.
In this work, we present a framework for accurately benchmarking efficiency of ABE: ABE Squared. In particular, we focus on uncovering the multiple layers of optimization that are relevant to the implementation of ABE schemes. Moreover, we focus on making any comparison fairer by considering the influence of the potential design goals on any optimizations. On the lowest layer, we consider the available optimized arithmetic provided by state-of-the-art cryptographic libraries. On the higher layers, we consider the choice of elliptic curve, the order of the computations, and importantly, the instantiation of the scheme on the chosen curves. Additionally, we show that especially the higher-level optimizations are dependent on the goal of the designer, e.g. optimization of the decryption algorithm. To compare schemes more transparently, we develop this framework, in which ABE schemes can be justifiably optimized and compared by taking into account the possible goals of a designer. To meet these goals, we also introduce manual, heuristic type-conversion techniques where existing techniques fall short. Finally, to illustrate the effectiveness of ABE Squared, we implement several schemes and provide all relevant benchmarks. These show that the design goal influences the optimization approaches, which in turn influence the overall efficiency of the implementations. Importantly, these demonstrate that the schemes also compare differently than existing works previously suggested.

Keywords: attribute-based encryption · implementation · benchmarking · ciphertext-policy attribute-based encryption

1 Introduction

Since attribute-based encryption (ABE) was introduced in 2005 by Sahai and Waters [SW05], much progress has been made in the development of pairing-based ABE schemes. As is common in the field of cryptography, whenever a new scheme is presented, its efficiency

Received: 2021-10-15 Accepted: 2022-12-15 Published: 2022-02-15

is compared to that of other state-of-the-art schemes. For ABE, the Charm framework [AGM+13] is used in many cases [RW13, RW15, AC17a, ABGW17], which simplifies the prototyping of new pairing-based schemes and provides benchmarking tools. However, because Charm mainly aims at usability in this endeavor, it uses several abstraction layers between the schemes and the necessary arithmetic. As a result, not all available optimizations can be used in benchmarking efforts, even though these might be significant in any comparisons. Furthermore, by default, the Charm framework builds on the PBC library [Lyn13], which only supports outdated elliptic curves that have been proven not to provide 128 bits of security [KB16, BD19]. Consequently, many implementations and efficiency comparisons use these outdated curves. By extension, those implementations do not provide realistic estimates of computational costs in practice. When implemented for practice, curves that currently provide 128 bits of security should be used. Because these might provide different trade-offs in efficiency, the implementations may incur different computational costs than the curves used in the old benchmarks [Ara17, CDS20].

Oftentimes, works that do not use Charm in their efficiency analyses have similar issues. For instance, they may not use all, if any, optimized arithmetic or other lower-level optimization techniques [Zeu20, AHM+16, TKN20, PRMV21]. Such techniques allow for faster computations of exponentiations, such as fixed-base exponentiations or multiple-base exponentiations [Sco11, Möl01]. These are often used when elliptic-curve schemes are deployed in practice, and provide a significant computational advantage over regular variable-base exponentiations. Other implementations may be targeted for specific platforms such as certain embedded devices [SR13, WZSI14, MTP+21]. Hence, they are difficult to use in future efficiency comparisons without implementing the schemes for those specific devices. On the other hand, software implementations that do optimize the arithmetic used in the schemes [ZPM+15] have implemented all underlying arithmetic for some specific elliptic curve, and are therefore difficult to adapt to other, more up-to-date, curves. This is problematic, since this particular curve, e.g., the BN254 curve [BN05], may turn out to provide e.g., only 100 bits of security [SKSW20].

Another common denominator of these implementations is the absence of a clearly-defined design rationale when the schemes are instantiated in pairing-friendly elliptic-curve groups. For instance, choosing a suitable pairing-friendly group providing 128 bits of security [Gui20b] is important for the overall efficiency [Sco11, CDS20]. However, not every curve may be a good choice for every scheme. Moreover, at the protocol level, schemes are often designed in the symmetric, type-I setting [Sco11]. That means that the used pairing operation $\hat{e}\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$—which maps two source groups $\mathbb{G}$ and $\mathbb{H}$ to a target group $\mathbb{G}_T$—is assumed to be symmetric, i.e., the two source groups are isomorphic, and thus, $\mathbb{G} = \mathbb{H}$. On the other hand, in practice, it is better to use asymmetric, type-III pairings due to their efficiency [GPS08] and security [Gal14], such that no efficiently computable isomorphism exists between the two source groups, i.e., $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ with $\mathbb{G} \neq \mathbb{H}$. While such schemes can be converted from the type-I to the type-III setting [RCS12], existing works that facilitate this [AGH13, AGOT14, AGH15, AHO16] are often not used in the implementation of ABE schemes.

Nevertheless, such a design rationale determines the groups in which the computations are performed during key generation, encryption and decryption, and is therefore crucial when analyzing the efficiency of the scheme. This rationale heavily influences the choice of pairing-friendly groups and the type conversion, and by extension, the efficiency of the scheme. For instance, operations in $\mathbb{G}$ are generally more efficient than those in $\mathbb{H}$ [Ara17, AGM+, CDS20]. Consequently, if a designer places all ciphertext components in $\mathbb{G}$, then the encryption efficiency is optimized at the expense of the key generation efficiency. Another designer might want to optimize the key generation efficiency, and therefore places all key components in $\mathbb{G}$, and the ciphertext components in $\mathbb{H}$. Because not all implementations take into account and specify these considerations, they cannot be

effectively and meaningfully compared [VAH21]. In fact, a somewhat unethical cryptographer who, for instance, wants to promote their new scheme's fast encryption algorithm could place all ciphertext components of their own scheme in $\mathbb{G}$, while they place the compared scheme's ciphertext components in $\mathbb{H}$. As a result, their own scheme might outperform the other scheme, even though the other scheme would have outperformed the new scheme if its ciphertext components had also been placed in $\mathbb{G}$. In summary, for various efficiency goals, a different distribution of the key and ciphertext components over the two source groups may be optimal.

In this work, we aim to resolve the aforementioned issues. In particular, we provide a framework for benchmarking and comparing efficiency of ABE schemes that takes into account important features such as optimized arithmetic and conversion techniques. Along the way, we introduce novel conversion techniques to obtain e.g., a type conversion with an optimized decryption algorithm. We also show how this framework can be applied to existing schemes by implementing and benchmarking them. Lastly, we illustrate how these benchmarks can be compared fairly, by comparing the variants that are optimized in the same way, e.g., the variants with an optimized decryption algorithm.

1.1 Our contribution

We set up ABE Squared, a general framework for accurately benchmarking efficiency of ABE. This framework describes optimization approaches in the implementation of ABE schemes based on various design goals, by unifying multiple established areas in optimization. By choosing one design goal, multiple schemes can be optimized in a uniform way, and thus be fairly compared. Concretely,

- we identify four *optimization layers* that are important in the implementation of the schemes:
  - the used arithmetic and group operations;
  - the choice of pairing-friendly curve;
  - the order of computations;
  - the type-conversion techniques;
- we formulate various *optimization approaches* for several clearly defined design goals:
  - optimized key generation;
  - optimized encryption;
  - optimized decryption;
  - balanced (in any combination of the algorithms, e.g., balanced key generation/encryption, balanced encryption/decryption);
- as part of the optimization approaches, we introduce new heuristic, manual *conversion techniques* from the type-I to the type-III setting, which takes into account the other optimization layers. This is especially important for optimizing the decryption algorithm, for which the existing frameworks fall short.

To illustrate the effectiveness of our framework, we provide the *implementations* of several important ciphertext-policy attribute-based encryption (CP-ABE) [BSW07] schemes:

- Wat11-I [Wat11];
- Wat11-IV [Wat08]—which is Wat11-I using a full-domain hash;
- RW13 [RW13]—the "unbounded" version of Wat11-I;
- AC17 [AC17b]—the more efficient version of Wat11-I (which can also be employed with a full-domain hash).

1.2 Background

We provide further background information and motivate our choices and some of the features of the new framework.

Pairing-based ABE. Currently, pairing-based ABE is the most established type of ABE. In addition, ABE schemes based on the hardness of the learning with errors problem [Boy13]—which is believed to be hard in the post-quantum setting—may also be efficient enough for practice [DDP+18]. However, the most expressive and secure lattice-based scheme was only recently introduced [ABN+21, DKW21], and it is considerably less expressive and less secure than the most expressive and most secure pairing-based schemes [LL20]. Because of its maturity, we focus only on pairing-based ABE in this work.

Ciphertext-policy ABE. In ABE, the key pairs are associated with attributes. Specifically, in ciphertext-policy ABE (CP-ABE) [BSW07], the secret keys are associated with a set of attributes. Subsequently, the messages are encrypted under some access policy defined over the attributes. The resulting ciphertext can be decrypted by some key, if the attribute set associated with some key satisfies this policy. Conversely, in key-policy ABE (KP-ABE) [GPSW06a], the keys are associated with policies and the ciphertexts with attribute sets. Since CP-ABE can implement a fine-grained (attribute-based) data access control [HFK+19] mechanism on a cryptographic level [ETS18, MJ18] and is thus of much practical interest, we focus on and implement several CP-ABE schemes. Because the European Telecommunications Standards Institute has put efforts in standardizing CP-ABE for e.g., cloud and mobile services, we focus on the implementation of ABE on high-performance devices such as smartphones and computers.

RELIC. Our framework uses the RELIC toolkit [AGM+], which is a cryptographic library that can be used for building elliptic-curve and pairing-based cryptographic schemes. In particular, it implements pairing-friendly elliptic curves that provide at least 128 bits of security, such as BLS12-381 and BLS12-446 [BLS02, Bow, BD19, WB19]. It provides high-speed implementations of frequently used arithmetic and group operations. In general, using a library for the lowest level of optimizations, i.e., arithmetic and groups operations, allows us to easily instantiate the schemes with new—possibly more secure or more efficient—curves, in the case that it is necessary to do so. We chose RELIC because it is actively maintained, and compared to other libraries such as MIRACL [Sco03], it supports more pairing-friendly curves providing 128 bits of security. This allows us to compare the efficiency of the implemented schemes on multiple curves with the same security level. In this way, we can investigate which trade-offs the various curves incur.

BLS12-381. A specific elliptic curve of interest supported by RELIC is the BLS12-381 curve [Bow, WB19], which is an instantiation of the curves designed by Barreto, Lynn and Scott (BLS) [BLS02]. In the last few years, BLS12-381 has established itself as a popular curve. For example, it is used in the Algorand blockchain [GHM+17, BKLS02] and by the privacy-oriented Zcash blockchain in the implementation of zk-SNARKS [BCCT12, ZCa21].

Other pairing-friendly curves. We show that the chosen pairing-friendly curve influences the overall efficiency of the scheme, and is thus an important aspect in the optimization. To illustrate this, we benchmark the schemes on various curves, at the same security levels (see Section 3.2 for a selection of the curves). As part of the optimization approaches, we pick the best curve, given the chosen design goal.

OpenABE. In addition to Charm, we also compare our implementations with the OpenABE [Zeu20] implementation of the scheme by Waters [Wat11, Wat08]. It is a publicly available software implementation of ABE that is actively maintained, and that is specifically designed for practical application rather than for benchmarking efforts. Furthermore, OpenABE also relies on RELIC for the curve arithmetic. However, it is specifically configured to only support the BN254 curve.

The implemented schemes. We implement several variants of three schemes: Wat11 [Wat11, Wat08], RW13 [RW13] and AC17 [AC17b]. Specifically, we consider two versions of Wat11 and AC17: their small-universe and large-universe variants. In small-universe schemes, the number of attributes that can be used in the system is bounded, while in large-universe schemes, this number is unbounded. We have chosen these schemes, in part, due to their popularity in follow-up work (see Section 4.1). By benchmarking and comparing these constructions, we obtain a better understanding of their efficiency. Compared to Charm, their efficiency also compares differently, which illustrates that our framework truly leads to significant improvements in the accuracy of the benchmarks.

Our type-conversion techniques. As part of our framework, we introduce novel heuristic and manual techniques to convert the schemes from the type-I to the type-III setting. Although type conversion has been extensively studied, culminating in various works that even automate this effort [AGH13, AGOT14, AGH15, AHO16], these do not sufficiently optimize the schemes for all of the design goals that we consider in our optimization approaches. In particular, some of these conversion techniques focus, in terms of efficiency, mainly on the sizes of the parameters [AGOT14, AGH15, AHO16]. These are not sufficient in optimizing the type-converted scheme for e.g., the optimized decryption approach. In contrast, [AGH13] also allows for the optimization of the computational costs, but only takes into account the number of group operations, e.g., exponentiations, and not the computational costs of the various group operations. This does not allow us to distinguish between the different group operations within the same group, even though they may incur different computational costs. Furthermore, [AGH13] does not explicitly allow the choice of pairing-friendly curve or the order of the computations to be taken into account. Hence, we develop techniques to consider these different computational costs, by also measuring these costs for various suitable curves, and by determining the most efficient order of computations. In this way, we can minimize the computational costs of the algorithm(s) more accurately, based on the chosen design goal and associated optimization approach.

Code. Our implementations of the schemes are available in the public domain, and can be found at `https://github.com/abecryptools/abe_squared`.

Positioning our framework. Our framework aims to bridge a gap in the benchmarking of ABE schemes, as described above. Notable software implementations that are built on libraries such as RELIC and that provide benchmarking utilities—and that are still maintained—are Charm and OpenABE. However, their goals are arguably different from ours (see Figure 1). Charm and OpenABE are both focused on usability, albeit in different ways. Charm aims to be usable in the prototyping and benchmarking of schemes, so that cryptographers can implement new schemes without having to know implementation details. OpenABE aims to be usable for practical applications, providing ready-to-use ABE implementations for practitioners. As a result, neither of their implementations uses all available optimized arithmetic. In contrast, our framework, ABE Squared, focuses on optimization rather than usability, such that a more accurate view of the efficiency of ABE schemes can be obtained. Although the implementations can be used by any cryptographer to benchmark and compare the schemes, they are not immediately suitable for practical

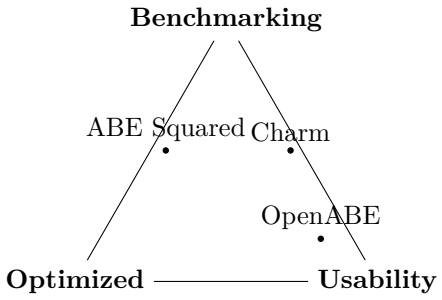


Figure 1: Rough overview of the position of our framework compared to Charm and OpenABE, with respect to the goals: benchmarking, optimization and usability.

applications like OpenABE. Furthermore, our implementations do not aim to provide a platform to readily implement new schemes, like Charm does, since it requires a significant engineering expertise and some familiarity with RELIC.

2 Preliminaries

2.1 Notation

We use the following notations. If an element is chosen uniformly at random from some finite set S, then we denote this as $x \in_R S$. We denote $[a, b] = \{a, a+1, ..., b-1, b\}$, and $[b] = [1, b]$. We use boldfaced variables $\mathbf{A}$ and $\mathbf{v}$ for matrices and vectors, respectively.

2.2 Access structures

In CP-ABE, the ciphertexts are associated with access policies, e.g., Boolean formulas consisting of the operators "AND" and "OR". To ensure that only authorized users can decrypt, the policies are converted into some suitable access structures.

2.2.1 Linear secret sharing schemes

For the definitions of the schemes, we represent policies $\mathbb{A}$ by linear secret sharing scheme (LSSS) matrices [GPSW06b], i.e., $\mathbb{A} = (\mathbf{A}, \rho)$ is such that $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ is a matrix, and ρ maps its rows to attributes. Then, for some random vector $\mathbf{v} = (s, v_2, ..., v_{n_2})$, the i-th share of secret s generated by this matrix is $\lambda_i = \mathbf{A}_i \mathbf{v}^\intercal$, where $\mathbf{A}_i$ denotes the i-th row of $\mathbf{A}$. Let $\mathcal{S}$ be an attribute set, and define $\Upsilon = \{i \in [n_1] \mid \rho(i) \in \mathcal{S}\}$. If $\mathcal{S}$ satisfies $\mathbb{A}$, then there exist $\varepsilon_i \in \mathbb{Z}_p$ for all $i \in \Upsilon$ such that $\sum_{i \in \Upsilon} \varepsilon_i \mathbf{A}_i = (1, 0, ..., 0)$, and thus $\sum_{i \in \Upsilon} \varepsilon_i \lambda_i = s$.

2.2.2 Implementing access structures

In our implementations, we represent the access structures as access trees [GPSW06a]. An access policy represented as a string is converted into a tree. The leaves correspond to the attributes and the nodes to OR, AND or (t, n)-threshold gates. In the full version [dlPVA22], we recap the algorithms to convert policies to access trees and LSSS matrices, and an algorithm to convert policies to more efficient LSSS matrices [LW10], which, as we show in Section 4.6.3, are more efficient than the access trees used by OpenABE.

2.3 Ciphertext-policy ABE

Definition 1 (Ciphertext-policy ABE [BSW07])**.** A ciphertext-policy attribute-based encryption (CP-ABE) scheme with some universe of attributes $\mathcal{U}$ consists of four algorithms:

- $\text{Setup}(\lambda) \to (\text{MPK}, \text{MSK})$: The setup takes as input a security parameter λ, it outputs the master public-secret key pair (MPK, MSK).
- $\text{KeyGen}(\mathcal{S}, \text{MSK}) \to \text{SK}_{\mathcal{S}}$: The key generation takes as input a set of attributes $\mathcal{S}$ and the master secret key MSK, and outputs a secret key $\text{SK}_{\mathcal{S}}$.
- $\text{Encrypt}(M, \mathbb{A}, \text{MPK}) \to \text{CT}_{\mathbb{A}}$: The encryption takes as input a message M, an access policy $\mathbb{A}$ and the master public keys MPK. It outputs a ciphertext $\text{CT}_{\mathbb{A}}$.
- $\text{Decrypt}(\text{CT}_{\mathbb{A}}, \text{SK}_{\mathcal{S}}) \to M'$: The decryption takes as input the ciphertext $\text{CT}_{\mathbb{A}}$ with an access policy $\mathbb{A}$, and a secret key $\text{SK}_{\mathcal{S}}$ for a set of attributes $\mathcal{S}$. It succeeds and outputs the message M' if $\mathcal{S}$ satisfies $\mathbb{A}$. Otherwise, it aborts.

A scheme is called correct if $M = M'$.

Large-universe ABE. The universe of attributes $\mathcal{U}$ can be small or large [SW05]. If in the Setup, a public key is generated for each attribute, the universe is small. Conversely, if the size of the master public key does not depend on the size of the universe, the universe is large. In some large-universe schemes, the public key associated with some attribute is generated with e.g., a full-domain hash (FDH) [PTMW10, GPSW06a].

Multi-use ABE. The policies used during encryption may be restricted in the number of times that one attribute may occur. If an attribute may only occur once, we call the scheme one use. If it allows unlimited occurrences of one attribute, we call it multi use.

2.4 Pairings (or bilinear maps)

We define a pairing to be an efficiently computable map e on three groups $\mathbb{G}, \mathbb{H}$ and $\mathbb{G}_T$ of prime order p, i.e., $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$, with generators $g \in \mathbb{G}, h \in \mathbb{H}$ such that for all $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, h^b) = e(g, h)^{ab}$ (bilinearity), and for $g^a \neq 1_{\mathbb{G}}, h^b \neq 1_{\mathbb{H}}$, it holds that $e(g^a, h^b) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}'}$ denotes the identity of the associated group $\mathbb{G}'$ (non-degeneracy). We refer to $\mathbb{G}$ and $\mathbb{H}$ as the two source groups, and $\mathbb{G}_T$ as the target group. If an isomorphism exists between $\mathbb{G}$ and $\mathbb{H}$, i.e., $\mathbb{G} = \mathbb{H}$, we call the pairing symmetric or of type I. If $\mathbb{G} \neq \mathbb{H}$, we call the pairing asymmetric. Specifically, if an efficiently computable homomorphism exists from $\mathbb{H}$ to $\mathbb{G}$ (but not from $\mathbb{G}$ to $\mathbb{H}$), we call the pairing of type II, and if there exists no such efficiently computable homomorphism, we call the pairing of type III. With respect to the efficiency and security, type-III pairing groups are preferred [GPS08, Gal14]. However, most ABE schemes are designed in the type-I setting; hence, they need to be converted to the type-III setting [AGOT14, AGH15, AHO16]. We use $\hat{e}$ for pairings in general, and e for type-III pairings.

2.5 Pairing-based ABE

Most ABE schemes follow the same structure, and can be captured in the pair encodings framework [Att14, Att16, AC17b], which considers only "what happens in the exponent" of the keys and ciphertexts, and clearly indicates in which group each component resides. Schemes that fit in this framework have a master public key, secret keys and ciphertexts that exist almost entirely in the two source groups. The only exception is one target group element in the master public key and ciphertexts, which is used to mask the message, e.g., $M \cdot e(g, h)^{\alpha s}$. To decrypt, $e(g, h)^{\alpha s}$ must be recovered by pairing and possibly

exponentiating the appropriate ciphertext and secret key components. Note that, because the secret keys and ciphertexts consist mostly of components in the two source groups, no pairing operations are required during key generation and encryption.

We also use a shorter notation derived from this framework. For example, the master public key component $B_{\text{att}} = g^{b_{\text{att}}}$ is denoted as $[b_{\text{att}}]_{\mathbb{G}}$. Similarly, the secret key component $K = h^{\alpha - rb}$ is denoted as $[\alpha - rb]_{\mathbb{H}}$ and the ciphertext component $C = M \cdot e(g,h)^{\alpha s}$ is denoted as $[m + \alpha s]_{\mathbb{G}_T}$, where we assume that $M = e(g,h)^m$ for some $m \in \mathbb{Z}_p$. We symbolize the pairing operation as expected, e.g., $e([s]_{\mathbb{G}}, [r]_{\mathbb{H}}) = e(g,h)^{sr}$, and indicate an exponentiation with integer r as $[b]^r_{\mathbb{G}}$, which then evaluates to $[rb]_{\mathbb{G}}$. As a rule, we use variables involving the letter b for the master public key, variables involving the letter r for the secret keys and variables involving the letter s for the ciphertext.

3 Our framework: ABE Squared

We introduce our framework, ABE Squared, in this section. Concretely, we identify several layers of optimization: the arithmetic and group operations, the pairing-friendly groups that are used, the order of the computations and the type conversion (see Figure 2). The goal of our framework is to optimize the theoretical description of the scheme. As such, we want to obtain a description of the scheme that directly yields the most efficient implementation. In this process, the design goal associated with a practical application is crucial: some applications may require an optimized encryption while others require an optimized decryption algorithm. In order to achieve this goal, these four layers need to be optimized. We do this by devising optimization approaches based on these design goals.

These optimization approaches consist of several steps. In particular, we first analyze the efficiency of the arithmetic and group operations used in the schemes by benchmarking their efficiency in the pairing-friendly groups that can be used. Subsequently, we show how the order of computations can be optimized, given the efficiency of the available algorithms for arithmetic in the chosen pairing-friendly groups. (Note, however, that the optimal order may depend on the choice of pairing-friendly groups and the distribution of the key and ciphertext components, i.e., in which groups these live. Because type conversion—which determines this distribution—is the next step, we may need to adjust the order at a later stage in the optimization approach.) Finally, we show how the schemes can be instantiated in these groups to obtain the best possible efficiency, given the design goal. To this end, we devise new manual and heuristic techniques to convert the scheme from the type-I to the type-III setting. Possibly, the choices that are made during this type conversion might require that we circle back to the choice of pairing-friendly groups or the order of computations. For instance, it may not be clear what the best choice of pairing-friendly group is without simply benchmarking the schemes for all of them.

3.1 Optimized arithmetic and group operations

We analyze the efficiency of the arithmetic that may be required to perform the algorithms of a scheme. Many efficient algorithms exist to perform arithmetic in groups $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$, including optimized algorithms for certain combinations of arithmetic. Furthermore, depending on the fixed use of certain variables, the use of precomputation may significantly speed up the computations.

- **Variable-base exponentiation (VBE):** an exponentiation of the form g^x, in which the variable base g varies in each execution of the algorithm;
- **Fixed-base exponentiation (FBE):** an exponentiation of the form g^x, in which the base g is fixed after the setup and is the same in each execution of the algorithm;

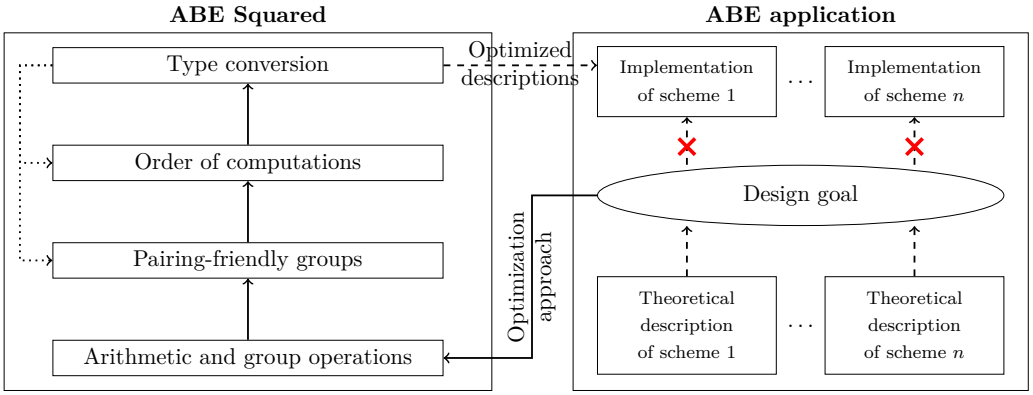


Figure 2: Overview of the ABE Squared framework and its relationship with ABE applications. In particular, the diagram describes the steps needed between the theoretical descriptions and the implementations of (possibly multiple) ABE schemes. Instead of moving from a design goal to the implementation of a scheme directly, we first optimize the theoretical description of the scheme for the chosen design goal.

- **Multiple-base exponentiation (MBE):** a product of multiple exponentiations [Möl01], typically of the form $\prod_{i \in \mathcal{I}} g_i^{x_i}$ such that g_i are bases and x_i are exponents for each $i \in \mathcal{I}$ with $|\mathcal{I}| \geq 2$. Note that RELIC refers to these as simultaneous exponentiations[1] instead, and has two functions for this algorithm: `_mul_sim`, a two-base variant and `_mul_sim_lot`, a multi-base variant;

- **Multi-pairing:** a product of pairing operations can be executed more efficiently [GS06]. In general, a pairing computation consists of a Miller loop [Mil04] and a final exponentiation. In a pairing product, the final exponentiation can be shared, i.e., it only needs to be performed once. In this way, only the Miller loop needs to be executed for each additional pairing operation in the product;

- **Fixed-argument pairing:** a pairing operation can be computed more efficiently if the first argument is fixed. For instance, [CS10] speeds up the Miller loop by 37%. RELIC does not support fixed-argument pairings, however;

- **Hashing into the group:** a mapping from the set of arbitrary-length strings $\{0,1\}^*$ to a group. RELIC supports these, including a more optimized variant for the BLS12-381 curve [WB19].

3.2 Optimal choices of pairing-friendly groups

Another aspect that influences the efficiency of the algorithms is the choice of the pairing-friendly group [GPS08, Ara17]. In general, many pairing-friendly groups exist that provide

[1] Actually, RELIC refers to multiple-base exponentiations as simultaneous multiplications, where 'multiplication' refers to a scalar multiplication. A scalar multiplication is an additive operation in an elliptic-curve group analog to an exponentiation in a multiplicative group.

Table 1: The computational costs of various algorithms on the elliptic curves used in our comparison, expressed in the number of 10^3 clock cycles. For each pair of curves with the same security level, the lowest costs are typeset in **bold**. These benchmarks were run on an AMD Ryzen 7 PRO 4750 processor, with power management disabled and throttle to max. frequency (one single core) at 4.1 GHz. Note that TBE and MBE denote a two-base exponentation (run with `_mul_sim`) and multi-base exponentiation with two bases (run with `_mul_sim_lot`), respectively, and MP a multi-pairing with two pairings.

	Costs of the algorithms in $\mathbb{G}$					Costs of the algorithms in $\mathbb{H}$					In $\mathbb{G}_T$	Pairing costs	
Curve	**VBE**	**FBE**	**TBE**	**MBE**	**Hash**	**VBE**	**FBE**	**TBE**	**MBE**	**Hash**	**VBE**	**Pair**	**MP**
BN254	91	51	135	160	58	157	112	339	273	167	236	425	585
BLS12-381	**184**	**102**	**266**	**319**	225	**324**	**247**	**729**	**548**	548	**496**	**1245**	**1618**
BN382	266	153	382	473	**151**	487	372	1105	837	**480**	745	1405	1963
BLS12-446	**275**	**158**	**433**	718	339	**491**	**376**	**1108**	1278	828	**734**	**1836**	**2399**
BN446	400	231	570	**481**	**210**	747	577	1714	**822**	**666**	1215	2207	3081

128 bits of security [Gui20a], currently the recommended minimum security level for cryptography [Bar20]. These groups typically consist of elliptic-curve groups, such as the BLS [BLS02] and BN [BN05] curves. Some of the curves listed in [Gui20a] provide more than 128 bits of security, and therefore, they will still likely yield sufficient security if the most novel attacks are slightly improved [KB16, BD19, Gui20a]. In contrast, other curves provide slightly fewer than 128 bits of security and may not provide sufficient security if these attacks are improved. RELIC [AGM+] supports three curves with security levels in the $[129, 135]$-range, i.e., BN446, BLS12-446 and BLS12-455, and two curves in the $[125, 128]$-range, i.e., BLS12-381 and BN382. On the one hand, curves with a higher security level provide less efficient arithmetic [GPS08]. On the other hand, these curves provide more than 128 bits of security. This might also be beneficial, because most ABE schemes decrease a few bits in security as some of the parameters, e.g., the size of the access policies, grow [Wat11, RW13, AC17b]. If curves with a security level in the $[125, 128]$-range are used, the implementations of these schemes provide even fewer than 128 bits of security. For instance, BLS12-381 currently provides roughly 126 bits of security [GMT20], and the schemes that we have selected lose an additional 4 bits for the maximum policy sizes that we will use (and may even lose an additional 7 bits, see Section 4.1.4). Therefore, the implementations provide 122 bits of security. In contrast, BLS12-446 and BN446 provide 132 bits of security [GS19], and thus, the implementations provide 128 bits of security.

3.3 Benchmarks of the group operations on various curves

To choose a suitable curve, it is important to know how efficiently the group operations perform. Table 1 lists the performances of various algorithms on the elliptic curves that we will use in our benchmarks in Section 4. The table shows that, at the same security level, BLS12 curves outperform the BN curves in almost all algorithms except hashing and multiple-base exponentiations with large numbers of bases. Therefore, we expect that for most, if not all, schemes, the BLS12 curves are better choices than the BN curves. The table also shows that the arithmetic in $\mathbb{G}$ is generally faster than the arithmetic in $\mathbb{H}$, which in turn is faster than the arithmetic in $\mathbb{G}_T$. Furthermore, performing an additional pairing operation—whose costs are slightly lower than the costs incurred by a Miller loop—is more costly than exponentiating in $\mathbb{G}$ and $\mathbb{H}$, while it is less costly than exponentiating in $\mathbb{G}_T$.

3.4 Optimizing the order of computations

The order of the computations can also be optimized. The most notable example of an optimized order of computations is to share a pairing operation when several components share an argument on the other side of the pairing [PTMW10]. From this point forward, we will refer to this kind of product as a shared-argument pairing product. For

instance, rather than computing $\prod_{i\in\Upsilon}\hat{e}(K,C_i)$, one can compute $\hat{e}(K,\prod_{i\in\Upsilon}C_i)$, which only requires one pairing operation and $|\Upsilon|$ multiplications in one of the source groups instead of one $|\Upsilon|$-multi-pairing. Similar optimizations can be done by allowing the key generation authority to generate components such as $h^{r_{\text{att}}(b_1\text{att}+b_0)+rb'}$ by first computing $r_{\text{att}}(b_1\text{att}+b_0)+rb'$ in $\mathbb{Z}_p$ and then exponentiating h with the result, rather than computing this as $h^{r_{\text{att}}(b_1\text{att}+b_0)+rb'}=h^{r_{\text{att}}b_1\text{att}}h^{r_{\text{att}}b_0}h^{rb'}$. While the former only costs one exponentiation and three multiplications in $\mathbb{Z}_p$, the latter requires a three-base exponentiation, which is generally much less efficient. In optimizing the order of computations, it is important to know the efficiency of the operations in the various groups. For instance, $\prod_{i\in\Upsilon}(\hat{e}(K_{1,i},C_{1,i})\cdot\hat{e}(K_{2,i},C_{2,i}))^{\varepsilon_i}$ is often optimized to $\prod_{i\in\Upsilon}\hat{e}(K_{1,i}^{\varepsilon_i},C_{1,i})\cdot\hat{e}(K_{2,i}^{\varepsilon_i},C_{2,i})$, because two exponentiations in $\mathbb{G}$ are more efficient than one exponentiation in $\mathbb{G}_T$. While this is the case for the curves considered in this work, it might be the case that, for some curves, it is more efficient to do one exponentiation in $\mathbb{G}_T$ instead of two in $\mathbb{G}$. Furthermore, we show in Section 3.6 that the optimal order may depend on the distribution of the key and ciphertext components over the two source groups (e.g., Remark 3).

3.5 Our optimization approaches for specific design goals

In optimizing the ABE schemes, we consider various approaches based on specific real-world design goals. In particular, this influences the conversion of the schemes to the type-III setting, but possibly also the choice of an elliptic curve. For pairing-based schemes in general, such conversions from the type-I to the type-III setting were previously considered in [AGH13, AGOT14, AGH15, AHO16], which all automate this effort and which focus mostly on other predicate encryption primitives such as identity-based encryption [Sha84]. However, these frameworks only optimize the parameter sizes, and not necessarily the computational costs of the algorithms. While e.g., optimizing the ciphertext size also results in an optimized efficiency of the encryption algorithm, such approaches might not necessarily lead to an optimized decryption algorithm. Furthermore, depending on the application in which ABE is going to be deployed, a practitioner may prefer a more balanced approach, in which the total costs of e.g., the encryption and decryption algorithms are optimized rather than either one of them. To this end, we define the following optimization approaches based on design goals.

- **Optimized key generation (OK):** optimize the efficiency of the key generation algorithm;
- **Optimized encryption (OE):** optimize the efficiency of the encryption algorithm;
- **Optimized decryption (OD):** optimize the efficiency of the decryption algorithm;
- **Balanced key generation/encryption (BKE):** optimize the average costs of the key generation and encryption algorithms;
- **Balanced encryption/decryption (BED):** optimize the average costs of the encryption and decryption algorithms.

A practitioner can also devise optimization approaches for other design goals, e.g., "balanced key generation/decryption". In general, a practitioner can specify any goal in which the average/total costs of any subset of algorithms is minimized. Even though these may be useful in practice, the conversion techniques used in these approaches are likely similar to those used in the aforementioned approaches.

The importance of computational-efficiency focused approaches. In contrast to most conversion frameworks [AGOT14, AGH15, AHO16], we do not necessarily describe our approaches in terms of the sizes of the public keys, secret keys or ciphertexts, but rather in terms of the computational costs of the algorithms like [AGH13]. However, due to the fact that the smallest group $\mathbb{G}$ also provides the most efficient arithmetic, we estimate that our optimized encryption and key generation approaches coincide with the optimized ciphertext and secret key size approaches in [AGH15]. The other three design goals, on the other hand, do not seem to match with any of the approaches in these conversion frameworks [AGOT14, AGH15, AHO16], even though these may be of interest to practitioners. For instance, optimizing either the key generation or encryption algorithm may result in a heavy performance penalty on the other algorithms, while a balanced approach would ensure that an algorithm can perform efficiently while only requiring minimal sacrifice in efficiency on the other algorithms. Furthermore, we show that an optimal key or ciphertext size does not necessarily imply an optimal decryption cost, but requires a more intricate, in-depth analysis of the decryption algorithm and the arithmetic provided by the chosen groups and pairing. At a high level, our approach is thus closer to [AGH13], which focuses on optimizing the computational efficiency of the scheme, albeit in an automated way. Another common denominator between [AGH13] and our work is that the converted scheme is not automatically secure, though we argue that the converted schemes are secure nonetheless. An advantage of our type-conversion techniques over [AGH13] is that we take into account the costs of the arithmetic and group operations in our optimizations.

3.6 Our type-conversion methods

We describe our type-conversion methods, which can be used to convert a scheme from the type-I to the type-III setting given some specific design goal (as discussed in Section 3.5). We assume that the scheme to be converted is given in the type-I setting, and that it can be somewhat freely converted to the type-III setting without breaking its security. This is often the case: schemes are predominantly designed in the type-I setting [Sco11], but the symmetry of the pairing in many cases is not needed for their security [AGH13]. The symmetry is, on the other hand, to some extent important for the correctness. Specifically, the key and ciphertext components that are paired during decryption need to live in different source groups. If these two paired components live in the same group, then this yields incorrectness of decryption, for the simple reason that they cannot be paired. In addition, when full-domain hashes are used, we are slightly more limited, since the components involving these need to be placed in the same source groups (see Remark 1). In sum, while we have much freedom in how we convert from the type-I to the type-III setting, we are bounded by the correctness of the scheme.

Furthermore, as we mentioned, conversion from the type-I to the type-III setting is not trivial, as any conversion heavily influences the efficiency of a scheme. Hence, we ideally want to apply this conversion in the optimal way considering the optimization approach (associated with the chosen design goal) and the correctness of the decryption algorithm. For instance, if we want to convert some scheme to the type-III setting such that it has the most efficient encryption algorithm (i.e., as in the OE approach), then we attempt to place as many ciphertext components in the first group $\mathbb{G}$ as possible. This consequently means that the key components that are paired with these ciphertext components need to be placed in the second group $\mathbb{H}$. For the other approaches, the conversion is often more intricate, and requires knowledge of the computational costs in the groups $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$.

For each optimization approach, we follow the same steps:

(1) We first list the secret key and ciphertext components, and order them in such a way, that it is clear which components are paired during decryption such that we can maintain correctness of decryption;

(2) We specify for each key-ciphertext component pair whether they need to be exponentiated and whether they occur in a product during decryption;

(3) We determine the computational costs, for each key and ciphertext component, of the key generation and encryption algorithm;

(4) To determine the computational costs of the decryption algorithm, the order of the computations needs to be optimized (Section 3.4), which depends on the curve, and possibly the distribution of the components;

(5) Based on this information, we can determine the best possible distribution of the key and ciphertext components over the two source groups for a specific optimization approach. We describe how this can be done below.

Remark 1 (Full-domain hashes). A full-domain hash (FDH) is a mapping $\mathcal{H}_1 \colon \{0,1\}^* \to \mathbb{G}$ that maps arbitrarily-long bit strings into the group. Because no hashes $\mathcal{H}_1 \colon \{0,1\}^* \to \mathbb{G}$ and $\mathcal{H}_2 \colon \{0,1\}^* \to \mathbb{H}$ exist such that $e(\mathcal{H}_1(\text{att}), h) = e(g, \mathcal{H}_2(\text{att}))$ holds [GPS08], we need to place the key and ciphertext components involving the FDH in the same group. As a consequence, we have less flexibility in optimizing the schemes using FDHs for its large-universeness.

3.6.1 Optimized encryption

Given the list of paired key-ciphertext components, the strategy is simple: we place as many ciphertext components in the first source group as possible. Because we need to place the ciphertext components involving the FDH in $\mathbb{G}$, we also place the key components involving an FDH in $\mathbb{G}$, and thus place the ciphertext components paired with these key components in $\mathbb{H}$.

3.6.2 Optimized key generation

Similarly, given the list of paired key-ciphertext components, the strategy is simple: we place as many key components in the first source group as possible. Similarly as in the optimized encryption approach, we always place ciphertext components involving an FDH in $\mathbb{G}$, and thus place the key components paired with these ciphertext components in $\mathbb{H}$.

3.6.3 Optimized decryption

To optimize decryption, we need to take a more careful approach. First, we need to consider whether group elements need to be exponentiated during decryption, because they occur in a shared-argument pairing product (see Section 3.4), e.g., $\prod_j \hat{e}(K', C_j)^{\varepsilon_j} = \hat{e}(K', \prod_j C_j^{\varepsilon_j})$. In this case, our conversion consists of placing the shared argument in $\mathbb{H}$ and the other—which needs to be exponentiated—in $\mathbb{G}$. For all key-ciphertext component pairs that do not occur in a shared-argument pairing product, it does not matter whether the key or ciphertext component is placed in $\mathbb{G}$, as long as any potential exponentiation happens in the first source group. In these cases, we will place, by default, the ciphertext component in $\mathbb{G}$, as it is oftentimes more important to optimize the encryption algorithm than the key generation algorithm. If the application allows the use of precomputation tables for all key components, we may also choose to place the key component in $\mathbb{G}$, and perform the exponentiations with a fixed-base exponentiation.

Remark 2 (Shared exponentiations in $\mathbb{G}_T$). During decryption, the combination of a pairing operation and an exponentiation, e.g., $\hat{e}(K_j, C_j)^{\varepsilon_j}$, may be part of a larger pairing product, in which multiple key-ciphertext component pairs are exponentiated with the same value, e.g., $\hat{e}(K_{1,j}^{\varepsilon_j}, C_{1,j}) \cdot \hat{e}(K_{2,j}^{\varepsilon_j}, C_{2,j}) \cdot \hat{e}(K_{3,j}^{\varepsilon_j}, C_{3,j}) = (\hat{e}(K_{1,j}, C_{1,j}) \cdot \hat{e}(K_{2,j}, C_{2,j}) \cdot \hat{e}(K_{3,j}, C_{3,j}))^{\varepsilon_j}$. If we compute it in the first way, then we require a 3-multi-pairing and three exponentiations

in $\mathbb{G}$. If we compute it in the second way, then we require three pairing operations and one exponentiation in $\mathbb{G}_T$, which may be more efficient, depending on the curve. Furthermore, the second ordering of the pairing operations may allow the use of fixed-argument pairings [CS10, Sco11], in which case the key components need to be placed in $\mathbb{G}$. In the first ordering, we cannot use a fixed-argument pairing operation without requiring that the exponentiation is placed in $\mathbb{H}$, which likely negatively affects the computational costs (see also Remark 4). In conclusion, this illustrates that optimizing a scheme to attain the most efficient decryption is more intricate than previous conversion techniques suggested.

3.6.4 Balanced key generation/encryption

For a balanced efficiency of the key generation and encryption algorithms, we optimize the total key generation and encryption costs. We do this by considering the computational costs for each key-ciphertext component pair. For each pair, we place the component with the highest computational costs in $\mathbb{G}$, and the other in $\mathbb{H}$. For instance, if the pair (K, C) like in our example is such that the computation of K only requires a fixed-base exponentiation, and the computation of C requires a multi-base exponentiation, then we place C in $\mathbb{G}$ and K in $\mathbb{H}$. In this approach, it is also important to consider whether the pairs occur in a shared-argument pairing product during decryption. In this case, we place the shared argument in $\mathbb{H}$ and the other components in $\mathbb{G}$. Therefore, the shared argument incurs only a constant cost in $\mathbb{H}$ in the computation (during key generation or encryption), while it incurs a linear cost in $\mathbb{G}$ (during encryption or key generation), subsequently optimizing the total costs of these computations.

3.6.5 Balanced encryption/decryption

Similarly, for a balanced efficiency of the encryption and decryption algorithms, we optimize the total encryption and decryption costs. This may be a slightly more complicated endeavor than the balanced key generation/encryption approach due to the more complicated nature of the optimized decryption strategy. Like in this strategy, we need to take into account whether a key-ciphertext component pair occurs in a shared-argument pairing product or not. In this case, it is beneficial for the decryption costs to place the shared argument in $\mathbb{H}$ and the other components in $\mathbb{G}$. However, this may more negatively affect the encryption costs than that it positively affects the decryption costs. For instance, suppose that the coefficients ε_j are small, e.g., $\varepsilon_j \in \{0, 1\}$ like in [LW10]. Then, $\prod_j \hat{e}(C', K_{\rho(j)})^{\varepsilon_j}$ can be computed as $\hat{e}(\prod_j K_{\rho(j)}^{\varepsilon_j}, C')$ to minimize the decryption costs, requiring a linear number of multiplications in $\mathbb{G}$. However, this ensures that C' is in $\mathbb{H}$, and therefore likely costs at least one exponentiation in $\mathbb{H}$ instead of $\mathbb{G}$ (depending on the computational costs of C'). If the expected average costs incurred by the multiplications needed during decryption is lower than the costs incurred by computing C', we might want to place C' in $\mathbb{G}$ and place $K_{\rho(j)}$ in $\mathbb{H}$.

Remark 3 (Optimizing decryption for the OE and OK approaches). The key-ciphertext component distribution that follows from applying the OE and OK approaches may not be optimal for the order of computations performed in the decryption. For instance, consider the case that several shared-argument pairings have shared exponentiations, as in Remark 2, e.g., $\hat{e}(K, \prod_j C_{1,j}^{\varepsilon_j}) \cdot \hat{e}(K', \prod_j C_{2,j}^{\varepsilon_j}) \cdot \prod_j \hat{e}(K_{3,j}^{\varepsilon_j}, C_{3,j}) = \prod_j (\hat{e}(K, C_{1,j}) \cdot \hat{e}(K', C_{2,j}) \cdot \hat{e}(K_{3,j}, C_{3,j}))^{\varepsilon_j}$. Then, due to the distribution of the key and ciphertext components, the left-hand side—which has an optimized order for the curves considered in this work—may be less efficient to compute than the right-hand side, for some curves. For instance, doing $(2 + n)$-multi-pairing operations, two n-multiple-base exponentiations in $\mathbb{H}$ and n exponentiations in $\mathbb{G}$ may be more costly for some n than doing $3n$ pairing operations and n exponentiations in $\mathbb{G}_T$. This is not the case for the schemes and curves in this work. In any case, it does illustrate that, after the type conversion

has finished, we may have to circle back to the optimized order of computations to verify whether it is still optimized, given the distribution of components.

3.7 Example: type-converting Wat11

We explain our type-conversion techniques through an example: by converting the CP-ABE scheme by Waters (Wat11) [Wat11] from the type-I to the type-III setting. We first show how to convert the small-universe version of Wat11, and then argue how these conversions translate to the large-universe version of Wat11.

3.7.1 Wat11-I: the small-universe variant

In the type-I setting, the Wat11-I scheme [Wat11] is defined as follows:

Definition 2 (The Wat11-I-SYM scheme [Wat11])**.** The small-universe CP-ABE scheme by Waters is defined in the type-I (or: symmetric) setting as follows.

- Setup(λ): Taking as input the security parameter λ, the setup generates two groups $\mathbb{G}, \mathbb{G}_T$ of prime order p with generator $g \in \mathbb{G}$, and chooses a pairing $\hat{e}\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. The universe of attributes is $\mathcal{U}$. The setup also generates random integers $\alpha, b, b_{\text{att}} \in_R \mathbb{Z}_p$ for all att $\in \mathcal{U}$. It outputs $\text{MSK} = (\alpha, b, \{b_{\text{att}}\}_{\text{att}\in\mathcal{U}})$ as its master secret key and publishes the master public key as

$$\text{MPK} = (g, A = \hat{e}(g,g)^{\alpha}, B = g^{b}, \{B_{\text{att}} = g^{b_{\text{att}}}\}_{\text{att}\in\mathcal{U}}).$$

- KeyGen($\text{MSK}, \mathcal{S}$): On input a set of attributes $\mathcal{S}$, the algorithm generates random integers $r \in_R \mathbb{Z}_p$ and computes the secret key as

$$\text{SK}_{\mathcal{S}} = (K = g^{\alpha - rb}, K' = g^{r}, \{K_{\text{att}} = g^{r b_{\text{att}}}\}_{\text{att}\in\mathcal{S}}).$$

- Encrypt($M, \text{MPK}, \mathbb{A}$): A message $M \in \mathbb{G}_T$ is encrypted under access policy $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$, $\rho\colon \{1, ..., n_1\} \to \mathcal{U}$ by generating random integers $s, s_i, v_j \in_R \mathbb{Z}_p$ for all $i \in [n_1]$ and $j \in [2, n_2]$, and computing the ciphertext as

$$\text{CT}_{\mathbb{A}} = (C = M \cdot A^{s}, C' = g^{s}, \{C_{1,j} = B^{\lambda_j} B_{\rho(j)}^{s_j}, C_{2,j} = g^{s_j}\}_{j\in[1,n_1]}),$$

  such that λ_i denotes the i-th entry of the vector $\mathbf{A} \cdot (s, v_2, ..., v_{n_2})^{\intercal}$.

- Decrypt($\text{SK}_{\mathcal{S}}, \text{CT}_{\mathbb{A}}$): Suppose that $\mathcal{S}$ satisfies $\mathbb{A}$, and suppose $\Upsilon = \{j \in \{1, ..., n_1\} \mid \rho(j) \in \mathcal{S}\}$, such that $\{\varepsilon_j \in \mathbb{Z}_p\}_{j\in\Upsilon}$ exist with $\sum_{i\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$. Then, the plaintext M is retrieved by computing

$$C / \left(\hat{e}(C', K) \cdot \prod_{j\in\Upsilon} \left(\hat{e}(C_{1,j}, K') / \hat{e}(C_{2,j}, K_{\rho(j)}) \right)^{\varepsilon_j} \right).$$

3.7.2 Listing the key and ciphertext components

To convert the scheme to the type-III setting, we first consider which key components need to be paired with which ciphertext components (see Table 2). In this way, we can ensure that each pair has exactly one component in each source group.

Table 2: A list of the key-ciphertext component pairs and their costs incurred in computing them during key generation and encryption, in terms of fixed-base exponentiations (FBE) and two-base exponentiations (2-MBE). For the decryption costs, we list whether the pairing is indexed and whether it needs to be exponentiated.

Key component	Costs	Ciphertext component	Costs	Decryption	
				Indices	Exponentiation
K	FBE	C'	FBE	-	-
K'	FBE	$C_{1,j}$ $(j \in [n_1])$	2-MBE	$j \in \Upsilon \subseteq [n_1]$	ε_j
K_{att} $(\text{att} \in \mathcal{S})$	FBE	$C_{2,j}$ $(j \in [n_1])$	FBE	$j \in \Upsilon \subseteq [n_1]$	ε_j

3.7.3 Optimized encryption and key generation

For the optimized encryption and key generation approaches, it is clear in which source groups the components need to be placed. Because the scheme does not involve hashing into the group, we have much freedom. For the optimized encryption approach, we can simply place all ciphertext components in $\mathbb{G}$, and the key components in $\mathbb{H}$. Conversely, for the optimized key generation approach, we can place all key components and $\mathbb{G}$ and the ciphertext components in $\mathbb{H}$.

3.7.4 Balanced key generation/encryption

For a more balanced approach in the efficiency of key generation and encryption, we take into account the number of components on the "other side of the pairing" during decryption. For instance, if one places K' in $\mathbb{G}$, then all $C_{1,j}$ need to be placed in $\mathbb{H}$, blowing up the encryption costs considerably. Hence, we place K' in $\mathbb{H}$ and $C_{1,j}$ in $\mathbb{G}$ to make the key generation and encryption costs more balanced. For the $(K_{\text{att}}, C_{1,j})$ key-ciphertext component pair, there is no such trade-off, as both cost one fixed-base exponentiation. In this case, we favor the encryption algorithm (as mentioned in Section 3.6), as it is probably run more often than the key generation algorithm. (Note, however, that one may want to take a different approach, and favor the key generation over the encryption algorithm instead.) For this reason, we place $C_{1,j}$ in $\mathbb{G}$ and K_{att} in $\mathbb{H}$. Thus, the optimized encryption and the balanced key generation/encryption efficiency approaches yield the same constructions, since all key components are placed in $\mathbb{H}$.

3.7.5 Optimized decryption

To optimize the decryption algorithm, we need to consider the best order of the operations performed during decryption, i.e.,

$$C / \left(\hat{e}(C', K) \cdot \left(\hat{e} \left(\prod_{j \in \Upsilon} C_{1,j}^{\varepsilon_j}, K' \right) / \prod_{j \in \Upsilon} \hat{e} \left(C_{2,j}^{\varepsilon_j}, K_{\rho(j)} \right) \right) \right).$$

Because a pairing operation is usually one of the most expensive operations, we want to minimize the use of these. Consequently, we use a shared-argument pairing and place the exponentations in $\mathbb{G}$ (Section 3.4). To ensure this, it is therefore better to put $C_{1,j}$ in $\mathbb{G}$ and K' in $\mathbb{H}$. For the other product of pairing operations, i.e., $\prod_{j \in \Upsilon} \hat{e} \left(C_{2,j}^{\varepsilon_j}, K_{\rho(j)} \right)$, it does not matter in which groups $K_{\rho(j)}$ and $C_{2,j}$ live, as we can exponentiate in $\mathbb{G}$, regardless of whether $K_{\rho(j)}$ or $C_{2,j}$ is in it. If, on the other hand, one is willing to use precomputation tables for all key components K_{att}, then we can speed up decryption by placing K_{att} in $\mathbb{G}$. Because this may require a large amount of precomputation space, this may, however, not be desirable in practice. Hence, we do not use precomputation, and, as mentioned in Section 3.6, we choose to favor the encryption efficiency over the key generation efficiency. We thus place the ciphertext component $C_{2,j}$ in $\mathbb{G}$ and K_{att} in $\mathbb{H}$.

Table 3: The distributions of the key and ciphertext components of Wat11-I over the groups $\mathbb{G}$ and $\mathbb{H}$, for each optimization approach, i.e., optimized encryption (OE), optimized key generation (OK), optimized decryption (OD), balanced key generation/encryption (BKE) and balanced encryption/decryption (BED).

Key	**Group**					**Ciphertext**	**Group**				
component	OE	OK	OD	BKE	BED	**component**	OE	OK	OD	BKE	BED
K	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	C'	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$
K'	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$C_{1,j}$	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$
K_{att}	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$C_{2,j}$	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$

Remark 4 (Fixed-argument pairings). We can hardly speed up—if we can, at all—the decryption algorithm by using a fixed-argument pairing operation, which decreases the Miller loop costs by 37% [CS10]. This would however require us to place the key components in the first source group, and in the case of $\hat{e}(\prod_{j\in\Upsilon} C_{1,j}^{\varepsilon_j}, K')$—which would thus be computed as $e(K', \prod_{j\in\Upsilon} C_{1,j}^{\varepsilon_j})$ in the type-III setting—this may slow down the computation, as we would require $|\Upsilon|$ exponentiations in $\mathbb{H}$ instead of in $\mathbb{G}$. Depending on the expected number of exponentiations, the decrease in computational costs required by performing the pairing operation may be outweighed by the additional costs incurred by the exponentiations. For the pairings involving $C_{2,j}$ and $K_{\rho(j)}$, we might be able to benefit from using a fixed-argument pairing, on the condition that we can do the exponentiation in $\mathbb{G}$ as well. Otherwise, the speed-up that is obtained from using a fixed-argument pairing may be outweighed by additional overhead that the exponentiation in $\mathbb{H}$ incurs over an exponentiation in $\mathbb{G}$. However, it is unclear if it is possible to adjust existing algorithms [CS10] to facilitate both using a fixed-argument pairing and doing an exponentiation in $\mathbb{G}$. Furthermore, because precomputation needs to be done for each K_{att}, this requires the storage of possibly thousands of points.

3.7.6 Balanced encryption/decryption

Because the type conversion is the same for the optimized encryption and decryption approaches, it is, by extension, also the same for the balanced encryption/decryption approach. This is because, for each key-ciphertext component pair, we chose the best distribution of the two source groups with respect to the encryption and decryption efficiency. This therefore also yields the best efficiency trade-offs for the two.

3.7.7 Overview of the distributions for each optimization approach

In Table 3, we summarize the distributions of the key and ciphertext components for each approach. In particular, it shows that the distributions are the same for the optimized encryption, optimized decryption, balanced key generation/encryption and balanced encryption/decryption approaches.

3.7.8 Wat11-IV: the large-universe variant

The large-universe variant of the Waters scheme (Wat11-IV) [Wat08] replaces the generator $g^{b_{\text{att}}}$ by the output of a hash function, i.e., $\mathcal{H}(\text{att})$, where $\mathcal{H}\colon \{0,1\}^* \to \mathbb{G}$ denotes a hash function that maps arbitrary strings randomly in the group $\mathbb{G}$. The advantage of this is that the scheme can support any arbitrary string as attribute without requiring to change the master public key to be updated. Compared to the original, small-universe variant of the scheme, little needs to change. However, the use of a hash into the group gives us a little less freedom in the conversion from the type-I to the type-III setting. By Remark 1, we necessarily place the key and ciphertext components involving the hash in the same source group. For the optimized encryption and optimized key generation approaches, it is

Table 4: The distributions of the key and ciphertext components of Wat11-IV over the groups $\mathbb{G}$ and $\mathbb{H}$, for each optimization approach.

Key component	**Group**					**Ciphertext component**	**Group**				
	OE	OK	OD	BKE	BED		OE	OK	OD	BKE	BED
K	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	C'	$\mathbb{G}$	$\mathbb{H}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$
K'	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$C_{1,j}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$
K_{att}	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$C_{2,j}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$	$\mathbb{H}$

evident that these therefore need to be placed in the first source group (see Table 1). For optimized decryption, it follows from the $(K', C_{1,j})$ key-ciphertext component pair—which occur in a shared-argument pairing product—that the components involving the hash need to be placed in the first source group. That is, K' needs to be placed in $\mathbb{H}$ because it is the shared argument in the shared-argument pairing product, while $C_{1,j}$—which involves the hash—needs to be placed in $\mathbb{G}$. By extension, this requires K_{att} to be placed in $\mathbb{G}$ as well, because it involves a hash. The distribution of the key and ciphertext components is therefore almost entirely fixed for all optimization approaches. We can only choose the distribution of components K and C'. Because these incur a constant cost, the key generation, encryption and decryption costs are almost entirely fixed as well. Table 4 describes the distributions of the components over the two source groups of Wat11-IV. As it shows, the distributions are the same for the pairs $(K', C_{1,j})$ and $(K_{\text{att}}, C_{2,j}$. For the pair (K, C'), the distribution is the same as for Wat11-I.

3.8 Selecting the best elliptic curve for a specific goal

To fully optimize a scheme, it is important that the best curve is selected for each scheme and for each design goal. In general, this may not be the same curve for each scheme and each design goal, as the different choices of curves provide different trade-offs in efficiency. For instance, BN382 provides efficient hashing in the two source groups, while BLS12-381 provides efficient exponentiations and pairing operations [Ara17]. It may therefore be the case that ABE schemes that require many hashing operations are more efficient on the BN382 curve, while schemes that do not require these perform better on BLS12-381 curves. More generally, curves exist that provide more efficient arithmetic in $\mathbb{G}$ [CDS20] or that provide more efficient products of pairings [GF16]. However, these are unfortunately not supported by RELIC.

To determine the optimal curve for each scheme and each design goal, we compare the efficiency of the scheme on several curves providing the same level of security. To this end, we compare the computational costs of several ABE schemes on the curves providing the same level of security supported by RELIC.

4 Benchmarking

We show how our framework can be applied to several existing ABE schemes.

4.1 The schemes

In this work, we analyze and implement several selectively secure ciphertext-policy ABE schemes. We have motivated our choice to implement CP-ABE schemes in Section 1.2, and we will motivate the choice to implement selectively secure schemes below. The schemes that we implement are the Waters schemes (the previously considered small and large universe variants called Wat11-I and Wat11-IV) [Wat11, Wat08], the Rouselakis-Waters large-universe scheme without random oracles (RW13) [RW13] and the Agrawal-Chase multi-use scheme (AC17) [AC17b].

4.1.1 The Wat11 schemes

The Wat11 schemes [Wat11, Wat08] are the ciphertext-policy variants of the first ABE schemes [SW05, GPSW06a] and the selectively secure and more efficient variants of its fully secure counterpart [LOS+10]. In general, the structure of the scheme is important, as it provides the structure for many follow-up schemes, e.g., [Att14, Wee14, KW19], which provide better security guarantees than Wat11. Furthermore, Wat11-IV, i.e., the large-universe variant using an FDH, is also implemented in OpenABE [Zeu20]. We have chosen to analyze this scheme mainly for its popularity.

4.1.2 The RW13 scheme

The RW13 scheme [RW13] is the selectively secure and ciphertext-policy counterpart of the fully secure KP-ABE scheme by Lewko and Waters [LW11b]. Like Wat11-IV, it supports large universes. However, instead of using an FDH to generate a public key for each attribute string, it uses a special type of hash, first introduced in identity-based encryption by Boneh and Boyen [BB04]. In particular, this hash does not need to be modeled as a random oracle [BR93] in the security proof. Much like Wat11, it is an important scheme due to its many follow-up schemes, e.g., [Att14, HW14, CGKW18, KW19, Att19]. Yet, despite its theoretical popularity, it is not often considered in efficiency comparisons with other schemes. This is one of the main reasons why we analyze its efficiency.

4.1.3 The AC17 scheme

The AC17 scheme [AC17b] is the multi-use variant of the second CP-ABE scheme by Waters in [Wat11] (Wat11-II) in the selective-security setting. In the full-security setting, AC17 is the multi-use variant of the one-use scheme by Attrapadung [Att14]. A somewhat related scheme is FAME [AC17a], which is single use, supports large universes and is derived from the small-universe CP-ABE scheme by Chen, Gay and Wee [CGW15]. The advantage of these single-use schemes is that they require fewer pairing operations during decryption, making decryption more efficient than e.g., Wat11-I. This is one of the main reasons why standardization institutes such as ETSI (European Telecommunications Standards Institute) [ETS18] have expressed interest in this scheme. However, the drawback of these single-use schemes is that each attribute may only occur once in each access structure. To support both multi-use access structures and to benefit from an efficient decryption algorithm, AC17 combines the techniques of Wat11-I and Wat11-II. In this way, AC17 is more efficient than Wat11-I and more flexible than Wat11-II and FAME. Much like for the Wat11-I scheme, we also consider the large-universe variant of AC17 using an FDH (which subsequently yields security in the random oracle model [BR93]) in our analysis. We have chosen to analyze this scheme because of its flexibility and efficiency.

4.1.4 On the security of these schemes

We briefly discuss the security of the implemented schemes.

Selective versus full security. As mentioned, we consider the selectively secure variants of Wat11, RW13 and AC17, because this yields a cleaner comparison of the schemes on a structural level. In contrast, many fully secure variants of these schemes exist, which are similar to Wat11, RW13 and AC17 on a structural level, but these differ in the underlying groups. For instance, LOSTW10 [LOS+10] is a fully secure variant of Wat11 and is instantiated in composite-order groups. For the same security level, LOSTW10 performs one to two orders of magnitude worse than Wat11, which can be instantiated in a prime-order group [Gui13]. Other fully secure variants of Wat11 [Att19, KW19], which allow for instantiation in prime-order groups, might simply use different underlying group structures,

which may affect the efficiency as well. However, this difference in efficiency might then be (partially) attributed to the choice of underlying groups, and not necessarily the different structures of the schemes. For a fair comparison of two structurally different schemes, one could first compare the efficiency of the selectively secure variants, instantiated in prime-order groups. Then, one can extrapolate the comparison to the full-security setting by considering the efficiency of the chosen underlying groups, which can be chosen the same if all the compared schemes have a fully secure counterpart in the same framework, e.g., [Att14, Wee14, CGW15, Att16, Att19]. Note that this is the case for Wat11, RW13 and AC17, which all have instantiations in the pair encodings framework [AC17b, Att19].

Security in idealized models. The security of the implemented schemes depends on idealized models, such as the random oracle model (ROM) [BR93] and the generic group model (GGM) [Sho97]. In particular, the large-universe variants of Wat11 and AC17 model the FDH as a random oracle in the proofs. Furthermore, the security of all three schemes depends on a q-type assumption [BBG05, Boy08]. These are assumptions that are parametrized in some parameter q, which in turn depends on one or more system parameters of the scheme. Typically, the q-type assumptions that are frequently used in selective-security proofs grow stronger as q increases. Specifically, Cheon [Che06] has shown that the security strength decreases by roughly $\log_2(\sqrt{n_2})$ bits, where n_2 denotes the maximum number of columns in an LSSS access structure, used during encryption. For instance, if the maximum policy size is 100 (like in this work), then we lose at least 4 bits (due to Cheon's attack) and at most 11 bits of security (in GGM, due to Boneh, Boyen and Goh's asymptotic lower bound [BBG05, Boy08]). If we use curves in the $[129, 135]$-bit security range, such as BLS12-446 or BN446, which provide 132 bits of security [GS19], then we still have roughly 128 bits of security (and at least 121 bits, considering the asymptotic lower bound). For curves in the $[125, 128]$-bit security range such as BLS12-381 or BN382, it should be investigated whether the resulting security strength (i.e., 115-122 bits) is sufficient for some given practical setting. In any case, the fully secure variants of these schemes in the pair encodings framework [AC17b, Att19] are provably secure under different q-type assumptions. For these assumptions, attacks such as those by Cheon [Che06] do not exist yet. Hence, it is unclear if the security strength is impacted for the fully secure variants of the schemes at all.

On the security of the type-converted schemes. An additional advantage of considering the selectively secure variants of the schemes is that their security proofs carry over to their type conversions as well. That is, in all of their selective security proofs [Wat11, RW13, AC17b], the inputs to a q-type assumption are embedded in the key and ciphertext components, such that the q-type assumption can be broken if the scheme's security can be broken. This q-type assumption is typically shown to be generically secure [BBG05, Boy08]. By extension, any type-converted variant of this q-type assumption with the same number of parameters is also generically secure with the same security loss in the GGM. In the worst case, they have twice as many parameters, and thus lose at most one additional bit of security in the GGM [Boy08].

4.2 The optimized type-converted constructions in short notation

Because we have five optimization approaches and five schemes—two small-universe and three large-universe schemes—we potentially obtain twenty-five different constructions to analyze. To effectively highlight the differences, we use the representation of ABE as introduced in Section 2.5, which only consists of the main differences: the master public key, the secret keys, the ciphertexts and decryption. For each scheme, we use $\lambda_j = \mathbf{A}_j \mathbf{v}^\intercal$ as the j-th share of the secret s, where $\mathbf{A}$ is the $n_1 \times n_2$ LSSS matrix and

$\mathbf{v} = (s, v_2, ..., v_{n_2}) \in_R \mathbb{Z}_p^{n_2}$ is a vector with random entries, both used during encryption. During decryption, we use $\Upsilon = \{j \in [1, n_1] \mid \rho(j)\}$ and $\{\varepsilon_j\}_{j\in\Upsilon}$ such that $\sum_{j\in\Upsilon} \varepsilon_j \lambda_j = s$ (Section 2.2.1). We also distinguish between variables that the authority—that generates the keys—does and does not know. By placing a bar above a variable, e.g., $\bar{b}_{\text{att}}$, we indicate that the authority does not know b_{att}. For instance, by using an FDH to generate b_{att}, the implicit exponent b_{att} in $\mathcal{H}(\text{att}) = g^{b_{\text{att}}}$ is unknown, and is thus represented as $[\bar{b}_{\text{att}}]_{\mathbb{G}}$.

In this section, we use the following naming convention. For schemes that already have an implementation, we append the name of the scheme with the suffix "CP", i.e., ciphertext-policy, to distinguish the schemes from their potential key-policy and other counterparts. For our type conversions, we use as suffix the appropriate acronym associated with the applied optimization approach. For example, Wat11-IV-OE is the name of the variant of Wat11-IV that is optimized with respect to the encryption algorithm. For conciseness, we do not use "CP" in the suffix of the names of our optimizations.

4.2.1 Wat11-IV

We obtain two different type-converted constructions of Wat11-IV, which is the large-universe variant of Wat11. As we mentioned in Section 3.7.8, the type conversion is essentially fixed for almost all variables, except for the pair $([\alpha - rb], [s])$.

Wat11-IV-OE. We define Wat11-IV-OE as the type-converted variant of Wat11-IV with the most optimized encryption and decryption algorithms, and the most balanced key generation/encryption and encryption/decryption algorithms as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{H}}, \{[r\bar{b}_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{G}}, \{[\lambda_j b + s_j \bar{b}_{\rho(j)}]_{\mathbb{G}}, [s_j]_{\mathbb{H}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m+\alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon} [\lambda_j b + s_j \bar{b}_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{j\in\Upsilon} e([r\bar{b}_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_j]_{\mathbb{H}})$.

Wat11-IV-OK. We define Wat11-IV-OK as the type-converted variant of Wat11-IV with the most optimized key generation algorithm as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{H}}, \{[r\bar{b}_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_j \bar{b}_{\rho(j)}]_{\mathbb{G}}, [s_j]_{\mathbb{H}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m+\alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon} [\lambda_j b + s_j \bar{b}_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{j\in\Upsilon} e([r\bar{b}_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_j]_{\mathbb{H}})$.

4.2.2 RW13

We obtain two different type-converted constructions of RW13. In general, owing to the lack of an FDH to obtain the large-universe property, RW13 is more flexible to convert. For example, for an optimized encryption (resp. key generation) algorithm, all ciphertexts (resp. keys) should be placed in $\mathbb{G}$. For an optimized decryption, we need to observe the order of computations and the efficiency of the group operations to determine the best conversion.

RW13-OK. We define RW13-OK as the type-converted variant of RW13 with the most optimized key generation algorithm as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{H}}, [b_0]_{\mathbb{H}}, [b_1]_{\mathbb{H}}, [b']_{\mathbb{H}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{G}}, \{[r_{\text{att}}(b_1 x_{\text{att}} + b_0) + rb']_{\mathbb{G}}, [r_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_j b']_{\mathbb{H}}, [s_j(\rho(j)b_1 + b_0)]_{\mathbb{H}}, [s_j]_{\mathbb{H}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e([r]_{\mathbb{G}}, \prod_{j\in\Upsilon}[\lambda_j b + s_j b']_{\mathbb{G}}^{-\varepsilon_j}) \cdot \prod_{j\in\Upsilon}\left(e([r_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [s_j(b_1\rho(j) + b_0)]_{\mathbb{H}}) \cdot e([r_{\rho(j)}(b_1\rho(j) + b_0) + rb']_{\mathbb{G}}^{\varepsilon_j}, [s_j]_{\mathbb{H}})\right)$.

RW13-OE. We define RW13-OE as the type-converted variant of RW13 with the most optimized encryption algorithms as follows. We show that it also has the most efficient decryption algorithm, as well as the most balanced key generation/encryption and encryption/decryption.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}}, [b_0]_{\mathbb{G}}, [b_1]_{\mathbb{G}}, [b']_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{H}}, \{[r_{\text{att}}(b_1 x_{\text{att}} + b_0) + rb']_{\mathbb{H}}, [r_{\text{att}}]_{\mathbb{H}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{G}}, \{[\lambda_j b + s_j b']_{\mathbb{G}}, [s_j(\rho(j)b_1 + b_0)]_{\mathbb{G}}, [s_j]_{\mathbb{G}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_j b']_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{j\in\Upsilon}\left(e([s_j(b_1\rho(j) + b_0)]_{\mathbb{G}}^{-\varepsilon_j}, [r_{\rho(j)}]_{\mathbb{H}}) \cdot e([s_j]_{\mathbb{G}}^{\varepsilon_j}, [r_{\rho(j)}(b_1\rho(j) + b_0) + rb']_{\mathbb{H}})\right)$.

Note that the decryption algorithm is already optimized. Specifically, $e(\prod_{j\in\Upsilon}[\lambda_j b + s_j b']_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}})$ is the most efficient, because the multi-base exponentiation happens in $\mathbb{G}$ and not in $\mathbb{H}$, meaning that $[\lambda_j b + s_j b']$ necessarily needs to be placed in $\mathbb{G}$ and $[r]$ in $\mathbb{H}$. Furthermore, for $e([s_j(b_1\rho(j) + b_0)]_{\mathbb{G}}^{-\varepsilon_j}, [r_{\rho(j)}]_{\mathbb{H}})$ and $e([s_j]_{\mathbb{G}}^{\varepsilon_j}, [r_{\rho(j)}(b_1\rho(j) + b_0) + rb']_{\mathbb{H}})$, it does not matter in which groups the components are placed for the decryption efficiency. For all curves considered in this work, it is more efficient to exponentiate in $\mathbb{G}$. For these pairing operations, this can always be done in $\mathbb{G}$, regardless of whether the key or ciphertext component is placed in $\mathbb{G}$.

Moreover, this variant is also the most balanced. Because this variant is already optimized with respect to its encryption and decryption efficiency, it logically also optimizes the total encryption and decryption costs, and thus also provides the variant with the most balanced encryption-decryption efficiency. Furthermore, it has the most balanced key generation-encryption efficiency, because each key component can be generated with a single fixed-base exponentiation. In contrast, the associated ciphertext component costs *at least* a single fixed-base exponentiation, and is thus at least as expensive.

4.2.3 AC17

We obtain three different type-converted constructions of the small-universe variant of AC17, and two type-converted constructions of the large-universe variant of AC17. During encryption, we also include an additional function $\tau\colon [n_1] \to [\mu]$, where μ denotes the maximum number of uses of each attribute, such that τ is a mapping where for all $j, j' \in [n_1]$ for which $\rho(j) = \rho(j')$, it holds that $\tau(j) \neq \tau(j')$, i.e., each occurrence of the same attribute is mapped to a different integer in $[\mu]$. For the small-universe variants, we also include the universe of attributes $\mathcal{U}$ in the setup. Note that, in the conversions of the small-universe variants, we have much freedom in how we convert the schemes. In contrast, in the large-universe variant, the FDH almost entirely fixes the conversion, much like in the Wat11-IV scheme.

AC17-OE. We define AC17-OE as the type-converted variant of the small-universe variant of AC17 with the most optimized encryption algorithm as follows. Much like in the Wat11-I scheme, we place all ciphertext components in $\mathbb{G}$.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}}, \{[b_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{U}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{H}}, \{[rb_{\text{att}}]_{\mathbb{H}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{G}}, \{[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{G}}\}_{j\in[1,n_1]}, \{[s_l]_{\mathbb{G}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{l\in[\mu]} e([s_l]_{\mathbb{G}}, \prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{H}}^{\varepsilon_j})$.

AC17-OK. We define AC17-OK as the type-converted variant of the small-universe variant of AC17 with the most optimized key generation algorithm as follows. Much like in the Wat11-I scheme, we place all key components in $\mathbb{G}$.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{H}}, \{[b_{\text{att}}]_{\mathbb{H}}\}_{\text{att}\in\mathcal{U}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{G}}, \{[rb_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{H}}\}_{j\in[1,n_1]}, \{[s_l]_{\mathbb{H}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e([r]_{\mathbb{G}}, \prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{H}}^{-\varepsilon_j}) \cdot \prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

AC17-OD. We define AC17-OD as the type-converted variant of the small-universe variant of AC17 with the most optimized decryption algorithm as follows. We also show that this is the variant with the most balanced key generation/encryption and encryption/decryption efficiency. To optimize the decryption efficiency of AC17, we first consider the order of computations in the decryption algorithm of AC17-OK, i.e.,

$$[m + \alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e([r]_{\mathbb{G}}, \prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{H}}^{-\varepsilon_j}) \cdot \prod_{l\in[m]} e\left(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}}\right).$$

First, we observe that for the $e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}})$ part, the component distribution does not matter. Hence, we place the ciphertext component $[s]$ in $\mathbb{G}$ and the key component in $\mathbb{H}$. For the $e([r]_{\mathbb{G}}, \prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{H}}^{-\varepsilon_j})$ part, the distribution matters: placing the product of exponentiations over the ciphertext components $[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]$ in $\mathbb{G}$ would make the algorithm faster. This is also the case for $\prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}}, \{[b_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{U}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{H}}, \{[rb_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{G}}, \{[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{G}}\}_{j\in[1,n_1]}, \{[s_l]_{\mathbb{H}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

Note that this variant also has the most balanced key generation-encryption, and most balanced encryption-decryption efficiency. For the BKE variant, this follows quite simply. For the pair $([\alpha - rb], [s])$, the costs are the same for the key and ciphertext component, so we place the ciphertext component in $\mathbb{G}$. The pairs $([r], [\lambda_j b + s_{\tau(j)} b_{\rho(j)}])$ have a linear number of ciphertext components, and the pairs $([rb_{\rho(j)}], [s_l])$ have a linear number of key components. Therefore, in both cases, we place the linear costs in $\mathbb{G}$.

For the BED variant, this follows simply for the pairs $([\alpha - rb], [s])$ and $([r], [\lambda_j b + s_{\tau(j)} b_{\rho(j)}])$, which have the same distributions in the optimized encryption and optimized decryption variants, and are thus also optimized in the total costs. For the pairs $([rb_{\rho(j)}], [s_l])$, we consider the associated encryption and decryption costs. For encryption, we require m fixed-base exponentiations. The decryption costs can be upper bounded by $|\Upsilon|$ exponentiations, and lower bounded by one $|\Upsilon|$-base exponentiation (for $\mu = 1$), in the group in which the key component lives. Because, generally, we assume that $|\Upsilon| > \mu$, the costs of the $|\Upsilon|$-base exponentiation dominate those of the μ fixed-base exponentiations. To minimize the total costs, we thus place $[rb_{\rho(j)}]$ in $\mathbb{G}$ and $[s_l]$ in $\mathbb{H}$.

AC17-LU-OE. We define AC17-LU-OE as the type-converted variant of the large-universe variant of AC17 with the most optimized encryption as follows. This variant also has the most efficient decryption algorithm, and the most balanced key generation/encryption and encryption/decryption efficiency. Much like for Wat11-IV (Section 3.7.8), the distribution of AC17-LU is almost entirely fixed because of the FDH.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{H}}, \{[r\bar{b}_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{G}}, \{[\lambda_j b + s_{\tau(j)}\bar{b}_{\rho(j)}]_{\mathbb{G}}\}_{j\in[1,n_1]}, \{[s_l]_{\mathbb{H}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)}\bar{b}_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[r\bar{b}_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

Note that, in general, the only non-fixed pair is $([\alpha - rb], [s])$. For the optimized encryption variant, we place the ciphertext component $[s]$ in $\mathbb{G}$, and for the optimized key generation variant, we place the key component in $\mathbb{G}$. For the decryption efficiency, the distribution does not matter, and therefore, we place the ciphertext component in $\mathbb{G}$ for the optimized decryption approach, but also for the most balanced encryption-decryption approach. Similarly, the total costs of the key generation and encryption algorithms are the same for the two possible distributions, and thus, we also place the ciphertext component in $\mathbb{G}$ for the balanced key generation-encryption approach.

AC17-LU-OK. We define AC17-LU-OK as the type-converted variant of the large-universe variant of AC17 with the most optimized key generation as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{H}}, \{[r\bar{b}_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_{\tau(j)}\bar{b}_{\rho(j)}]_{\mathbb{G}}\}_{j\in[1,n_1]}, \{[s_l]_{\mathbb{H}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)}\bar{b}_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[r\bar{b}_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

4.2.4 Comparing our type conversions with existing implementations

We can now compare all the various optimization approaches described above with the implementations in the literature. The type-converted schemes considered in this work have been previously implemented in Charm [AGM+13] and OpenABE [Zeu20]. Specifically, Wat11-I and RW13 have been implemented in Charm, and Wat11-IV has been implemented in OpenABE. Furthermore, FAME [AC17a]— which is related to the AC17 scheme—was previously implemented in Charm as well. We briefly compare their type conversions with ours, such that we can determine with respect to which optimization approach these implementations could be interpreted to be optimized.

Wat11-I-CP. The small-universe variant of Wat11 as presented in Charm[2] is defined as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}}, \{[b_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{U}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{H}}, \{[rb_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_j b_{\rho(j)}]_{\mathbb{G}}, [s_j]_{\mathbb{H}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m+\alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_j b_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{j\in\Upsilon} e([rb_{\rho(j)}]_{\mathbb{G}}, [s_j]_{\mathbb{H}}^{\varepsilon_j})$.

Note that this construction does not match any of our type-converted constructions of Wat11-I. (It is, however, similar to Wat11-IV-OE and Wat11-IV-OK without the use of a hash to achieve large-universeness.) The designers [AC17a] explain that their type conversions aim to balance the total work fairly between encryption and key generation. This is indeed the case, as the total costs of the key generation and encryption algorithms are the same as the total costs of our variant with a balanced key generation-encryption efficiency. The difference between the two conversions can be attributed to our "deterministic" decisions in the conversion: for each key-ciphertext component pair that incur equal costs, we place the ciphertext component in $\mathbb{G}$. In general, for such pairs, the distribution does not affect the total costs of the key generation and encryption algorithms as they are equal for the two choices.

Wat11-IV-CP. The large-universe variant of Wat11 as presented in OpenABE[3] is defined as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{H}}, \{[r\bar{b}_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_j \bar{b}_{\rho(j)}]_{\mathbb{G}}, [s_j]_{\mathbb{H}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m+\alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_j \bar{b}_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{j\in\Upsilon} e([r\bar{b}_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_j]_{\mathbb{H}})$.

Note that this construction is exactly the same as Wat11-IV-OE. Therefore, the implementation of OpenABE is optimized with respect to its encryption and decryption efficiency rather than its key generation efficiency.

RW13-CP. The variant of RW13 as presented in Charm[4] is defined as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}}, [b_0]_{\mathbb{G}}, [b_1]_{\mathbb{G}}, [b']_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{H}}, \{[r_{\text{att}}(b_1 x_{\text{att}} + b_0) + rb']_{\mathbb{G}}, [r_{\text{att}}]_{\mathbb{H}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_j b']_{\mathbb{G}}, [s_j(\rho(j)b_1 + b_0)]_{\mathbb{G}}, [s_j]_{\mathbb{H}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m+\alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot \prod_{j\in\Upsilon} \left(e([\lambda_j b + s_j b']_{\mathbb{G}}^{-1}, [r]_{\mathbb{H}}) \cdot e([s_j(b_1\rho(j) + b_0)]_{\mathbb{G}}^{-1}, [r_{\rho(j)}]_{\mathbb{H}}) \cdot e([r_{\rho(j)}(b_1\rho(j) + b_0) + rb']_{\mathbb{G}}, [s_j]_{\mathbb{H}})\right)^{\varepsilon_j}$.

[2] https://github.com/JHUISI/charm/blob/dev/charm/schemes/abenc/waters11.py
[3] https://github.com/zeutro/openabe/blob/master/src/abe/zcontextcpwaters.cpp
[4] https://sites.google.com/site/yannisrouselakis/ccs13abe

AC17-CP. The small-universe variant of AC17 with a similar key-ciphertext component distribution as FAME [AC17a] in Charm[5] is defined as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}}, \{[b_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{U}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{H}}, \{[rb_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{G}}\}_{j\in[n_1]}, \{[s_l]_{\mathbb{H}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} b_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

Note that this construction is similar to our construction AC17-OD. Therefore, this construction is optimized with respect to the decryption algorithm.

AC17-LU-CP. The large-universe variant of AC17 with a similar key-ciphertext component distribution as FAME [AC17a] in Charm is defined as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{G}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{G}}, [r]_{\mathbb{H}}, \{[r\bar{b}_{\text{att}}]_{\mathbb{G}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{H}}, \{[\lambda_j b + s_{\tau(j)} \bar{b}_{\rho(j)}]_{\mathbb{G}}\}_{j\in[1,n_1]}, \{[s_l]_{\mathbb{H}}\}_{l\in[\mu]})$.
- **Decryption:** $[m + \alpha s]_{\mathbb{G}_T} \cdot e([\alpha - rb]_{\mathbb{G}}^{-1}, [s]_{\mathbb{H}}) \cdot e(\prod_{j\in\Upsilon}[\lambda_j b + s_{\tau(j)} \bar{b}_{\rho(j)}]_{\mathbb{G}}^{-\varepsilon_j}, [r]_{\mathbb{H}}) \cdot \prod_{l\in[\mu]} e(\prod_{j\in\Upsilon\cap\tau^{-1}(l)}[rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_l]_{\mathbb{H}})$.

Note that this construction is similar to our construction AC17-LU-OK. Therefore, this construction is optimized with respect to the key generation algorithm.

4.2.5 Variants of Wat11-I with bad distributions of components

We also include two variants of Wat11-I, Wat11-I-BAD-I and Wat11-I-BAD-II, with a bad distribution of components in our analysis. This will illustrate that the choice of conversion technique does matter in any analysis, even if the arithmetic and group operations are optimized. The two implementations of this type conversion differ in whether they use optimized arithmetic or not. In particular, Wat11-I-BAD-II uses optimized arithmetic, while Wat11-I-BAD-I does not. This will illustrate that the use of optimized arithmetic speeds up the algorithms significantly. The converted scheme is defined as follows.

- **Master public key:** $([1]_{\mathbb{G}}, [1]_{\mathbb{H}}, [\alpha]_{\mathbb{G}_T}, [b]_{\mathbb{H}}, \{[b_{\text{att}}]_{\mathbb{H}}\}_{\text{att}\in\mathcal{U}})$.
- **Secret keys:** $([\alpha - rb]_{\mathbb{H}}, [r]_{\mathbb{G}}, \{[rb_{\text{att}}]_{\mathbb{H}}\}_{\text{att}\in\mathcal{S}})$.
- **Ciphertexts:** $([m + \alpha s]_{\mathbb{G}_T}, [s]_{\mathbb{G}}, \{[\lambda_j b + s_j b_{\rho(j)}]_{\mathbb{H}}, [s_j]_{\mathbb{G}}\}_{j\in[1,n_1]})$.
- **Decryption:** $[m+\alpha s]_{\mathbb{G}_T} \cdot e([s]_{\mathbb{G}}^{-1}, [\alpha - rb]_{\mathbb{H}}) \cdot e([r]_{\mathbb{G}}, \prod_{j\in\Upsilon}[\lambda_j b + s_j b_{\rho(j)}]_{\mathbb{H}}^{-\varepsilon_j}) \cdot \prod_{j\in\Upsilon} e([rb_{\rho(j)}]_{\mathbb{G}}^{\varepsilon_j}, [s_j]_{\mathbb{H}})$.

Note that the distribution of this scheme is bad, because it maximizes the number of operations in $\mathbb{H}$. In particular, all algorithms require a linear number of operations in $\mathbb{H}$. In contrast, Wat11-I-OE and Wat11-I-OK require a linear number of operations in $\mathbb{G}$ in the encryption and decryption, and key generation and decryption, respectively.

[5] https://github.com/JHUISI/charm/blob/dev/charm/schemes/abenc/ac17.py

4.3 Implementation details

We rely on the RELIC toolkit [AGM+] version 0.5[6] to implement the schemes described in Section 4.2. Multiple versions of the library were generated for the `x86-64` architecture, i.e., for curves BN256, BN382, BN466, BLS12-381, and BLS12-446. We use the gcc compiler version `10.3.0` using a Linux distribution with kernel version `5.11.0-18` and the following flags `-O3 -funroll-loops -fomit-frame-pointer -finline-small-functions -march=native -mtune=native`.

RELIC is a cryptographic library that provides efficient arithmetic implementations of prime and binary fields, bilinear maps and extension fields. Since it implements architecture-dependent code, it aims at optimizing speed. The rationale of using RELIC against other alternatives has to do with the current contributions from academia. Examples are the recent integration of the fast constant-time GCD algorithm of Bernstein and Yang [BY19] and the records RELIC has set [AFK+12]. In addition, of all libraries, RELIC supports the most curves providing at least 128 bits of security [SKSW20]. Therefore, it allows us to compare the efficiency of ABE schemes with regard to various curves. The default compilation options of RELIC for field arithmetic consist of the integrated modular addition, multiplication, squaring, Montgomery reduction and sliding window modular exponentiation. The extension field arithmetic options are based on the integrated lazy-reduced extension field arithmetic of RELIC. Finally, the compilation flags for the bilinear pairing implementation select the optimal Ate pairing [Ver10]. We use the library to perform, for instance, the optimal Ate pairing of two group elements in a parametrized elliptic curve of embedding degree 12, and to perform multi-pairing operations and simultaneous exponentations of group elements.

To optimize the number of clock cycles of our implementations, we rely on the multi-pairing operations of RELIC (`pp_map_sim_oatep`), fixed-base exponentiation via precomputation tables (`_mul_fix`), simultaneous exponentiation of multiple elements from the same group (`_mul_sim_lot`) and simultaneous exponentiation of two elements of the same group (`_mul_sim`). Note that, in the default configuration of RELIC, most of these algorithms do not run in constant time.

Our implementations contain a correctness check at the end of the decryption operation. We measure the number of clock cycles for each algorithm, i.e., the setup, key generation, encryption, and decryption. We also measured the number of clock cycles of the necessary arithmetic, for which we provided benchmarks in Section 3.3. We calculate the average of the number of clock cycles in each case over 10,000 iterations per operation.

We use the implementation of the access structures based on access trees (Section 2.2) provided by OpenABE [Zeu20]. We replace it in Section 4.6.3 by precomputed LSSS matrices (see the description of the more efficient LSSS matrices in the full version [dlPVA22]). Finally, we benchmark the implementations based on an increasing number of attributes i.e., $1, 5, 10, 20, 30, ..., 100$. For encryption and decryption, we use an AND policy that increases linearly with the number of attributes. We measure our implementations using the AMD Ryzen 7 PRO 4750 processor with power management disabled (one single core) and throttle at max. frequency (4.1 GHz).

4.3.1 A note on the `_mul_sim_lot` function

In the decryption algorithm, we use the `_mul_sim_lot` function. Since the coefficients used are typically very close to the group order, their encodings in RELIC are very small, and thus, the efficiency of this function in our implementations is not the same as depicted in Table 1. Instead, it is much faster. Furthermore, for the BLS and BN curves, the GLV [GLV01] recoding (used in the `_mul_sim_lot` function) of the exponent is different, and yields more efficient encodings for BN curves than for BLS curves. The costs of

[6] Particularly related to commit `260c9f8b423bbf74cc04fd4315dc770c85d162d8`.

Table 5: The computational costs of `_mul_sim_lot` on the curves in 10^3 cycles, where the exponents are the coefficients ε_j associated with the j-th attribute.

	In group $\mathbb{G}$			**In group $\mathbb{H}$**		
	Number of attributes			Number of attributes		
Curve	2	10	100	2	10	100
BN254	8	37	398	88	339	5285
BLS12-381	153	566	3466	163	778	10216
BN382	11	56	504	248	929	10262
BLS12-446	208	574	4047	214	1021	14546
BN446	13	67	564	321	886	8491

`_mul_sim_lot` for our specific inputs are listed in Table 5. In future implementations, it may be better to replace the `_mul_sim_lot` by a custom function, or replace the access trees by the more efficient LSSS matrices as considered in Section 4.6.3. Converting Boolean formulas into these matrices yields exponentiations with $\varepsilon_j \in \{0, 1\}$.

4.4 Memory footprint

We analyze the memory footprint of the data structures in our implementation of the setup, key generation, encryption and decryption algorithms by providing upper bounds on their memory consumption. This analysis illustrates that, for computationally powerful devices such as computers and smartphones, the memory footprint is reasonable. Roughly, our implementations use two kinds of data structures that are loaded in working memory: precomputation tables, and regular storage costs (such as the keys and ciphertexts, and policies and sets of attributes). In addition, some of our computations use temporary data structures to store intermediate results, but the overhead incurred by these is small, i.e., less than one KiB. The maximum overhead incurred by the RELIC functions depends mostly on the regular storage costs, and is thus upper-bounded by those costs.

4.4.1 Regular storage costs

A large part of the memory consumption is attributed to the regular storage costs, i.e., the sets, access policies, keys and ciphertexts. The algorithms use the following structures:

- Setup: $\mathrm{MPK}, \mathrm{MSK}$;
- Key generation: $\mathrm{MPK}, \mathrm{MSK}, \mathrm{SK}, \mathcal{S}$;
- Encryption: $\mathrm{MPK}, \mathrm{CT}, \mathbb{A}$;
- Decryption: $\mathrm{MPK}, \mathrm{SK}, \mathcal{S}, \mathrm{CT}, \mathbb{A}$.

The sizes of the keys and ciphertexts can be observed directly from their descriptions in Section 4.2 and the used elliptic curve. The sizes of the sets and policies depend on the representation of the attributes. Furthermore, we analyze the size of the access structures (i.e., trees or matrices) associated with the policies in more detail below.

Access trees. For almost all schemes, we use the OpenABE access tree structure. In OpenABE[7], one node in the tree contains the following information: node type (i.e., whether it is a leaf or a gate), the threshold value, the number of children, pointers to the children (we assume in this case the number of children is always two, each pointer being 8 bytes in a 64-bit architecture), node identifiers (i.e., prefix, label and index) and two auxiliary variables. Let $\ell_{\mathsf{node}} \approx 48$ bytes denote the size of the node, then the total costs incurred by the access tree are roughly $(2 \cdot |\mathbb{A}| - 1) \cdot \ell_{\mathsf{node}}$, because each operator (e.g., AND) and each attribute in the policy are assigned to a node.

[7] https://github.com/zeutro/openabe/blob/a58892b32bae20b012f352760c864b1a4ca9600d/src/include/openabe/utils/zpolicy.h

LSSS matrices. The size of the LSSS matrix in RW13-OE-LSSS (Section 4.6.3) depends on the number of attributes and operators in the policy. To obtain a realistic upper bound on the matrix size, we assume that all operators are AND—which yields the largest matrices—and each entry is represented by as an element in $\mathbb{Z}_p$. The maximum size is thus $|\mathbb{A}|^2 \cdot \ell_{\mathbb{Z}_p}$, where $\ell_{\mathbb{Z}_p}$ denotes the length of an element in $\mathbb{Z}_p$.

4.4.2 Precomputation tables

Our implementations use precomputation tables for the generators of the source groups $\mathbb{G}$ and $\mathbb{H}$ to speed up the exponentiations. For the RW13 implementations, we also generate precomputation tables for the (four) other public keys. In RELIC, the default configuration for fixed-base exponentiation uses the single-table comb method [LL94], and yields precomputation tables of 16 points (in affine coordinates) for all curves.

4.4.3 Upper bounds on the working memory consumption

Table 6 summarizes our analysis of the memory footprint. In particular, Table 6a lists the storage costs for each implemented scheme. Table 6b provides upper bounds on the total memory consumption for the BLS12-381 and BN446 curves, of which the latter yields the highest memory consumption. Note that all implementations fit easily in RAM of computationally powerful devices such as computers and smartphones.

4.5 Performance analysis of our implementations

We analyze the performance of the implementations in our framework and compare it with those in existing frameworks such as Charm [AGM+13] and OpenABE [Zeu20]. Our goal with this comparison is not necessarily to illustrate that our implementations are faster than those of Charm and OpenABE. Rather, we want to show that the choice of optimization approach as well as the use of all available optimized arithmetic influences this analysis. Not consistently and systematically using these may result in an unfair comparison of two schemes.

There are two main differences between our implementations and those in Charm and OpenABE. First, in contrast to Charm and OpenABE, we have removed all abstractions between the scheme and the used arithmetic. Therefore, we can use all available optimizations to accelerate the scheme as much as possible. This also includes constructing and evaluating the access structures, which we have optimized as well. Note that this is not a concern for our comparisons in this section, as the additional overhead incurred by the construction and evaluation of access structures is the same for each scheme, while it considerably simplifies the implementation of the schemes. In this way, we evaluate the efficiency of the schemes on a structural level. The additional overhead incurred by the construction and evaluation of the access structures can then be measured separately to obtain a more complete understanding of the implementation's efficiency in practice. Second, another difference is that our implementations follow a clearly articulated design rationale: we have optimized each scheme with respect to several optimization approaches. This may heavily influence the efficiency of a scheme, and thus the subsequent comparison.

4.5.1 Linear costs of the schemes

We show that the computational costs of all the schemes are linear (see Figure 3). Hence, for any efficiency analysis, it suffices to compare the computational costs for small sets of attributes, e.g., 1 and 10, and for a large set of attributes, e.g., 100. This makes any analysis more compact, as we can place the results in tables, rather than in graphs. Furthermore, because the setup is only performed once, they do not matter much in the overall efficiency comparison. Therefore, we do not consider those in the following comparisons.

Table 6: The analysis of the memory footprint for all implemented schemes and optimization approaches (OA). Table 6a lists the general storage costs of the master public key MPK, the master secret key MSK, the secret key SK, the set $\mathcal{S}$, and the ciphertext CT, where |att| denotes the length of a single attribute (in our implementations: att $\in \mathbb{Z}_p$), $\mathbb{A}$ denotes the policy length, $|\mathcal{S}|$ denotes the set size, and $\ell_{\mathbb{G}'}$ denotes the length of a single element in group $\mathbb{G}'$. Based on this and the rest of our analysis, Table 6b provides upper bounds on the memory consumption for the BLS12-381 and BN446 curves, where we distinguish the regular costs (R) from the precomputation tables (P) and the total costs (T).

Scheme	|MPK|				|MSK|		|SK|			|CT|			
	$\ell_{\mathbb{G}}$	$\ell_{\mathbb{H}}$	$\ell_{\mathbb{G}_T}$	|att|	$\ell_{\mathbb{Z}_p}$	|att|	$\ell_{\mathbb{G}}$	$\ell_{\mathbb{H}}$	|att|	$\ell_{\mathbb{G}}$	$\ell_{\mathbb{H}}$	$\ell_{\mathbb{G}_T}$	|att|
Wat11-I-OE	$\|\mathcal{U}\|+2$	1	1	$\|\mathcal{U}\|$	$\|\mathcal{U}\|+2$	$\|\mathcal{U}\|$	-	$\|\mathcal{S}\|+1$	$\|\mathcal{S}\|$	$2\|\mathbb{A}\|+1$	-	1	$2\|\mathbb{A}\|$
Wat11-I-OK	1	$\|\mathcal{U}\|+2$	1	$\|\mathcal{U}\|$	$\|\mathcal{U}\|+2$	$\|\mathcal{U}\|$	$\|\mathcal{S}\|+1$	-	$\|\mathcal{S}\|$	-	$2\|\mathbb{A}\|+1$	1	$2\|\mathbb{A}\|$
Wat11-IV-OE	2	1	1	-	2	-	$\|\mathcal{S}\|$	$\|\mathcal{S}\|+1$	$\|\mathcal{S}\|$	$\|\mathbb{A}\|+1$	$\|\mathbb{A}\|$	1	$2\|\mathbb{A}\|$
Wat11-IV-OK	2	1	1	-	2	-	$\|\mathcal{S}\|+1$	$\|\mathcal{S}\|$	$\|\mathcal{S}\|$	$\|\mathbb{A}\|$	$\|\mathbb{A}\|+1$	1	$2\|\mathbb{A}\|$
RW13-OE	5	1	1	-	5	-	-	$2\|\mathcal{S}\|+2$	$2\|\mathcal{S}\|$	$3\|\mathbb{A}\|+1$	-	1	$3\|\mathbb{A}\|$
RW13-OK	1	5	1	-	5	-	$2\|\mathcal{S}\|+2$	-	$2\|\mathcal{S}\|$	-	$3\|\mathbb{A}\|+1$	1	$3\|\mathbb{A}\|$
AC17-OE	$\|\mathcal{U}\|+2$	1	1	$\|\mathcal{U}\|$	$\|\mathcal{U}\|+2$	$\|\mathcal{U}\|$	-	$\|\mathcal{S}\|+2$	$\|\mathcal{S}\|$	$\|\mathbb{A}\|+2$	-	1	$2\|\mathbb{A}\|$
AC17-OK	1	$\|\mathcal{U}\|+2$	1	$\|\mathcal{U}\|$	$\|\mathcal{U}\|+2$	$\|\mathcal{U}\|$	$\|\mathcal{S}\|+2$	-	$\|\mathcal{S}\|$	-	$\|\mathbb{A}\|+2$	1	$2\|\mathbb{A}\|$
AC17-OD	$\|\mathcal{U}\|+2$	1	1	$\|\mathcal{U}\|$	$\|\mathcal{U}\|+2$	$\|\mathcal{U}\|$	$\|\mathcal{S}\|+1$	2	$\|\mathcal{S}\|$	$\|\mathbb{A}\|+1$	1	1	$2\|\mathbb{A}\|$
AC17-LU-OE	2	1	1	-	2	-	$\|\mathcal{S}\|$	2	$\|\mathcal{S}\|$	$\|\mathbb{A}\|+1$	1	1	$2\|\mathbb{A}\|$
AC17-LU-OK	2	1	1	-	2	-	$\|\mathcal{S}\|+1$	1	$\|\mathcal{S}\|$	$\|\mathbb{A}\|$	2	1	$2\|\mathbb{A}\|$

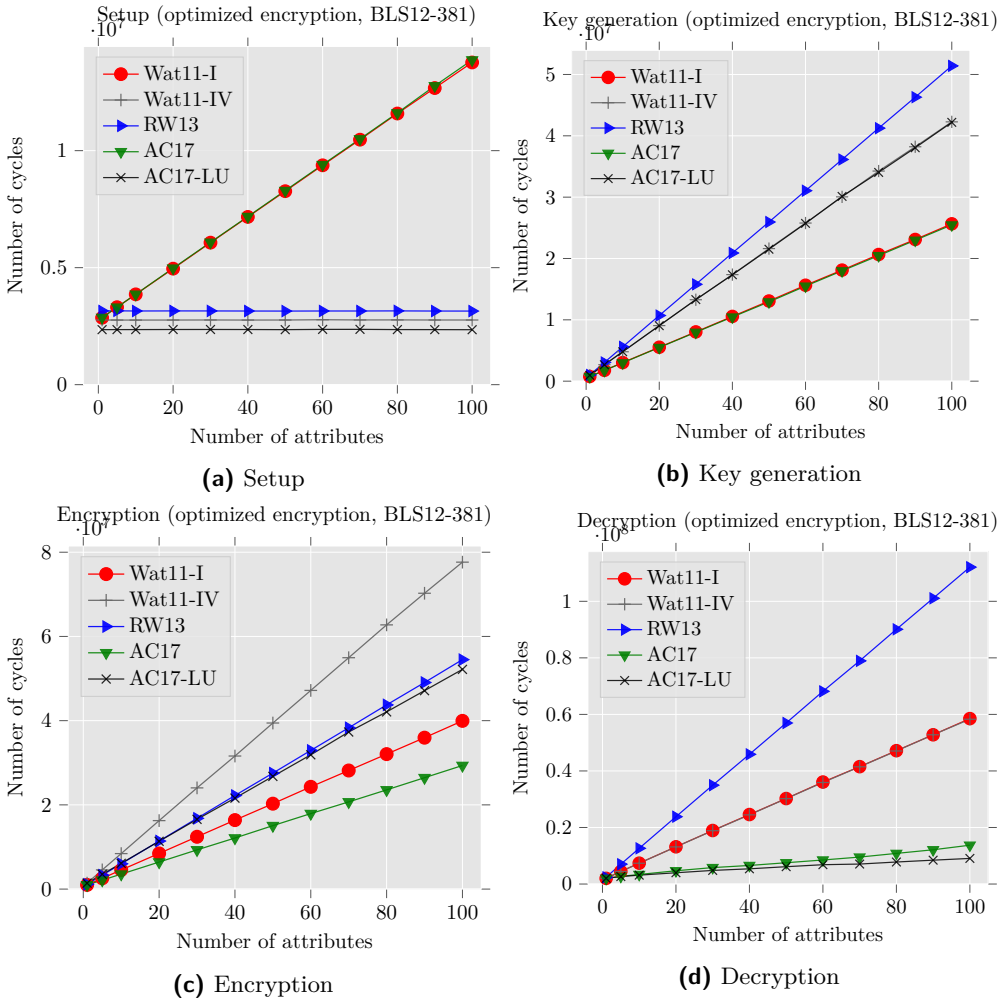
(a) General description of the regular storage costs

Scheme	OA	Curve	Setup			Key generation			Encryption			Decryption		
			R	P	T	R	P	T	R	P	T	R	P	T
Wat11-I	OE	BLS12-381	23	4	28	51	3	54	56	2	58	84	0	84
		BN446	32	5	37	67	4	71	71	2	72	106	0	106
	OK	BLS12-381	33	4	37	51	2	52	84	3	87	102	0	102
		BN446	43	5	48	67	2	69	103	4	107	128	0	128
Wat11-IV	OE	BLS12-381	1	4	6	19	3	22	52	4	56	70	0	70
		BN446	1	5	6	26	4	29	64	5	69	89	0	89
	OK	BLS12-381	1	4	6	19	4	24	52	3	55	70	0	70
		BN446	1	5	6	26	5	31	64	4	68	89	0	89
RW13	OE	BLS12-381	1	11	12	52	3	55	57	8	64	107	0	107
		BN446	2	12	14	66	4	69	71	9	80	135	0	135
	OE-L	BLS12-381	1	11	12	52	3	55	476	8	483	526	0	526
		BN446	2	12	14	66	4	69	723	9	732	787	0	787
	OK	BLS12-381	2	16	18	34	15	49	85	2	86	117	0	117
		BN446	2	19	21	44	18	62	104	2	106	146	0	146
AC17	OE	BLS12-381	23	4	28	51	3	54	47	2	48	75	0	75
		BN446	32	5	37	68	4	71	60	2	61	95	0	95
	OK	BLS12-381	33	4	37	51	2	52	66	3	69	84	0	84
		BN446	43	5	48	67	2	69	82	4	85	106	0	106
	OD	BLS12-381	23	4	28	42	3	45	47	4	52	65	0	65
		BN446	32	5	37	57	4	60	60	5	65	85	0	85
AC17-LU	OE	BLS12-381	1	4	6	19	4	24	33	3	36	52	0	52
		BN446	1	5	6	26	5	31	42	4	46	67	0	67
	OK	BLS12-381	1	4	6	19	4	24	33	3	36	52	0	52
		BN446	1	5	6	26	5	31	42	4	46	67	0	67

(b) Upper bounds on the memory footprint in KiB (= 1024 bytes) for 100 attributes

4.5.2 Comparing our framework with Charm and OpenABE

One of the main goals of our framework is to fully optimize the efficiency of all the schemes with respect to the same design goal. To show how effective this is compared to existing works, we have run benchmarks of the Charm [AGM+13] implementations of Wat11-I and RW13 as well as the implementations of our optimizations, i.e., Wat11-I-OE,



(a) Setup **(b)** Key generation **(c)** Encryption **(d)** Decryption

Figure 3: The computational costs in cycles for all the schemes on the BLS12-381 curve, where the schemes are optimized with respect to the encryption efficiency.

Wat11-I-OK, RW13-OE, and RW13-OK (Section 4.2). We have also run benchmarks of the OpenABE [Zeu20] implementation of Wat11-IV as well as the implementations of our two optimizations, i.e., Wat11-IV-OE and Wat11-IV-OK (Section 4.2). Because the implementations in Charm and OpenABE support, at best, the BN254 curve [BN05], we compare the computational costs of the schemes on this curve. The results in Table 7 show that our implementations greatly improve on the implementations of Charm, the costs being at least one order of magnitude lower. For decryption, our implementations perform even a factor 100-300 faster than the Charm implementation. This is a huge and important speed-up that can be noticed in practice. For instance, Table 7b shows that Charm takes several seconds to execute decryption for large policies with RW13, which increases even further if more up-to-date curves such as BLS12-381 are used. In contrast, our implementations never require more than 15 *milliseconds* to execute. Compared to the OpenABE implementation of Wat11-IV, our implementations perform roughly equally efficient in the key generation, a factor 1.6 faster in the encryption algorithm, and a factor 4 faster in the decryption algorithm.

In addition, Table 7 illustrates that comparing optimized implementations of the schemes yields a different comparison of two schemes. For instance, the Charm implementations

Table 7: Comparison of the computational costs of the Charm and OpenABE implementations of Wat11 and RW13, and our implementations, on the BN254 curve. For the costs with 100 attributes, we also provide the factor (×) by which the costs are lower than the least efficient variant of the scheme. For each scheme, the lowest costs are typeset in **bold**. Table 7b expresses the costs in milliseconds (for a processor with a frequency of 4.1 GHz).

Scheme/ Variant	Key generation				Encryption				Decryption			
	# of attributes				# of attributes				# of attributes			
	1	10	100	×	1	10	100	×	1	10	100	×
Wat11-I												
Charm	5956	20930	169273	-	18810	79498	683225	-	112564	465581	4142503	-
OE	347	1369	11618	15	**495**	**2395**	**21529**	32	**753**	**2357**	**20748**	200
OK	**166**	**646**	**5415**	31	825	5133	47940	14	760	2649	24682	168
RW13												
Charm	9793	48375	430168	-	20420	89673	775595	-	165344	1361208	13347677	-
OE	464	2559	23621	18	**566**	**3113**	**28740**	27	**907**	**4101**	**39651**	337
OK	**223**	**1236**	**11353**	38	934	6260	59413	13	922	4468	45220	295
Wat11-IV												
OpenABE	806	2486	19265	-	1129	6137	57261	-	3002	10668	94426	-
OE	417	1782	**16015**	1.2	**645**	**3612**	**33909**	1.7	**746**	**2339**	**20517**	4.6
OK	**365**	**1774**	16337	1.2	727	3760	34612	1.7	762	2401	21116	4.5

(a) Costs in the number of 10^3 clock cycles

Scheme/ Variant	Key generation			Encryption			Decryption		
	# of attributes			# of attributes			# of attributes		
	1	10	100	1	10	100	1	10	100
Wat11-I									
Charm	1.5	5.1	41.3	4.6	19.4	166.6	27.5	113.6	1010.4
OE	0.1	0.3	2.8	0.1	0.6	5.3	0.2	0.6	5.1
OK	0.0	0.2	1.3	0.2	1.3	11.7	0.2	0.6	6.0
RW13									
Charm	2.4	11.8	104.9	5.0	21.9	189.2	40.3	332.0	3255.5
OE	0.1	0.6	5.8	0.1	0.8	7.0	0.2	1.0	9.7
OK	0.1	0.3	2.8	0.2	1.5	14.5	0.2	1.1	11.0
Wat11-IV									
OpenABE	0.2	0.6	4.7	0.3	1.5	14.0	0.7	2.6	23.0
OE	0.1	0.4	3.9	0.2	0.9	8.3	0.2	0.6	5.0
OK	0.1	0.4	4.0	0.2	0.9	8.4	0.2	0.6	5.2
Wat11-IV									
OpenABE	0.2	0.6	4.7	0.3	1.5	14.0	0.7	2.6	23.0
OE	0.1	0.4	4.0	0.2	0.9	8.3	0.2	0.6	5.1
OK	0.1	0.4	3.9	0.2	0.9	8.3	0.2	0.6	5.0

(b) Costs in milliseconds

of Wat11-I and RW13 show that Wat11-I outperforms RW13 in all algorithms: its key generation is 154% faster, its encryption is 13% faster and its decryption is 222% faster. In contrast, our implementations of the same schemes, optimized with respect to the encryption algorithm compare differently. These implementations also illustrate that Wat11-I outperforms RW13, but the differences in costs are distinct: key generation is only 103% faster, encryption is even 33% faster, and decryption is only 91% faster. This means that, in contrast to what the Charm implementations suggest, Wat11-I encryption is actually even faster than RW13, and RW13 decryption does not perform as badly compared to Wat11-I. However, note that the schemes also have different properties, the most notable difference being that Wat11-I is a small-universe construction, while RW13 is a large-universe construction. We show in Section 4.7 that Wat11-IV—the large-universe variant of Wat11-I—is actually slower than Wat11-I, and is even outperformed by RW13 in the key generation and encryption algorithms for the OK and OE approaches, respectively. This difference in results illustrates that it is important to compare two implementations of schemes with the same properties, which are subsequently optimized with respect to

Table 8: The computational costs of several implementations of Wat11-I on the BLS12-381 curve, based on their optimization approaches (OA). The costs are expressed in 10^3 clock cycles. For 100 attributes, we determine the most efficient variant (typeset in **bold**), and the increase (in percentages) that the other variants incur compared to this variant.

OA	Key generation				Encryption				Decryption			
	# of attributes				# of attributes				# of attributes			
	1	10	100	Increase	1	10	100	Increase	1	10	100	Increase
OE	759	3029	25653	143.0%	**990**	**4540**	**39951**	-	**2005**	**7379**	**58515**	-
OK	**317**	**1249**	**10555**	-	1756	10814	101181	153.3%	2016	7611	63151	7.9%
BAD-I	841	3776	33834	220.5%	1536	9285	88477	121.5%	3760	29841	296748	407.1%
BAD-II	605	2852	25408	140.7%	1454	9186	86535	116.6%	2005	7649	63166	7.9%

the same goals. If this is not done, then one might unjustifiably draw the conclusion that Wat11-IV is a more efficient scheme (given any design goal) than RW13.

4.5.3 Comparing Wat11-I optimized approaches with bad approaches

We also compare the computational costs of our optimizations of Wat11-I with the badly optimized variants Wat11-I-BAD-I (Section 4.2.5), and Wat11-I-BAD-II (Section 4.2.5). By investigating the difference between Wat11-I-BAD-II and our optimizations, we can analyze the advantage of strategically placing the key and ciphertext components in the two source groups in the type conversion. By comparing Wat11-I-BAD-I and Wat11-I-BAD-II, we can determine the advantage of using all optimized arithmetic and group operations. Table 8 shows that, indeed, our optimizations are generally better than Wat11-I-BAD-II. That is, the key generation and decryption algorithms of Wat11-I-OE variant perform comparably to Wat11-I-BAD-II, while its encryption is much faster (i.e., by 116%). Furthermore, encryption of Wat11-I-OK is slightly slower (i.e., by 14%) than Wat11-I-BAD-II, but its key generation is faster by 140%. In addition, the table shows that the use of optimized arithmetic matters much, as the computational costs of Wat11-I-BAD-I increase compared to Wat11-I-BAD-II: key generation by 33%, encryption by 2% and decryption by 369%.

4.6 Comparing different optimizations

For each of the five schemes, we list the computational costs on the curves: two curves in the $[125, 128]$-bit security range, BLS12-381 and BN382, and two curves in the $[129, 135]$-bit security range, BLS12-446 and BN446. By doing this, we illustrate that the efficiency of the scheme depends heavily on the chosen optimization approaches. Furthermore, this allows us to determine the best choice of curve, depending on the optimization approach. These should thus be clearly specified and explained in any benchmarking efforts.

4.6.1 Comparing the schemes on different curves

First, we find the best curves for each scheme and optimization approach. In Table 9, we compare the efficiency for each optimization of each scheme on two curves: the BLS12-381 and BN382 curves, which are two curves with roughly the same security level. Table 9 shows that there is no universally best choice when it comes to the efficiency of the schemes. In particular, for the curves in the $[125, 128]$-bit security range, BLS12-381 outperforms BN382 in all algorithms for the Wat11 and RW13 schemes. For the AC17 schemes, decryption is faster on BN382 (for more than 1 attribute), and for key generation and encryption, BLS12-381 is faster. For the curves in the $[129, 135]$-bit security range, BLS12-446 outperforms the BN446 curve in almost all algorithms for the Wat11 and RW13 schemes, except the key generation of Wat11-IV for 100 attributes. In the full version [dlPVA22], we summarize the best curve choices for the optimizations of AC17. In general,

Table 9: Comparison of the computational costs for each scheme (Sch) and optimization approach (OA) on four different curves: two curves in the [125, 128]-bit security range, BLS12-381 and BN382, and two curves in the [129, 135]-bit security range, BLS12-446 and BN446. The costs are expressed in 10^3 clock cycles. For each pair of curves, the lowest costs are typeset in **bold**.

Sch	OA	Curve	Key generation			Encryption			Decryption		
			# of attributes			# of attributes			# of attributes		
			1	10	100	1	10	100	1	10	100
Wat11-I	OE	BLS12-381	**759**	**3029**	**25653**	**990**	**4540**	**39951**	**2005**	**7379**	**58515**
		BN382	1132	4495	38050	1473	6585	57492	2546	7851	64934
		BLS12-446	**1142**	**4538**	**38454**	**1466**	**6714**	**58789**	**2980**	**10586**	**84291**
		BN446	1732	6933	58456	2286	9867	85086	3975	12259	99766
	OK	BLS12-381	**317**	**1249**	**10555**	**1756**	**10814**	**101181**	**2016**	**7611**	**63151**
		BN382	469	1859	15762	2625	16240	152142	2536	8688	72243
		BLS12-446	**481**	**1904**	**16166**	**2607**	**16209**	**152224**	**2955**	**10973**	**91919**
		BN446	705	2806	23856	4090	24739	231963	3974	12948	105355
Wat11-IV	OE	BLS12-381	**959**	**4798**	**42275**	**1413**	**8429**	**77641**	**2013**	**7356**	**58290**
		BN382	1118	5141	44759	1779	10428	96217	2523	7807	64806
		BLS12-446	**1286**	**6972**	64972	**1934**	**12321**	**117130**	**2966**	**10605**	**84761**
		BN446	1805	7695	**63968**	2869	15691	141311	3980	12209	99993
	OK	BLS12-381	**812**	**4632**	**42135**	**1553**	**8568**	**77898**	**2015**	**7376**	**58441**
		BN382	923	5055	45712	2046	10941	99011	2585	7991	66184
		BLS12-446	**1073**	**6794**	65766	**2151**	**12602**	**119816**	**2960**	**10583**	**86263**
		BN446	1491	7459	**64486**	3282	16307	144159	4074	12469	101924
RW13	OE	BLS12-381	**1018**	**5602**	**51401**	**1143**	**6002**	**54491**	**2396**	**12630**	**112072**
		BN382	1506	8290	76109	1707	8903	80680	3120	13794	127772
		BLS12-446	**1520**	**8376**	**76853**	**1697**	**9045**	**82362**	**3535**	**18172**	**162064**
		BN446	2318	12732	117068	2648	13447	121476	4871	21372	195891
	OK	BLS12-381	**429**	**2347**	**21657**	**2026**	**13469**	**128221**	**2398**	**12828**	**118998**
		BN382	633	3487	31998	3005	20062	190792	3123	14675	137410
		BLS12-446	**650**	**3576**	**32660**	**3015**	**20242**	**191950**	**3550**	**18694**	**173177**
		BN446	951	5228	47748	4688	30858	291444	4883	22184	203171
AC17	OE	BLS12-381	**753**	**2986**	**25471**	**998**	**3550**	**29348**	**2021**	**3383**	13736
		BN382	1132	4498	38262	1474	5146	41974	2555	3517	**11603**
		BLS12-446	**1140**	**4543**	**38615**	**1466**	**5234**	**42860**	**2985**	**4520**	18294
		BN446	1740	6985	58691	2288	7684	61472	4013	4897	**11749**
	OK	BLS12-381	**318**	**1253**	**10635**	**1752**	**8494**	**76067**	**2016**	**3375**	13696
		BN382	470	1866	15776	2609	12778	114130	2544	3509	**11454**
		BLS12-446	**485**	**1914**	**16103**	**2615**	**12772**	**114055**	**2979**	**4510**	18258
		BN446	724	2877	24456	4085	19579	174546	3988	4880	**11721**
	OD	BLS12-381	**612**	**1547**	**10904**	**1144**	**3688**	**29317**	**2017**	3154	9073
		BN382	907	2301	16184	1680	5348	41776	2530	**2632**	**4033**
		BLS12-446	**922**	**2342**	**16653**	**1682**	**5410**	**42957**	**2971**	**4086**	10947
		BN446	1402	3508	24607	2677	8034	62200	3998	4108	**5662**
AC17-LU	OE	BLS12-381	**958**	**4796**	**42196**	**1408**	**6115**	**52176**	**2022**	3142	9060
		BN382	1118	5151	45093	1776	7054	59276	2538	**2639**	**4058**
		BLS12-446	**1275**	**6983**	64930	**1910**	**8863**	**79350**	**2942**	**4092**	11033
		BN446	1763	7656	**63891**	2865	10458	84030	3984	4099	**5676**
	OK	BLS12-381	**797**	**4607**	**41913**	**1556**	**6262**	**52326**	**2020**	3143	9076
		BN382	890	4901	44523	2001	7289	59307	2538	**2637**	**4056**
		BLS12-446	**1022**	**6778**	65029	**2176**	**9086**	**79552**	**2967**	**4100**	11109
		BN446	1441	7343	**64017**	3204	10854	84034	3995	4121	**5704**

decryption is the most efficient on BN446, while the other algorithms are more efficient on BLS12-446 (with the exception of key generation of AC17-LU for 100 attributes).

4.6.2 Comparing schemes for different optimizations

In Section 3.5, we explained that the chosen design goal influences the optimization approach, including the conversion strategy from the type-I to the type-III setting. To this end, we have converted each scheme with respect to the different design goals. We illustrate the trade-offs incurred by the conversions in Table 10. It shows that, indeed, the variant of a scheme that is optimized with respect to a specific algorithm also outperforms the other variants in this algorithm. Note that, as expected, for the schemes without an FDH, these differences are much more pronounced than the schemes with an FDH.

4.6.3 Comparing access trees with LSSS matrices

We also analyze the computational costs of RW13 for two variants of the scheme: one using the access trees like in the rest of this work, and one using more efficient LSSS matrices as mentioned in Section 2.2 (see the full version [dlPVA22] for a description). These LSSS matrices were also used in the Charm implementation of FAME [AC17a]. However, they do not compare it with the use of access trees used in OpenABE and our other implementations. Therefore, we investigate the computational advantage of using LSSS matrices. In particular, our comparison suggests that the choice of access structure does matter in the performance analysis as well. As Table 11 shows, the use of LSSS matrices barely has an effect on the key generation and encryption efficiency. Nevertheless, the decryption costs—which are typically the highest—can be decreased considerably by using LSSS matrices. That is, the decryption costs of RW13 with access trees is up to 45% more costly than of RW13 with LSSS matrices. Hence, we recommend that LSSS matrices are used in practice.

4.7 Proof of concept: comparison of different schemes

We also show how the benchmarks can be used in the comparison of different schemes. We do this by comparing the computational costs of the schemes for the same optimization approaches. Specifically, we compare the large-universe schemes, Wat11-IV, AC17-LU and RW13, with one another to investigate which of the three performs best with respect to some chosen optimization approach. We also compare the large-universe variants Wat11-IV and AC17-LU with their small-universe counterparts, Wat11-I and AC17, to investigate the sacrifice in efficiency that the large-universe property requires.

4.7.1 Comparing the large-universe schemes

In Table 12, we compare the computational costs of the large-universe schemes, i.e., Wat11-LU, RW13 and AC17-LU. It shows that AC17-LU outperforms the other two in almost all optimizations and the subsequent implementations of the algorithms. The only exception is the optimized key generation approach, where RW13 provides the most efficient key generation algorithm, outperforming the other two schemes by a factor 2. It therefore seems that, currently, RW13 is the best choice when the design goal is to have an optimized key generation algorithm. For the other approaches, it is best to use AC17-LU. Notably, RW13 outperforms Wat11-IV in the OE, OK and BKE approaches, and thus constitutes not only a scheme that is interesting for its theoretical properties, but also for its efficiency.

Table 10: Comparison of the computational costs expressed in 10^3 clock cycles for each scheme and optimization approach for 100 attributes, on their most efficient curve in the [125, 128] and the [129, 135]-bit security ranges, respectively. For each scheme, we determine the most efficient variant, and the increase (in percentages) that the other variants incur compared to the most efficient variant. The lowest costs are typeset in **bold**.

Scheme	OA	Curve	Key generation		Encryption		Decryption	
			Costs	Increase	Costs	Increase	Costs	Increase
Wat11-I	OE	BLS12-381	25653	143.0%	**39951**	-	**58515**	-
	OK	BLS12-381	**10555**	-	101181	153.3%	63151	7.9%
Wat11-IV	OE	BLS12-381	42275	0.3%	**77641**	-	**58290**	-
	OK	BLS12-381	**42135**	-	77898	0.3%	58441	0.3%
RW13	OE	BLS12-381	51401	137.3%	**54491**	-	**112072**	-
	OK	BLS12-381	**21657**	-	128221	135.3%	118998	6.2%
AC17	CP	BLS12-381	10696	0.6%	29352	0.0%	9027	123.8%
	OE	BLS12-381	25471	139.5%	**29348**	-	13736	240.6%
	OK	BLS12-381	**10635**	-	76067	159.2%	13696	239.6%
	OD	BN382	16184	52.2%	41776	42.3%	**4033**	-
AC17-LU	CP	BLS12-381	42632	1.7%	52752	1.1%	9106	124.4%
	OE	BLS12-381	42196	0.7%	**52176**	-	9060	123.3%
	OK	BLS12-381	**41913**	-	52326	0.3%	9076	123.7%
	OD	BN382	45093	7.6%	59276	13.6%	**4058**	-

(a) BLS12-381 and BN382 curves

Scheme	OA	Curve	Key generation		Encryption		Decryption	
			Costs	Increase	Costs	Increase	Costs	Increase
Wat11-I	OE	BLS12-446	38454	137.9%	**58789**	-	**84291**	-
	OK	BLS12-446	**16166**	-	152224	158.9%	91919	9.0%
Wat11-IV	OE	BLS12-446	64972	0.8%	**117130**	-	**84761**	-
	OK	BN446	**64486**	-	144159	23.1%	101924	20.2%
RW13	OE	BLS12-446	76853	135.3%	**82362**	-	**162064**	-
	OK	BLS12-446	**32660**	-	191950	133.1%	173177	6.9%
AC17	CP	BLS12-446	16339	1.5%	43040	0.4%	10920	92.9%
	OE	BLS12-446	38615	139.8%	**42860**	-	18294	223.1%
	OK	BLS12-446	**16103**	-	114055	166.1%	18258	222.5%
	OD	BN446	24607	52.8%	62200	45.1%	**5662**	-
AC17-LU	CP	BLS12-446	64677	1.2%	79735	0.5%	11075	95.1%
	OE	BLS12-446	64930	1.6%	**79350**	-	11033	94.4%
	OK	BN446	64017	0.2%	84034	5.9%	5704	0.5%
	OD	BN446	**63891**	-	84030	5.9%	**5676**	-

(b) BLS12-446 and BN446 curves

4.7.2 Comparing small-universe schemes with their large-universe variant

In Table 13, we compare the large-universe variants of Wat11 and AC17 with their small-universe counterparts. This illustrates the sacrifice in efficiency incurred by the FDH that is instantiated to achieve the large-universe property. Concretely, the table shows that, for each optimization approach, the optimized algorithms of the small-universe variant outperform the large-universe variant. For the optimized key generation and encryption approaches, the small-universe variants perform overwhelmingly better: at least a factor 1.7, and at most a factor 4. For the optimized decryption approaches, the decryption costs are similar for the small-universe and large-universe variants. The reason why the trade-off in efficiency is so high is the FDH: not only does it limit us in the conversion from the

Table 11: Comparison of the computational costs of RW13-OE with the access trees of OpenABE, and with more efficient LSSS matrices, on the BLS12-381 curve. The costs are expressed in 10^3 clock cycles. For each number (#) of attributes, we also provide the increase (in percentage %) by which the costs of the variant are higher than the most efficient variant (which costs are typeset in **bold**).

Key generation				**Encryption**				**Decryption**			
#	Tree	LSSS	%	#	Tree	LSSS	%	#	Tree	LSSS	%
1	1018	**997**	2.1	1	1143	**1121**	2.0	1	2396	**2357**	1.7
5	3056	**3003**	1.8	5	3302	**3220**	2.6	5	7018	**5364**	30.8
10	5602	**5517**	1.5	10	6002	**5864**	2.3	10	12630	**9149**	38.0
20	10703	**10549**	1.5	20	11411	**11211**	1.8	20	23815	**16654**	43.0
30	15788	**15639**	1.0	30	16806	**16601**	1.2	30	34954	**24194**	44.5
40	20894	**20663**	1.1	40	22229	**22016**	1.0	40	45879	**31771**	44.4
50	25946	**25711**	0.9	50	27546	**27443**	0.4	50	56961	**39274**	45.0
60	31053	**30842**	0.7	60	32933	**32902**	0.1	60	68160	**47019**	45.0
70	36139	**35976**	0.5	70	**38307**	38682	1.0	70	78937	**54694**	44.3
80	41245	**41028**	0.5	80	**43745**	44165	1.0	80	90095	**62212**	44.8
90	46295	**46051**	0.5	90	**49060**	49880	1.7	90	101063	**69861**	44.7
100	51401	**51108**	0.6	100	**54491**	55545	1.9	100	112072	**77432**	44.7

type-I to the type-III setting, but we also need to perform a hash operation and use a variable-base exponentiation. This is also why RW13 outperforms Wat11-IV in Table 12.

5 Future work

This work provides the basis for further research at various levels.

5.1 Automating our framework

Our type-conversion methods are heuristic and manual. The reason why they are heuristic is because the conversion methods are inextricably intertwined with the efficiency of the arithmetic not only provided by the chosen curve, but also by the order of the computations and the implementation of the arithmetic (which might in turn depend on the architecture of the processor). Currently, automating our conversion techniques is not trivial. Due to the heuristic nature of our given methods (which requires us to circle back to earlier design choices to see if these need to be adjusted), it may be more difficult to automate than like in [AGOT14, AGH15, AHO16]. Instead, one could take a different approach. First, one could make a theoretical estimation of the computational costs of the arithmetic and group operations, for all possible pairing-friendly curves at the desired security level. Second, one could make a list of all possible distributions of the key and ciphertext components over the source groups. For each distribution and pairing-friendly curve, one can then determine the most efficient algorithms for the group operations and optimize the order of computations (which is not a trivial effort either). Given some optimization goal, the most efficient distribution can then be selected to be the optimal type conversion.

5.2 More pairing-friendly curves

In our analysis, we have only considered two curves in the $[125, 128]$-bit security range and two curves in the $[129, 135]$-bit security range. As we mentioned in the introduction, many pairing-friendly groups exist that provide at least 128 bits of security [Gui20a]. Notably, the KSS16 curves [KSS08] provide efficient arithmetic in the first source group and efficient multi-pairing operations [GF16, Ara17, CDS20]. These might be especially beneficial for

Table 12: Comparison of the computational costs for each large-universe scheme and optimization approach (OA) for 100 attributes, on their most efficient curves in the [125, 128]-bit and the [129, 135]-bit security ranges, respectively. For each optimization approach, we determine the most efficient scheme, and the increase (in percentages) that the other schemes incur compared to the most efficient variant (which costs are typeset in **bold**). The costs are expressed in 10^3 clock cycles.

OA	Scheme	Curve	Key generation		Encryption		Decryption	
			Costs	Increase %	Costs	Increase %	Costs	Increase %
OE	Wat11-IV	BLS12-381	42275	0.2%	77641	48.8%	58290	543.4%
	RW13	BLS12-381	51401	21.8%	54491	4.4%	112072	1137.1%
	AC17-LU	BLS12-381	**42196**	-	**52176**	-	**9060**	-
OK	Wat11-IV	BLS12-381	42135	94.6%	77898	48.9%	58441	543.9%
	RW13	BLS12-381	**21657**	-	128221	145.0%	118998	1211.2%
	AC17-LU	BLS12-381	41913	93.5%	**52326**	-	**9076**	-
OD	Wat11-IV	BLS12-381	**42275**	-	77641	42.5%	58290	1336.5%
	RW13	BLS12-381	51401	21.6%	**54491**	-	112072	2661.9%
	AC17-LU	BN382	45093	6.7%	59276	8.8%	**4058**	-

(a) BLS12-381 and BN382 curves

OA	Scheme	Curve	Key generation		Encryption		Decryption	
			Costs	Increase %	Costs	Increase %	Costs	Increase %
OE	Wat11-IV	BLS12-446	64972	0.1%	117130	47.6%	84761	668.2%
	RW13	BLS12-446	76853	18.4%	82362	3.8%	162064	1368.8%
	AC17-LU	BLS12-446	**64930**	-	**79350**	-	**11033**	-
OK	Wat11-IV	BN446	64486	97.4%	144159	81.2%	101924	817.5%
	RW13	BLS12-446	**32660**	-	191950	141.3%	173177	1458.9%
	AC17-LU	BLS12-446	65029	99.1%	**79552**	-	**11109**	-
OD	Wat11-IV	BLS12-446	64972	1.7%	117130	42.2%	84761	1393.4%
	RW13	BLS12-446	76853	20.3%	**82362**	-	162064	2755.4%
	AC17-LU	BN446	**63891**	-	84030	2.0%	**5676**	-

(b) BLS12-446 and BN446 curves

schemes such as RW13, which provide much freedom with respect to their type conversion. In order to improve the benchmarks in this framework, more curves need to be supported by RELIC. Alternatively, a framework or library can be set up with the estimated efficiency of frequently-used arithmetic of the curves providing 128 bits of security listed at [Gui20a]. This would also help in any automated efforts.

5.3 Improving usability, validity and verifiability

To simplify the accurate comparison of schemes even further, it is important to make the framework more usable for ABE designers. As a result, cryptographers can easily compare their new scheme with existing ones in a transparent way without requiring a deep understanding of cryptographic engineering. One could make the framework more usable by providing a functionality that allows designers to specify e.g., an encoding of a scheme (rather than a full-fledged description). In this way, it also becomes easier to analyze the schemes with respect to other metrics, such as validity and verifiability, e.g., either manually [VA21] or even automatically [ABGW17].

5.4 Implementing fully secure ABE

We have implemented only the selectively secure variants of Wat11, RW13 and AC17. While this provides a reliable comparison of the structure of the schemes, in practice, we

Table 13: Comparison of the computational costs of the small-universe with the large-universe variants of the Wat11 and AC17 schemes for 100 attributes, for each optimization approach, on their most efficient curve in the $[125, 128]$-bit and $[129, 135]$-bit security ranges, respectively. The costs are expressed in 10^3 clock cycles. We also provide the increase (in percentage) by which the costs of the variant are higher than the most efficient variant of the scheme (which costs are typeset in **bold**).

Scheme	OA	SU/ LU	Curve	Key generation		Encryption		Decryption	
				Costs	Increase	Costs	Increase	Costs	Increase
Wat11	OE	SU	BLS12-381	**25653**	-	**39951**	-	58515	0.4%
		LU	BLS12-381	42275	64.8%	77641	94.3%	**58290**	-
	OK	SU	BLS12-381	**10555**	-	101181	29.9%	63151	8.1%
		LU	BLS12-381	42135	299.2%	**77898**	-	**58441**	-
AC17	OE	SU	BLS12-381	**25471**	-	**29348**	-	13736	51.6%
		LU	BLS12-381	42196	65.7%	52176	77.8%	**9060**	-
	OK	SU	BLS12-381	**10635**	-	76067	45.4%	13696	50.9%
		LU	BLS12-381	41913	294.1%	**52326**	-	**9076**	-
	OD	SU	BN382	**16184**	-	**41776**	-	**4033**	-
		LU	BN382	45093	178.6%	59276	41.9%	4058	0.6%

(a) BLS12-381 and BN382 curves

Scheme	OA	SU/ LU	Curve	Key generation		Encryption		Decryption	
				Costs	Increase	Costs	Increase	Costs	Increase
Wat11	OE	SU	BLS12-446	**38454**	-	**58789**	-	**84291**	-
		LU	BLS12-446	64972	69.0%	117130	99.2%	84761	0.6%
	OK	SU	BLS12-446	**16166**	-	152224	5.6%	**91919**	-
		LU	BN446	64486	298.9%	**144159**	-	101924	10.9%
AC17	OE	SU	BLS12-446	**38615**	-	**42860**	-	18294	65.8%
		LU	BLS12-446	64930	68.1%	79350	85.1%	**11033**	-
	OK	SU	BLS12-446	**16103**	-	114055	43.4%	18258	64.4%
		LU	BLS12-446	65029	303.8%	**79552**	-	**11109**	-
	OD	SU	BN446	**24607**	-	**62200**	-	**5662**	-
		LU	BN446	63891	159.6%	84030	35.1%	5676	0.2%

(b) BLS12-446 and BN446 curves

require the use of a fully secure scheme. Since several frameworks exist that provide efficient generic conversions to the full-security setting [Wee14, Att14], it would be important to benchmark the underlying groups used in such conversions. In this way, the most efficient security-conversion technique can be selected. Note that those generic conversions are also compatible with the aforementioned encodings (Section 5.3).

5.5 Using other platforms

We have implemented and benchmarked the chosen schemes on an x64-based platform with a single core and significant computational resources (e.g., with a large RAM and high clock frequency). Possibly, other platforms allow for faster implementations, while more resource-constrained devices may perform slower e.g., due to their lower clock frequencies, or simply cannot even store the scheme's parameters in memory. Additionally, using single instruction multiple data (SIMD) extensions has been shown to significantly speed up elliptic-curve and pairing-based cryptography, e.g., with NEON for ARM-based [BS12, SR13, SLG+14] platforms and AVX2 for x64-based platforms [FL15, FLD19, CGT+20]. Furthermore, using multiple cores to parallelize the implementations has yielded improvements as well [FSV07, FSV08, GGP08], and may be especially useful in the implementation of the key generation and encryption algorithms owing to their already parallelized nature.

5.6 Using other algorithms for group operations

We have used RELIC for the implementations of the group operations used in the ABE schemes. As mentioned in Section 3.1, RELIC does not support all available algorithms, e.g., it does not support fixed-argument pairings. Furthermore, using precomputation tables in multiple-base exponentiations may significantly speed up the encryption algorithm [Möl01]. Conversely, implementing ABE in resource-constrained devices may require the use of different optimizations [FA17].

5.7 Expanding to other pairing-based ABE, and related primitives

Our methods are mostly targeted at optimizing pairing-based ABE of the specific structure described in Section 2.5. While this covers many ABE schemes, some schemes exist that do not have this exact structure, e.g., [LW11a, RW15]. Furthermore, our methods are also applicable to other pairing-based primitives that satisfy the targeted structure [AC17b, Att19], e.g., identity-based encryption [Sha84, BF01] and identity-based broadcast encryption [Del07]. Possibly, our framework can be expanded to cover pairing-based cryptography for other structures (and primitives) as well.

6 Conclusion

We have presented ABE Squared, a framework for accurately benchmarking efficiency of attribute-based encryption. Concretely, this framework aims to optimize the theoretical descriptions of ABE schemes for some chosen design goal by considering four optimization layers. These layers consider the arithmetic and group operations, the chosen pairing-friendly group, the order of the computations and the conversion techniques. By taking into account all layers during the optimization of a theoretical description, we are able to attain more efficient implementations. More specifically, we have devised several optimization approaches that aim to accomplish some chosen design goal, e.g., optimized key generation, encryption or decryption. By optimizing multiple schemes with respect to the same goal, they can be compared more fairly. Because existing conversion techniques did not allow us to e.g., optimize the decryption algorithm, we have given new heuristic and manual techniques that facilitate this. Unlike other existing works, these conversion techniques take into account the other three optimization layers.

To show the effectiveness of our framework, we have optimized and implemented five schemes: Wat11-I, Wat11-IV, RW13, AC17 and AC17-LU. These implementations show that, indeed, the efficiency of the schemes depends heavily on the design goals and subsequent optimization approaches. For example, Charm shows that Wat11-I is generally faster than RW13. This may result in the idea that Wat11-IV, the large-universe variant of Wat11-I, is also faster than RW13, because it is similar. In contrast, we have shown that RW13 outperforms Wat11-IV with respect to the optimized encryption and optimized key generation approaches. This illustrates clearly that taking into account any such design goals in the implementation and benchmarks is crucial in the comparisons as well. Therefore, the ABE Squared framework provides an instrumental contribution in the benchmarking—and eventually, in the deployment—of ABE.

Acknowledgments

The authors would like to thank the reviewers for their helpful comments and suggestions. They would also like to thank Diego Aranha for his tremendous help with RELIC.

References

[ABGW17] M. Ambrona, G. Barthe, R. Gay, and H. Wee. Attribute-based encryption in the generic group model: Automated proofs and new constructions. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *CCS*, pages 647–664. ACM, 2017.

[ABN+21] S. Agrawal, R. Biswas, R. Nishimaki, K. Xagawa, X. Xie, and S. Yamada. Cryptanalysis of Boyen's attribute-based encryption scheme in TCC 2013. Cryptology ePrint Archive, Report 2021/505, 2021. https://eprint.iacr.org/2021/505.

[AC17a] S. Agrawal and M. Chase. FAME: fast attribute-based message encryption. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *CCS*, pages 665–682. ACM, 2017.

[AC17b] S. Agrawal and M. Chase. Simplifying design and analysis of complex predicate encryption schemes. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT*, volume 10210 of *Lecture Notes in Computer Science*, pages 627–656. Springer, 2017.

[AFK+12] D. F. Aranha, L. Fuentes-Castañeda, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In M. Abdalla and T. Lange, editors, *Pairing*, volume 7708 of *Lecture Notes in Computer Science*, pages 177–195. Springer, 2012.

[AGH13] J. A. Akinyele, M. Green, and S. Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *CCS*, pages 399–410. ACM, 2013.

[AGH15] J. A. Akinyele, C. Garman, and S. Hohenberger. Automating fast and secure translations from type-I to type-III pairing schemes. In I. Ray, N. Li, and C. Kruegel, editors, *CCS*, pages 1370–1381. ACM, 2015.

[AGM+] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[AGM+13] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptogr. Eng.*, 3(2):111–128, 2013.

[AGOT14] M. Abe, J. Groth, M. Ohkubo, and T. Tango. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In J. A. Garay and R. Gennaro, editors, *CRYPTO*, volume 8616 of *Lecture Notes in Computer Science*, pages 241–260. Springer, 2014.

[AHM+16] N. Attrapadung, G. Hanaoka, T. Matsumoto, T. Teruya, and S. Yamada. Attribute based encryption with direct efficiency tradeoff. In M. Manulis, A.-R. Sadeghi, and S. A. Schneider, editors, *ACNS*, volume 9696 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 2016.

[AHO16] M. Abe, F. Hoshino, and M. Ohkubo. Design in type-I, run in type-III: Fast and scalable bilinear-type conversion using integer programming. In M. Robshaw and J. Katz, editors, *CRYPTO*, volume 9816 of *Lecture Notes in Computer Science*, pages 387–415. Springer, 2016.

[Ara17] D. Aranha. Pairings are not dead, just resting, 2017.

[Att14] N. Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 557–577. Springer, 2014.

[Att16] N. Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT*, volume 10032 of *Lecture Notes in Computer Science*, pages 591–623. Springer, 2016.

[Att19] N. Attrapadung. Unbounded dynamic predicate compositions in attribute-based encryption. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT*, volume 11476 of *Lecture Notes in Computer Science*, pages 34–67. Springer, 2019.

[Bar20] E. Barker. Recommendation for key management–part 1: General (revision 5). 2020.

[BB04] D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[BBG05] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[BCCT12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In S. Goldwasser, editor, *ITCS*, pages 326–349. ACM, 2012.

[BD19] R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *J. Cryptol.*, 32(4):1298–1336, 2019.

[BF01] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[BKLS02] P. S. L. M. Barreto, H. Yong Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.

[BLS02] P S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.

[BN05] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. E. Tavares, editors, *SAC*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.

[Bow] S. Bowe. BLS12-381: New zk-SNARK elliptic curve construction. https://blog.z.cash/new-snark-curve/.

[Boy08] X. Boyen. The uber-assumption family – a unified complexity framework for bilinear groups. In S. D. Galbraith and K. G. Paterson, editors, *Pairing*, volume 5209 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2008.

[Boy13] X. Boyen. Attribute-based functional encryption on lattices. In A. Sahai, editor, *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 122–142. Springer, 2013.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *CCS*, pages 62–73. ACM, 1993.

[BS12] D. J. Bernstein and P. Schwabe. NEON crypto. In E. Prouff and P. Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 320–339. Springer, 2012.

[BSW07] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *S&P*, pages 321–334. IEEE, 2007.

[BY19] D. J. Bernstein and B.-Y. Yang. Fast constant-time gcd computation and modular inversion. *TCHES*, 2019(3):340–398, 2019.

[CDS20] R. Clarisse, S. Duquesne, and O. Sanders. Curves with fast computations in the first pairing group. In S. Krenn, H. Shulman, and S. Vaudenay, editors, *CANS*, volume 12579 of *Lecture Notes in Computer Science*, pages 280–298. Springer, 2020.

[CGKW18] J. Chen, J. Gong, L. Kowalczyk, and H. Wee. Unbounded ABE via bilinear entropy expansion, revisited. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT*, volume 10820 of *Lecture Notes in Computer Science*, pages 503–534. Springer, 2018.

[CGT+20] H. Cheng, J. Großschädl, J. Tian, P. B. Rønne, and P. Y. A. Ryan. High-throughput elliptic curve cryptography using AVX2 vector instructions. In O. Dunkelman, M. J. Jacobson Jr., and C. O'Flynn, editors, *SAC*, volume 12804 of *Lecture Notes in Computer Science*, pages 698–719. Springer, 2020.

[CGW15] J. Chen, R. Gay, and H. Wee. Improved dual system ABE in prime-order groups via predicate encodings. In E. Oswald and M. Fischlin, editors, *EUROCRYPT*, volume 9057 of *Lecture Notes in Computer Science*, pages 595–624. Springer, 2015.

[Che06] J. H. Cheon. Security analysis of the strong diffie-hellman problem. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006.

[CS10] C. Costello and D. Stebila. Fixed argument pairings. In M. Abdalla and P. S. L. M. Barreto, editors, *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2010.

[DDP+18] W. Dai, Y. Doröz, Y. Polyakov, K. Rohloff, H. Sajjadpour, E. Savas, and B. Sunar. Implementation and evaluation of a lattice-based key-policy ABE scheme. *IEEE Trans. Inf. Forensics Secur.*, 13(5):1169–1184, 2018.

[Del07] C. Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2007.

[DKW21] P. Datta, I. Komargodski, and B. Waters. Decentralized multi-authority ABE for DNFs from LWE. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT*, volume 12696 of *Lecture Notes in Computer Science*, pages 177–209. Springer, 2021.

[dlPVA22] A. de la Piedra, M. Venema, and G. Alpár. ABE Squared: Accurately benchmarking efficiency of attribute-based encryption. Cryptology ePrint Archive, Report 2022/038, 2022.

[ETS18] ETSI. ETSI TS 103 532 (V1.1.1), 2018.

[FA17] H. Fujii and D. F. Aranha. Curve25519 for the Cortex-M4 and beyond. In T. Lange and O. Dunkelman, editors, *LATINCRYPT*, volume 11368 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2017.

[FL15] A. Faz-Hernández and J. C. López-Hernández. Fast implementation of Curve25519 using AVX2. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *LATINCRYPT*, volume 9230 of *Lecture Notes in Computer Science*, pages 329–345. Springer, 2015.

[FLD19] A. Faz-Hernández, J. C. López-Hernández, and R. Dahab. High-performance implementation of elliptic curve cryptography using vector instructions. *ACM Trans. Math. Softw.*, 45(3):25:1–25:35, 2019.

[FSV07] J. Fan, K. Sakiyama, and I. Verbauwhede. Montgomery modular multiplication algorithm on multi-core systems. In *SiPS*, pages 261–266. IEEE, 2007.

[FSV08] J. Fan, K. Sakiyama, and I. Verbauwhede. Elliptic curve cryptography on embedded multicore systems. *Des. Autom. Embed. Syst.*, 12(3):231–242, 2008.

[Gal14] S. D. Galbraith. New discrete logarithm records, and the death of type 1 pairings. https://ellipticnews.wordpress.com/2014/02/01/new-discrete-logarithm-records-and-the-death-of-type-1-pairings/, 2014.

[GF16] L. Ghammam and E. Fouotsa. Adequate elliptic curves for computing the product of n pairings. In S. Duquesne and S. Petkova-Nikova, editors, *WAIFI*, volume 10064 of *Lecture Notes in Computer Science*, pages 36–53, 2016.

[GGP08] P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC*, volume 5381 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2008.

[GHM+17] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.

[GLV01] 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.

[GMT20] A. Guillevic, S. Masson, and E. Thomé. Cocks-Pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Des. Codes Cryptogr.*, 88(6):1047–1081, 2020.

[GPS08] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discret. Appl. Math.*, 156(16):3113–3121, 2008.

[GPSW06a] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *CCS*. ACM, 2006.

[GPSW06b] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Report 2006/309, 2006.

[GS06] R. Granger and N.P. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006.

[GS19] A. Guillevic and S. Singh. On the alpha value of polynomials in the tower number field sieve algorithm. Cryptology ePrint Archive, Report 2019/885, 2019.

[Gui13] 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, *ACNS*, volume 7954 of *LNCS*, pages 357–372. Springer, 2013.

[Gui20a] A. Guillevic. Pairing-friendly curves. https://members.loria.fr/AGuillevic/pairing-friendly-curves/, 2020.

[Gui20b] A. Guillevic. A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC*, volume 12111 of *Lecture Notes in Computer Science*, pages 535–564. Springer, 2020.

[HFK+19] C. T. Hu, D. F. Ferraiolo, D. R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations, 2019.

[HW14] S. Hohenberger and B. Waters. Online/offline attribute-based encryption. In Hugo Krawczyk, editor, *PKC*, volume 8383 of *Lecture Notes in Computer Science*, pages 293–310. Springer, 2014.

[KB16] T. Kim and R. Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In M. Robshaw and J. Katz, editors, *CRYPTO*, volume 9814 of *Lecture Notes in Computer Science*, pages 543–571. Springer, 2016.

[KSS08] E. J. Kachisa, E. F. Schaefer, and M. Scott. Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In S. D. Galbraith and K. G. Paterson, editors, *Pairing*, volume 5209 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2008.

[KW19] L. Kowalczyk and H. Wee. Compact adaptively secure ABE for NC1 from k-lin. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT*, volume 11476 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2019.

[LL94] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 1994.

[LL20] H. Lin and J. Luo. Compact adaptively secure ABE from k-lin: Beyond nc^1 and towards NL. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT*, volume 12107 of *Lecture Notes in Computer Science*, pages 247–277. Springer, 2020.

[LOS+10] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

[LW10] A. Lewko and B. Waters. Decentralizing attribute-based encryption. Cryptology ePrint Archive, Report 2010/351, 2010.

[LW11a] A. Lewko and B. Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588. Springer, 2011.

[LW11b] A. B. Lewko and B. Waters. Unbounded HIBE and attribute-based encryption. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 547–567. Springer, 2011.

[Lyn13] B. Lynn. The stanford pairing based crypto library. http://crypto.stanford.edu/pbc, 2013.

[Mil04] V. S. Miller. The weil pairing, and its efficient calculation. *J. Cryptol.*, 17(4):235–261, 2004.

[MJ18] Y. Michalevsky and M. Joye. Decentralized policy-hiding ABE with receiver privacy. In J. López, J. Zhou, and M. Soriano, editors, *ESORICS*, volume 11099 of *Lecture Notes in Computer Science*, pages 548–567. Springer, 2018.

[Möl01] B. Möller. Algorithms for multi-exponentiation. In S. Vaudenay and A. M. Youssef, editors, *SAC*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2001.

[MTP+21] M. La Manna, L. Treccozzi, P. Perazzo, S. Saponara, and G. Dini. Performance evaluation of attribute-based encryption in automotive embedded platform for secure software over-the-air update. *Sensors*, 21(2):515, 2021.

[PRMV21] P. Perazzo, F. Righetti, M. La Manna, and C. Vallati. Performance evaluation of attribute-based encryption on constrained iot devices. *Comput. Commun.*, 170:151–163, 2021.

[PTMW10] M. Pirretti, P. Traynor, P. D. McDaniel, and B. Waters. Secure attribute-based systems. *J. Comput. Secur.*, 18(5):799–837, 2010.

[RCS12] 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, *PKC*, volume 7293 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2012.

[RW13] Y. Rouselakis and B. Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *CCS*, pages 463–474. ACM, 2013.

[RW15] Y. Rouselakis and B. Waters. Efficient statically-secure large-universe multi-authority attribute-based encryption. In R. Böhme and T. Okamoto, editors, *FC*, volume 8975 of *Lecture Notes in Computer Science*, pages 315–332. Springer, 2015.

[Sco03] M. Scott. MIRACL cryptographic SDK: Multiprecision Integer and Rational Arithmetic Cryptographic Library. https://github.com/miracl/MIRACL, 2003.

[Sco11] M. Scott. On the efficient implementation of pairing-based protocols. In L. Chen, editor, *IMACC*, volume 7089 of *Lecture Notes in Computer Science*, pages 296–308. Springer, 2011.

[Sha84] A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and D. Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[SKSW20] Y. Sakemi, T. Kobayashi, T. Saito, and R. S. Wahby. Pairing-Friendly Curves. Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-09, Internet Engineering Task Force, November 2020. Work in Progress.

[SLG+14] H. Seo, Z. Liu, J. Großschädl, J. Choi, and H. Kim. Montgomery modular multiplication on ARM-NEON revisited. In J. Lee and J. Kim, editors, *ICISC*, volume 8949 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2014.

[SR13] 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, *ACNS*, volume 7954 of *Lecture Notes in Computer Science*, pages 322–338. Springer, 2013.

[SW05] A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[TKN20] J. Tomida, Y. Kawahara, and R. Nishimaki. Fast, compact, and expressive attribute-based encryption. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC*, volume 12110 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.

[VA21] M. Venema and G. Alpár. A bunch of broken schemes: A simple yet powerful linear approach to analyzing security of attribute-based encryption. In K. G. Paterson, editor, *CT-RSA*, volume 12704 of *Lecture Notes in Computer Science*, pages 100–125. Springer, 2021.

[VAH21] M. Venema, G. Alpár, and J.-H. Hoepman. Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. Cryptology ePrint Archive, Report 2021/1172, 2021.

[Ver10] F. Vercauteren. Optimal pairings. *IEEE Trans. Inf. Theory*, 56(1):455–461, 2010.

[Wat08] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. Cryptology ePrint Archive, Report 2008/290, 2008.

[Wat11] 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, *PKC*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

[WB19] R. S. Wahby and D. Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *TCHES*, 2019(4):154–179, 2019.

[Wee14] H. Wee. Dual system encryption via predicate encodings. In Y. Lindell, editor, *TCC*, volume 8349 of *Lecture Notes in Computer Science*, pages 616–637. Springer, 2014.

[WZSI14] X. Wang, J. Zhang, E. M. Schooler, and M. Ion. Performance evaluation of attribute-based encryption: Toward data privacy in the iot. In *ICC*, pages 725–730. IEEE, 2014.

[ZCa21] ZCash. Company. https://z.cash/, 2021.

[Zeu20] Zeutro. The OpenABE library - open source cryptographic library with attribute-based encryption implementations in C/C++. https://github.com/zeutro/openabe, 2020.

[ZPM+15] E. Zavattoni, L. J. Dominguez Perez, S. Mitsunari, A. H. Sánchez-Ramírez, T. Teruya, and F. Rodríguez-Henríquez. Software implementation of an attribute-based encryption scheme. *IEEE Trans. Computers*, 64(5):1429–1441, 2015.