GLUE: Generalizing Unbounded Attribute-Based Encryption for Flexible Efficiency Trade-Offs

Marloes Venema[1] and Greg Alpár[1,2]

[1] Radboud University, Nijmegen, the Netherlands
[2] Open University of the Netherlands, Heerlen, the Netherlands
{m.venema,g.alpar}@cs.ru.nl

Abstract. Ciphertext-policy attribute-based encryption is a versatile primitive that has been considered extensively to securely manage data in practice. Especially completely unbounded schemes are attractive, because they do not restrict the sets of attributes and policies. So far, any such schemes that support negations in the access policy or that have online/offline extensions have an inefficient decryption algorithm.
In this work, we propose GLUE (Generalized, Large-universe, Unbounded and Expressive), which is a novel scheme that allows for the efficient implementation of the decryption while allowing the support of both negations and online/offline extensions. We achieve these properties simultaneously by uncovering an underlying dependency between encryption and decryption, which allows for a flexible trade-off in their efficiency. For the security proof, we devise a new technique that enables us to generalize multiple existing schemes. As a result, we obtain a completely unbounded scheme supporting negations that, to the best of our knowledge, outperforms all existing such schemes in the decryption algorithm.

Keywords: attribute-based encryption · unbounded attribute-based encryption · online/offline attribute-based encryption · non-monotone attribute-based encryption

1 Introduction

Attribute-based encryption (ABE) is an advanced type of public-key encryption in which the key pairs are associated with attributes [49]. In ciphertext-policy (CP) ABE, messages are encrypted under an access policy [18]. The resulting ciphertexts can then be decrypted by a secret key associated with a set of attributes[3] that satisfies the policy. Conversely, in key-policy (KP) ABE, the keys are associated with access policies and the ciphertexts with sets of attributes [28].

[3] In this paper, we will use the terms "sets of attributes" and "(attribute) sets" to refer to the attributes associated with the secret keys. We use the term "universe of attributes" to refer to the total set of attributes that can be used in the system.

To securely and efficiently implement access control on data, especially pairing-based CP-ABE proves to be an attractive primitive [18,36,50]. In 2018, the European Telecommunications Standards Institute (ETSI) published two technical reports on ABE [26,27], which include detailed descriptions of use cases, varying from cloud settings to mobile networks. In such settings, the computational resources of the key generation, encryption and decryption devices may vary. Thus, different use cases may require schemes with different efficiency trade-offs.

According to ETSI, ABE schemes should be efficient and secure. Interestingly, while ETSI proposes ABE to be used to enforce attribute-based access control [34] on data, it explicitly notes that ABE cannot satisfactorily support it, because ABE cannot support negations efficiently [27]. Indeed, the decryption algorithm of most ABE schemes supporting negations is incredibly expensive [45,38,55,11]. Recently, some interesting progress was made, yielding significant speed-ups in decryption time [51,15]. However, those schemes still have a costly (although less costly) decryption [15] or restrict the attribute sets [51].

In this work, we introduce a new scheme that enables the realization of the following properties:

(1) Large-universe: any string can be used as an attribute;
(2) Unbounded: no restrictions on e.g., the sizes of the policies or attributes sets, or the number of times that an attribute may occur in the policy;
(3) Expressive: support of monotone span programs, ensuring that policies represented as Boolean formulas consisting of conjunctions and disjunctions can be supported;
(4) Non-monotone: support of non-monotone span programs, ensuring that the policies can use negations;
(5) Compact: the key and ciphertext sizes depend, in the worst case, on the size of the set or the length of the policy.

Additionally, this scheme is designed to offer a flexible choice in the encryption/decryption efficiency trade-off during the setup of the parameters. In this way, the scheme can be fine-tuned to take into account the computational resources of the key generation, encryption and decryption devices. In particular, this feature allows for significant speed-ups in the decryption algorithm.

Large-universe and unbounded ABE. The universe of attributes, i.e., the attributes that can be used in the scheme, can be small or large [49]. In small-universe constructions, the number of attributes is bounded after the setup, e.g., because a public key needs to be generated for each attribute. In large-universe constructions, the universe of attributes is effectively unbounded. Moreover, the public keys do not depend on the attributes in the system, and a user can directly encrypt messages using the master public key and the attribute string. Large-universe ABE is thus more scalable than small-universe ABE, as the encrypting users do not need to first locate the necessary public keys before encrypting. Some large-universe schemes [49,54,14] are however undesirably restrictive [41], as they are bounded in the sizes of the policies or attribute sets, or the number

of times that an attribute may occur in the policy. Oftentimes, the scheme's efficiency depends on such bounds, e.g., the encryption costs grow linearly in the bounds on the policies or sets [53]. Hence, choosing high bounds is not a suitable solution either. Preferably, a scheme is unbounded in all parameters.

Expressivity and non-monotonicity. Most state-of-the-art ABE schemes are expressive in that their policies support monotone span programs (MSPs) [2,51,11,42]. An important subclass of MSPs are Boolean formulas consisting of conjunctions and disjunctions. As mentioned, popular access control models such as attribute-based access control [34] allow the policies to be any Boolean formula, including negations. ABE schemes that support negations are called non-monotone [45,55,11,51,15]. In addition to being more expressive, such schemes readily support revocation systems [38], which is crucial in practice as well.

Different types of non-monotonicity. For large-universe constructions, three types of non-monotonicity have been introduced: OSW-type [45], OT-type [44], and OSWOT-type [15]. In the negations considered by Ostrovsky, Sahai and Waters (OSW) [45], e.g., "NOT user: Alice", the entire set of attributes associated with the secret key, e.g., "{user: Bob, course: linear algebra, course: calculus}", is compared with the negated attribute to establish that the set does not contain it. In ABE implementations, this translates in a decryption cost that grows in not only the size of the policy, but also in size of the attribute sets. Such negations may thus not be efficient if the sets are large. In the negations considered by Okamoto and Takashima (OT) [44], e.g., "user: NOT Alice", the attribute labels, e.g., "user", play a role. In particular, the set must contain an attribute with label "user" and its value, e.g., "Bob", must differ from the negated attribute. While this is more efficient than OSW-type negations, the set of attributes is allowed to contain only one attribute for each label, e.g., such negations are not supported for the label "course" in our first example. Thus, schemes supporting this type of negations are bounded in the number of label re-uses, which is not always desirable. For instance, like in our example, users may have multiple attributes for labels such as "departments at a hospital", "courses followed at a university" or "mail addresses". As a solution, Attrapadung and Tomida [15] introduced OSWOT-type negations, e.g., "course: NOT cryptography", to extend OT-type negations, such that the negated attribute is compared with all attributes in the set that share the same label, e.g., "{course: linear algebra, course: calculus}". In this way, the flexibility of OSW-type negations and the efficiency of OT-type negations can be combined.

Achieving properties (1)-(5) simultaneously. Only a limited number of existing schemes support properties (1)-(5) simultaneously [52]. In fact, all pairing-based schemes that provide non-monotonicity and large-universeness use a polynomial-based hash—also known as a "Boneh-Boyen (BB) hash" [19]—that maps arbitrary attribute strings into the scheme [49,45]. Of those schemes, the

Table 1: Comparison of large-universe schemes supporting (non-)monotone span programs. For each scheme, we list whether it is unbounded (in the sets $\mathcal{S}$ and policies $\mathbb{A}$, and the number of attribute and label re-uses), whether it supports negations or has a provably secure extension that supports negations, and whether it is compact. Note that we have only listed schemes that are structurally different "in the exponent", i.e., their associated pair encodings [9] are different. For instance, the unbounded schemes in [37,41] have a similar structure and therefore only [41] is listed.

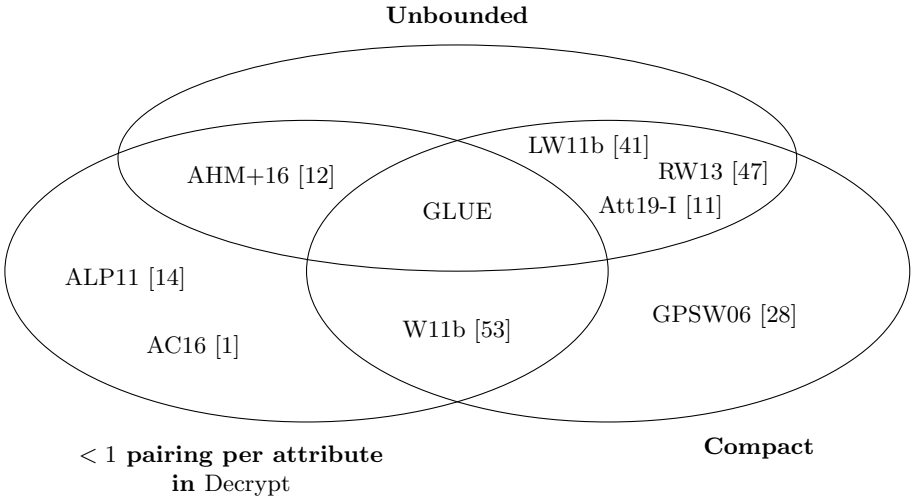
Scheme	**KP/CP**	**Unbounded**				**Negations**			**Compact**
		$\lvert\mathcal{S}\rvert$	$\lvert\mathbb{A}\rvert$	ARU	LRU	OSW	OT	OSWOT	
GPSW06-LU [28]	KP	✗	✓	✓	✓	✓ [45]	✗	✗	✓
BSW07 [18]	CP	✓	✓	✓	✓	✗	✗	✗	✓
ALP11 [14]	KP	✗	✓	✓	✓	✓	✗	✗	✗
W11-LU I [54]	CP	✓	✓	✓	✓	✗	✗	✗	✓
W11b-LU [53]	CP	✗	✗	✗	✗	✗	✗	✗	✓
LW11b [41,47]	KP	✓	✓	✓	✓	✓ [38]	✓ [11]	✓ [15]	✓
OT12 [44]	CP	✓	✓	✗	✗	✗	✓	✗	✗
RW13 [47]	CP	✓	✓	✓	✓	✓ [55]	✓ [11]	✓ [15]	✓
AHM+16 [12]	KP	✓	✓	✓	✓	✓ [14]	✗	✗	✗
AC16 [1]	CP	✗	✗	✓	✓	✓ [11,6]	✓ [15,6]	✓ [15,6]	✗
AC17b [2]	CP(/KP)	✓	✓	✗	✗	✗	✗	✗	✓
ABGW17 [7]	KP/CP	✓	✓	✓	✓	✗	✗	✗	✓
Att19-I-CP [11]	CP(/KP)	✓	✓	✓	✓	✓	✗	✓ [15]	✓
Att19-II-CP [11]	CP(/KP)	✗	✓	✓	✓	✓	✗	✗	✗
Att19-III-CP [11]	CP(/KP)	✓	✗	✓	✓	✓	✗	✗	✗
TKN20 [51]	KP/CP	✓	✓	✓	✗	✗	✓	✗	✓
GLUE	CP(/KP)	✓	✓	✓	✓	✓	✓	✓ [15]	✓

Note: $\mathcal{S}$ = attribute set; $\mathbb{A}$ = access policy;
ARU = attribute re-use in the policies; LRU = label re-use in the sets and policies

only ones that are completely unbounded [38,55,11,15] are based on the schemes by Lewko and Waters (the KP-ABE version) [41] and Rouselakis and Waters (the CP-ABE version) [47] (Table 1). However, all those schemes have an inefficient decryption algorithm compared to other ABE schemes [46], which use a full-domain hash (FDH) to achieve large-universeness. For this reason, such schemes are often favored in practice, despite their inability to support negations [26,27]. Nevertheless, since supporting negations fosters the expressivity of ABE, we aim at improving the decryption efficiency of schemes using a BB hash.

Improving decryption efficiency of schemes using a BB hash. To determine whether we can improve on the decryption efficiency of the existing schemes satisfying properties (1)-(5), we investigate *all* schemes using a BB hash to achieve large-universeness. In particular, if we consider all such schemes, then we see that a scheme that is unbounded, compact and costs less than one

Fig. 1: Overview of large-universe schemes using a BB hash.



pairing operation per attribute during decryption does not exist yet (Figure 1). Because pairing operations are the most expensive operations in pairing-based ABE, it is therefore important to minimize the use of those. In this work, we aim to achieve this: we provide a scheme that satisfies properties (1)-(5), while requiring less than one pairing operation per attribute during decryption.

1.1 Our contributions

We first give a high-level overview of our contributions. Then, we provide more (technical) details about these contributions.

- **New construction:** We present a new unbounded large-universe scheme using a BB hash (thus avoiding random oracles). Its encryption/decryption efficiency trade-off can be fine-tuned by taking into account the computational resources of the devices.
- **Generalizations:** Concretely, the scheme can be considered a generalization of two large-universe schemes: the Rouselakis-Waters (RW13) scheme [47] and the bounded large-universe scheme without random oracles by Waters (W11b) [53]. This generalization also illustrates a deeper connection among various designs.
- **Security proof:** We develop new proof techniques to ensure that the randomness provided by a BB hash can be simultaneously used for the keys and ciphertexts. To the best of our knowledge, we are the first to achieve this in the unbounded setting, and in the full-security setting.
- **Extensions:** We provide three extensions to the basic scheme: one online/offline [33] and two non-monotone extensions supporting OT-type and

OSW-type negations, respectively. Notably, we obtain an online/offline ABE scheme and a scheme supporting OSW-type negations with the most efficient decryption algorithms. This enables us to support OSWOT-type negations more efficiently, which is the most desirable in practice.

1.2 New construction: GLUE

We focus on three schemes that satisfy at least two out of the three depicted properties (see Figure 1): W11b [53], RW13 [47] and AHM+16 [12]. Those three schemes provide a good starting point for GLUE, our new scheme which satisfies all required properties. Intuitively, we apply the partitioning techniques of AHM+16 to combine the unbounded RW13 and the bounded W11b that allows for efficient decryption. However, as we show later, for the secure combination of these techniques, GLUE requires a more intricate approach.

We give a high-level description of the partitioning approach as introduced by AHM+16. First, we partition the attribute sets into smaller subsets. Then, we apply a (bounded) scheme with efficient decryption (in their case, ALP11 [14]) to each subset. Lastly, we use the unbounded techniques of e.g., RW13 or LW11b [41] to securely connect the subsets. In this way, the decryption costs of the scheme can be decreased. Unfortunately, this comes at a cost. Because bounded schemes typically have a more expensive encryption, the encryption costs are increased. From a broader perspective, this approach creates a scheme with a (flexibly) efficiency trade-off feature. As we will show later, this trade-off is determined by some parameter n. The encryption costs are higher by a factor n than those of unbounded schemes such as RW13, whereas the decryption costs are lower by this factor. Because this parameter n can be chosen during setup, it can be fine-tuned for the given practical context. If decryption needs to be efficient (which is often the case), one can choose larger n than in cases in which encryption needs to be efficient.

The main reason why we achieve the compactness property, in contrast to AHM+16, is due to the bounded scheme that is used. Because AHM+16 uses ALP11 [14], a scheme with constant-size ciphertexts and large keys whose sizes depend on the parameter n, its keys are large and its key generation is very expensive. Moreover, although the number of pairing operations required during decryption decreases, the number of exponentiations grows by a factor n for each matching attribute. As we will show, this means that AHM+16 decryption is not much more efficient than unbounded schemes such as RW13. As a solution, we use the W11b scheme, whose decryption costs consist of a constant number of pairing operations and no additional exponentiations. In this way, we achieve a much better speed-up in decryption, and since W11b is compact, the key sizes and key generation costs are not affected.

1.3 Generalizing RW13 by generalizing the hash

The main difference between the partitioning approach as applied by AHM+16 and us is that we have to partition both the key sets and the ciphertext poli-

cies. The reason for this is that AHM+16 uses ALP11, which is bounded in only the ciphertexts, whereas we use W11b, which is bounded in both the keys and ciphertexts. By extension, we need to apply some technique to connect the resulting key and ciphertext "parts". However, we will show that the security proof of W11b does not generalize to the unbounded setting, meaning that we have to devise a new proof technique for W11b that does generalize. Furthermore, it is not possible to apply the exact same approach as that of AHM+16. In particular, to prove security, they embed the scheme in the fully secure key-policy doubly-spatial encryption [32] scheme in [9], and then, they apply the embedding lemma [12]. We cannot use this approach, because, to the best of our knowledge, the W11b scheme cannot be embedded in an existing scheme in a similar fashion.

Hence, we take a slightly different approach: we generalize RW13 by generalizing its specific instantiation of the BB hash. A BB hash first takes as input a unique representation x_{att} of an attribute att in the integer set $\mathbb{Z}_p$, where p is the prime order of group $\mathbb{G}$ with generator $g \in \mathbb{G}$. Then, it computes the hash as $F_n(x_{\text{att}}) = \prod_{i=0}^{n} B_i^{x_{\text{att}}^i}$, where the generators $B_i = g^{b_i}$ implicitly embed the coefficients of the polynomial $f_n(x_{\text{att}}) = \sum_{i=0}^{n} b_i x_{\text{att}}^i$. Where RW13 (and its unbounded derivatives [33,11]) uses an implicit 1-degree polynomial, we use an implicit n-degree polynomial, like W11b [53]. However, as we will show, simply replacing the 1-degree polynomial by some n-degree polynomial does not immediately yield a secure scheme. To solve this, we replace another public-key variable used in the scheme by a polynomial.

1.4 Security proof

The AC17 framework. To benefit from the strong security guarantees as well as the generic transformations [11,6] within the Agrawal-Chase (AC17) [3] framework, we prove security within it. In general, the AC17 framework considers the pair encoding schemes (PESs) associated with an ABE scheme [9]. Intuitively, PESs are an abstraction of ABE schemes to what happens in the exponent space. If a PES is secure, the AC17 framework transforms it into a fully (also known as adaptively) secure scheme. The security notion for PES is called symbolic security, which consists of two parts: the selective and co-selective symbolic property. These properties are similar to selective security—which was commonly used before the first fully secure scheme [40]—and its counterpart: co-selective security [13]. These types of security are weaker than full security, because the attacker needs to commit to either the access policy or the set of attributes, before the master public key is generated. Nevertheless, they are easier to prove, so the AC17 framework simplifies the effort of proving full security considerably by giving such transformations. A small drawback is that the resulting schemes are provably secure under q-type assumptions [20], which are less well-understood than static security assumptions. Regardless, the assumptions used in the AC17 framework are implied by commonly-used q-type assumptions [9]. These have not been shown to yield less secure schemes in practice yet. Another advantage

of the AC17 framework is that any symbolically secure PES can be transformed in a PES that supports negations [6]. It is notoriously difficult to achieve this in the full-security model in combination with the large-universe property [11].

Proving the symbolic property. Proving the (co-)selective symbolic property is similar to proving selective security. Recall that, in these models, the attacker commits to the challenge access policy (resp. set of attributes). This is used in "program-and-cancel" proofs [54,47], in which the challenger embeds the policy (resp. set) in the public keys. In the simulation of the secret keys and challenge ciphertext, the components are programmed in a specific way, using that the set does not satisfy the policy (resp. policy is not satisfied by the set). Typically, the components that cannot be programmed are canceled by other non-programmable components. In the AC17 framework, this "programming" is replaced by "substitution", and the "canceling" is replaced by "evaluating to 0".

Security proof. One of the main difficulties of our scheme is proving selective and co-selective security. In the first place, proving security is difficult due to the lack of provably secure schemes that use the randomness provided by the BB hash for *both the keys and ciphertexts*. To the best of our knowledge, previously, only W11b [53] used the hash for both the keys and ciphertexts, but only in the bounded setting and in the selective-security model. However, the proof does not seem to readily generalize to the unbounded setting (see Appendix A). Hence, we develop a novel technique to prove security. We do this, in part, by combining several techniques.

- **Proof techniques using the hash for the keys:** We generalize the proof techniques used by Agrawal and Chase in [3] to prove full security of their scheme in [1]: the AC16 [1] scheme. AC16 is a CP-ABE scheme with constant-size ciphertexts, in which the randomness provided by the Boneh-Boyen hash is used for the *keys*. In the selective proof, the polynomial embedded in the public keys needs to be used by the secret keys after the public keys are generated. The proof does this by embedding a "reprogrammable" polynomial in the public keys. We call these polynomials to be reprogrammable in the sense that the randomizers of the secret keys can later program it to a suitable target polynomial. We use this general strategy for the keys.
- **Proof techniques using the hash for the ciphertext:** Even though the W11b [53] proof does not generalize to the unbounded setting, we are able to use a part of the proof strategy. In the selective proof, the implicit polynomial embedded in the public key is "programmed" to take into account the attributes that will be used in the challenge ciphertext. We use this general strategy for the ciphertexts.
- **Unbounded proof techniques:** One of the bottlenecks of the two aforementioned strategies is that they are bounded approaches: they use only one randomizer for the keys and one for the ciphertexts. To make them unbounded, we use the general approach of the RW13 [47] proof. This proof

gives us a rough idea of how the implicit polynomial and the randomizers should be programmed. Furthermore, it shows us how to use the polynomial an unbounded number of times: using layering and individual randomness techniques allows us to select the required instance of the polynomial.

Another bottleneck is that the "programmed" and "reprogrammable" approaches are orthogonal, and can therefore not be used simultaneously in the same polynomial without applying a trick. Presumably, this is also the reason why the W11b proof uses the programmed approach for both the keys and the ciphertexts, and applies an algebraic argument to ensure that everything can be simulated as required. We eliminate this bottleneck and combine all these proof techniques, by splitting the polynomial in the product of two smaller polynomials: one "programmed" polynomial and one "reprogrammable" polynomial. For the selective proof, we use the programmed polynomial for the ciphertexts and the reprogrammable polynomial for the keys. For the co-selective proof, the roles of the polynomials are reversed.

1.5 Practical extensions

We provide several extensions to our scheme. Because we prove security in the AC17 [3] framework, some of these extensions are automatically provably secure.

- **The key-policy and dual-policy versions**, by applying [11]. The key-policy version can be found in Appendix D;
- **Online/offline extensions**, by generalizing HW14 [33], which we do in Appendix E. Owing to its generality, these extensions also apply to the following extensions;
- **Non-monotone versions**, by applying [6,11,51]:
  - **OT-type:** the PES can be found in Appendix F.1;
  - **OSW-type:** the PES can be found in Appendix F.2.

Online/offline extension. The algebraic structure of large-universe schemes using a BB hash can also be exploited to increase the efficiency. Hohenberger and Waters [33] show how the key generation and encryption of RW13 [47] can be split in an online and offline phase. In this way, most of the computations required by these algorithms can be performed in an offline phase. During the online phase, little computational power is required. This is especially useful in practice when secret keys need to be generated frequently, e.g., in revocation systems [48]. The key generation authority then does not need to have computationally powerful resources to do this in an acceptable time frame. Similarly, the online/offline encryption variant can be used to minimize the encryption costs. This comes however at a cost: the decryption costs increase. Thus, reducing the decryption costs of the basic scheme also helps reducing the decryption costs of the online/offline version. With the online/offline version of GLUE, we can not only improve on the decryption efficiency of existing such schemes, but we can also mitigate the impact of the increase of n on the encryption efficiency.

Table 2: Theoretical efficiency comparison of all compact unbounded large-universe CP-ABE schemes supporting MSPs (that (can) support OSW(OT)-type negations), by analyzing the key generation, encryption and decryption costs with respect to the number of exponentiations c_{exp} and pairings c_{pair}.

Scheme	**Key generation**	**Encryption**	**Decryption**	
	c_{exp}	c_{exp}	c_{exp}	c_{pair}
RW13 [47]	$2+2\lvert\mathcal{S}\rvert$	$2+5\lvert\mathbb{A}\rvert$	$2\lvert\Upsilon\rvert$	$2+2\lvert\Upsilon\rvert$
Att19-I-CP [11]	$6+6\lvert\mathcal{S}\rvert$	$2+8\lvert\mathbb{A}\rvert$	$2\lvert\Upsilon\rvert$	$4+2\lvert\Upsilon\rvert$
GLUE	$2+\lvert\mathcal{S}\rvert+\left\lceil\frac{\lvert\mathcal{S}\rvert}{n_k}\right\rceil$	$2+\lvert\mathbb{A}\rvert(n_k+2n_c+1)+\left\lceil\frac{\lvert\mathbb{A}\rvert}{n_c}\right\rceil$	$2\lvert\Upsilon\rvert$	$2+\left\lceil\frac{\lvert\Upsilon\rvert}{n_k}\right\rceil+\left\lceil\frac{\lvert\Upsilon\rvert}{n_c}\right\rceil$
GLUE-N	$6+4\lvert\mathcal{S}\rvert+2\left\lceil\frac{\lvert\mathcal{S}\rvert}{n_k}\right\rceil$	$2+\lvert\mathbb{A}\rvert(n_k+2n_c+4)+\left\lceil\frac{\lvert\mathbb{A}\rvert}{n_c}\right\rceil$	$2\lvert\Upsilon\rvert$	$4+\left\lceil\frac{\lvert\Upsilon\rvert}{n_k}\right\rceil+\left\lceil\frac{\lvert\Upsilon\rvert}{n_c}\right\rceil$

(a) Costs for non-negated policies

Scheme	c_{exp}	c_{pair}
Att19-I-CP [11]	$2\lvert\Upsilon\rvert\cdot\lvert\mathcal{S}\rvert$	$4+\lvert\Upsilon\rvert+\min(\lvert\Upsilon\rvert,\lvert\mathcal{S}\rvert)$
GLUE-N (worst case)	$2\lvert\Upsilon\rvert\cdot\lvert\mathcal{S}\rvert+\left\lceil\frac{\lvert\mathcal{S}\rvert}{n_k}\right\rceil\cdot\lvert\Upsilon\rvert$	$4+\lvert\Upsilon\rvert+\left\lceil\frac{\lvert\mathcal{S}\rvert}{n_k}\right\rceil$
GLUE-N (best case)	$2\left\lceil\frac{\lvert\Upsilon\rvert}{n_c}\right\rceil\cdot\lvert\mathcal{S}\rvert+\left\lceil\frac{\lvert\mathcal{S}\rvert}{n_k}\right\rceil\cdot\lvert\Upsilon\rvert$	$4+\left\lceil\frac{\lvert\Upsilon\rvert}{n_c}\right\rceil+\left\lceil\frac{\lvert\mathcal{S}\rvert}{n_k}\right\rceil$

(b) Decryption costs for negated policies

Note: $\mathcal{S}$ = attribute set; $\mathbb{A}$ = access policy; Υ = matching attributes

1.6 Efficiency comparison with existing schemes supporting (1)-(5)

We generalize RW13 to achieve a scheme that supports or can support properties (1)-(5) whilst being able to achieve a more efficient decryption algorithm. In Table 2, we compare the efficiency of RW13 and its OSW-type non-monotone variant Att19-I-CP with GLUE (which supports MSPs only) and GLUE-N (which additionally supports OSW-type negations). In Section 6, we give more concrete estimates for the timings in practice and how they compare to existing schemes.

2 Preliminaries

2.1 Notation

If an element is chosen uniformly at random from a finite set S, then we denote this as $x \in_R S$. For integers $a < b$, we denote $[a,b] = \{a, a+1, ..., b-1, b\}$, $[b] = [1,b]$ and $\overline{[b]} = [0,b]$. We use boldfaced variables $\mathbf{A}$ and $\mathbf{v}$ for matrices and vectors, respectively. We denote $a : \mathbf{A}$ to substitute variable a by a matrix or vector $\mathbf{A}$. We define $\mathbf{1}_{i,j}^{d_1\times d_2} \in \mathbb{Z}_p^{d_1\times d_2}$ as the matrix with 1 in the i-th row and j-th column, and 0 everywhere else, and similarly $\mathbf{1}_i^{d_1}$ and $\overline{\mathbf{1}}_i^{d_2}$ as the row and column vectors with 1 in the i-th entry and 0 everywhere else.

2.2 Access structures

We represent access policies $\mathbb{A}$ by linear secret sharing scheme (LSSS) matrices, which support monotone span programs [17,29].

Definition 1 (Access structures represented by LSSS matrices [29]). *An access structure can be represented as a pair $\mathbb{A} = (\mathbf{A}, \rho)$ such that $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ is an LSSS matrix, where $n_1, n_2 \in \mathbb{N}$, and ρ is a function that maps its rows to attributes in the universe. Then, for some vector $\mathbf{v} = (s, v_2, ..., v_{n_2}) \in_R \mathbb{Z}_p^{n_2}$, 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 $\mathbf{A}$. In particular, if $\mathcal{S}$ satisfies $\mathbb{A}$, then there exist a set of rows $\Upsilon = \{i \in [n_1] \mid \rho(i) \in \mathcal{S}\}$ and coefficients $\varepsilon_i \in \mathbb{Z}_p$ for all $i \in \Upsilon$ such that $\sum_{i \in \Upsilon} \varepsilon_i \mathbf{A}_i = (1, 0, ..., 0)$, and by extension $\sum_{i \in \Upsilon} \varepsilon_i \lambda_i = s$, holds.*

If $\mathcal{S}$ does not satisfy $\mathbb{A}$, there exists $\mathbf{w} = (1, w_2, ..., w_{n_2}) \in \mathbb{Z}_p^{n_2}$ such that $\mathbf{A}_i \mathbf{w}^\intercal = 0$ for all $i \in \Upsilon$ [17].

2.3 Ciphertext-policy ABE

Definition 2 (Ciphertext-policy ABE [18]). *A ciphertext-policy ABE (CP-ABE) scheme 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).
- $\text{KeyGen}(\text{MSK}, \mathcal{S}) \to \text{SK}_\mathcal{S}$*: The key generation takes as input a set of attributes $\mathcal{S}$ and the master secret key* MSK*, and outputs a secret key* $\text{SK}_\mathcal{S}$.
- $\text{Encrypt}(\text{MPK}, \mathbb{A}, M) \to \text{CT}_\mathbb{A}$*: The encryption takes as input a plaintext message M, an access policy $\mathbb{A}$ and the master public keys* MPK*. It outputs a ciphertext* $\text{CT}_\mathbb{A}$.
- $\text{Decrypt}(\text{SK}_\mathcal{S}, \text{CT}_\mathbb{A}) \to M'$*: The decryption takes as input the ciphertext* $\text{CT}_\mathbb{A}$ *that was encrypted under an access policy $\mathbb{A}$, and a secret key* $\text{SK}_\mathcal{S}$ *associated with a set of attributes $\mathcal{S}$. It succeeds and outputs the plaintext message M' if $\mathcal{S}$ satisfies $\mathbb{A}$. Otherwise, it aborts.*

A scheme is called correct if decryption of a ciphertext with secret key yields the original plaintext message.

Large-universe and unbounded ABE. We consider a scheme to be large-universe if it does not impose bounds on the size of the universe. We call it unbounded, if it does not impose bounds on the sizes of the universe, sets of attributes and access policies, or on the number of times that an attribute occurs in an access policy. The term "unbounded ABE" is more prominently used for schemes that achieve this without requiring random oracles in the proof.

2.4 Security model

Definition 3 (Full CPA-security for CP-ABE [18]). *We define the game between challenger and attacker as follows:*

- ***Setup phase:*** *The challenger runs the Setup algorithm and sends the master public key* MPK *to the attacker.*
- ***First query phase:*** *The attacker queries secret keys for the sets of attributes* $\mathcal{S}_1, ..., \mathcal{S}_{n_1}$.
- ***Challenge phase:*** *The attacker specifies an access structure* $\mathbb{A}^*$ *such that none of the sets* $\mathcal{S}_i$ *satisfies it, generates two messages* M_0 *and* M_1 *of equal length, and sends these to the challenger. The challenger flips a coin, i.e.,* $\beta \in_R \{0, 1\}$*, encrypts* M_β *under* $\mathbb{A}^*$*, and sends the resulting ciphertext to the attacker.*
- ***Second query phase:*** *The attacker queries secret keys for the sets of attributes* $\mathcal{S}_{n_1+1}, ..., \mathcal{S}_{n_2}$ *with the restriction that none of the sets* $\mathcal{S}_i$ *satisfy access structure* $\mathbb{A}^*$.
- ***Decision phase:*** *The attacker outputs a guess* β' *for* β.

The advantage of the attacker is defined as $|\Pr[\beta' = \beta] - \frac{1}{2}|$. *A CP-ABE scheme is fully secure if all polynomial-time attackers have at most a negligible advantage in this security game.*

2.5 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, such that $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$, 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 unique identity element 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.

2.6 Pair encoding schemes

We define pair encoding schemes, and their associated security notions: the selective and co-selective symbolic properties. Informally speaking, pair encoding schemes abstract a pairing-based scheme to "what happens in the exponent".

Definition 4 (Pair encoding schemes (PES) [3]). *A pair encoding scheme for a predicate family* $P_\kappa\colon \mathcal{X}_\kappa \times \mathcal{Y}_\kappa \to \{0, 1\}$*, indexed by* $\kappa = (p, \text{par})$*, where* par *specifies some parameters, is given by four deterministic polynomial-time algorithms as described below.*

- $\text{Param}(\text{par}) \to n$*: On input* par*, the algorithm outputs* $n \in \mathbb{N}$ *that specifies the number of common variables, which are denoted as* $\mathbf{b} = (b_1, ..., b_n)$.

– $\mathrm{EncKey}(y,p) \to (m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \mathbf{b}))$*: On input* $p \in \mathbb{N}$ *and* $y \in \mathcal{Y}_\kappa$*, this algorithm outputs a vector of polynomials* $\mathbf{k} = (k_1, ..., k_{m_3})$ *defined over non-lone variables* $\mathbf{r} = (r_1, ..., r_{m_1})$ *and lone variables* $\hat{\mathbf{r}} = (\hat{r}_1, ..., \hat{r}_{m_2})$*. Specifically, the polynomial* k_i *is expressed as*

$$k_i = \delta_i \alpha + \sum_{j \in [m_2]} \delta_{i,j} \hat{r}_j + \sum_{j \in [m_1], k \in [n]} \delta_{i,j,k} r_j b_k,$$

for all $i \in [m_3]$*, where* $\delta_i, \delta_{i,j}, \delta_{i,j,k} \in \mathbb{Z}_p$*.*

– $\mathrm{EncCt}(x,p) \to (w_1, w_2, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}))$*: On input* $p \in \mathbb{N}$ *and* $x \in \mathcal{X}_\kappa$*, this algorithm outputs a vector of polynomials* $\mathbf{c} = (c_1, ..., c_{w_3})$ *defined over non-lone variables* $\mathbf{s} = (s, s_2, ..., s_{w_1})$ *and lone variables* $\hat{\mathbf{s}} = (\hat{s}_1, ..., \hat{s}_{w_2})$*. Specifically, the polynomial* c_i *is expressed as*

$$c_i = \sum_{j \in [w_2]} \eta_{i,j} \hat{s}_j + \sum_{j \in \overline{[w_1]}, k \in [n]} \eta_{i,j,k} s_j b_k,$$

for all $i \in [w_3]$*, where* $\eta_{i,j}, \eta_{i,j,k} \in \mathbb{Z}_p$*.*

– $\mathrm{Pair}(x,y,p) \to (\mathbf{E}, \overline{\mathbf{E}})$*: On input* p*,* x*, and* y*, this algorithm outputs two matrices* $\mathbf{E}$ *and* $\overline{\mathbf{E}}$ *of sizes* $(w_1+1) \times m_3$ *and* $w_3 \times m_1$*, respectively.*

A PES is correct for every $\kappa = (p, \mathrm{par})$*,* $x \in \mathcal{X}_\kappa$ *and* $y \in \mathcal{Y}_\kappa$ *such that* $P_\kappa(x,y) = 1$*, it holds that* $\mathbf{s}\mathbf{E}\mathbf{k}^\intercal + \mathbf{c}\overline{\mathbf{E}}\mathbf{r}^\intercal = \alpha s$*.*

Definition 5 (Symbolic property [3]). *A pair encoding scheme* $\Gamma = (\mathrm{Param}, \mathrm{EncKey}, \mathrm{EncCt}, \mathrm{Pair})$ *for a predicate family* $P_\kappa \colon \mathcal{X}_\kappa \times \mathcal{Y}_\kappa \to \{0,1\}$ *satisfies the* (d_1, d_2)*-selective symbolic property for positive integers* d_1 *and* d_2 *if there exist deterministic polynomial-time algorithms* EncB*,* EncS*, and* EncR *such that for all* $\kappa = (p, \mathrm{par})$*,* $x \in \mathcal{X}_\kappa$ *and* $y \in \mathcal{Y}_\kappa$ *with* $P_\kappa(x,y) = 0$*, we have that*

– $\mathrm{EncB}(x) \to \mathbf{B}_1, ..., \mathbf{B}_n \in \mathbb{Z}_p^{d_1 \times d_2}$*;*
– $\mathrm{EncR}(x,y) \to \mathbf{r}_1, ..., \mathbf{r}_{m_1} \in \mathbb{Z}_p^{d_1}, \mathbf{a}, \hat{\mathbf{r}}_1, ..., \hat{\mathbf{r}}_{m_2} \in \mathbb{Z}_p^{d_2}$*;*
– $\mathrm{EncS}(x) \to \mathbf{s}_0, ..., \mathbf{s}_{w_1} \in \mathbb{Z}_p^{d_2}, \hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_{w_2} \in \mathbb{Z}_p^{d_1}$*;*

such that $\langle \mathbf{s}_0, \mathbf{a} \rangle \neq 0$*, and if we substitute*

$$\hat{s}_{i'} : \hat{\mathbf{s}}_{i'}^\intercal \qquad s_i b_j : \mathbf{B}_j \mathbf{s}_i^\intercal \qquad \alpha : \mathbf{a} \qquad \hat{r}_{k'} : \hat{\mathbf{r}}_{k'} \qquad r_k b_j : \mathbf{r}_k \mathbf{B}_j,$$

for $i \in [w_1], i' \in [w_2], j \in [n], k \in [m_1], k' \in [m_2]$ *in all the polynomials of* $\mathbf{k}$ *and* $\mathbf{c}$ *(output by* EncKey *and* EncCt*, respectively), they evaluate to* $\mathbf{0}$*.*

Similarly, a pair encoding scheme satisfies the (d_1, d_2)*-co-selective symbolic security property if there exist* EncB, EncR, EncS *that satisfy the above properties but where* EncB *and* EncR *only take* y *as input, and* EncS *takes* x *and* y *as input.*

A scheme satisfies the (d_1, d_2)*-symbolic property if it satisfies the* (d_1', d_2')*-selective and* (d_1'', d_2'')*-co-selective properties for* $d_1', d_1'' \leq d_1$ *and* $d_2', d_2'' \leq d_2$*.*

Agrawal and Chase [3] prove that any PES satisfying the (d_1, d_2)-symbolic property can be transformed in a fully secure ABE scheme.

3 Generalizing Rouselakis-Waters

We first show how RW13 [47] can be generalized. On a high level, we do this by substituting the implicit 1-degree polynomial in the RW13 keys and ciphertexts with an n-degree polynomial. Like W11b [53], the randomness provided by this n-degree polynomial will be shared between the keys and ciphertexts. That is, suppose that n_k and n_c are positive integers such that $n = n_k + n_c - 1$, then the n-degree polynomial provides enough randomness for $n_k - 1$ attributes in the keys, and n_c attributes in the ciphertext. To optimally use this randomness, we therefore split the keys and ciphertexts in partitions of at most n_k and n_c attributes, respectively. For instance, if $\mathcal{S}$ denotes the set of attributes for which a key is requested, then $\mathcal{S}$ is split in partitions of maximum size n_k, i.e., $\mathcal{S} = \mathcal{S}_1 \cup ... \cup \mathcal{S}_m$ such that $|\mathcal{S}_l| \leq n_k$ for each $l \in [m]$. Then, to avoid boundedness, we apply the RW13 trick by introducing one "randomizer" for each partition (both in the keys and ciphertexts).

3.1 The Rouselakis-Waters scheme

First, we briefly review the RW13 scheme [47]. Specifically, the secret keys and ciphertexts are of the form

$$\begin{gathered}\mathrm{SK} = (K = h^{\alpha - rb}, K' = h^r, \{K_{1,\mathrm{att}} = h^{rb' + r_{\mathrm{att}}(x_{\mathrm{att}} b_1 + b_0)}, K_{2,\mathrm{att}} = h^{r_{\mathrm{att}}}\}_{\mathrm{att} \in \mathcal{S}}),\\ \mathrm{CT} = (C = M \cdot e(g,h)^{\alpha s}, C' = g^s, \{C_{1,j} = B^{\lambda_j} \cdot (B')^{s_j},\\ C_{2,j} = (B_1^{x_{\mathrm{att}_j}} B_0)^{s_j}, C_{3,j} = g^{s_j}\}_{j \in [n_1]}),\end{gathered}$$

where $B = g^b, B_1 = g^{b_1}, B_0 = g^{b_0}$ and $B' = g^{b'}$ denote public keys, $r, r_{\mathrm{att}} \in_R \mathbb{Z}_p$ are randomly chosen integers during the key generation for set $\mathcal{S}$, s, s_j are randomly chosen integers during encryption under access policy $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$, and x_{att} is the representation of attribute att in $\mathbb{Z}_p$. We have also denoted $x_{\mathrm{att}_j} = \rho(j)$ to clearly indicate the attributes in the ciphertext.

3.2 First attempt: a naive approach

Our first attempt is to directly replace the 1-degree polynomial, $x_{\mathrm{att}} b_1 + b_0$, by an n-degree polynomial, i.e., $f_n(x_{\mathrm{att}}) = \sum_{i=0}^{n} b_i x_{\mathrm{att}}^i$ (where $n = n_k + n_c - 1$):

$$\begin{gathered}\mathrm{SK} = (K = h^{\alpha - rb}, K' = h^r, \{K_{1,\mathrm{att}} = h^{rb' + \boxed{r_{\mathrm{att}} f_n(x_{\mathrm{att}})}}, K_{2,\mathrm{att}} = h^{r_{\mathrm{att}}}\}_{\mathrm{att} \in \mathcal{S}}),\\ \mathrm{CT} = (C = M \cdot e(g,g)^{\alpha s}, C' = g^s, \{C_{1,j} = B^{\lambda_j} \cdot (B')^{s_j},\\ C_{2,j} = \boxed{F_n(x_{\mathrm{att}_j})^{s_j}} = \left(\textstyle\prod_{i=0}^{n} B_i^{x_{\mathrm{att}_j}^i}\right)^{s_j}, C_{3,j} = g^{s_j}\}_{j \in [n_1]}),\end{gathered}$$

where $B_i = g^{b_i}$ for all $i \in [0, n]$. We split $\mathcal{S}$ in partitions of maximum size n_k, and the rows of $\mathbb{A}$ in partitions of size n_c. We ensure that the same randomizer is used for all attributes in the same partition, i.e., set $r_{\mathrm{att}} = r_{\mathrm{att}'}$ and $s_j = s_{j'}$, if att and att$'$, and att_j and $\mathrm{att}_{j'}$ are in the same partitions, respectively.

Unfortunately, the resulting scheme is insecure (see Appendix B for a concrete attack). Roughly, the reason is that $C_{1,j} = g^{\lambda_j b + s_j b'}$ does not sufficiently hide $\lambda_j b$, because the same s_j is used for all attributes in the same partition. Therefore, we need to introduce more randomness.

3.3 Second (successful) attempt

We show how to use another polynomial to introduce enough randomness. Because we only need enough randomness for the ciphertext partitions, we require an (n_c-1)-degree polynomial. This polynomial, $f'_{n_c-1}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} b'_i x_{\text{att}}^i$, will replace the "0-degree polynomial" b'. Because s_j provides randomness for one attribute, and f'_{n_c-1} provides randomness for $n_c - 1$ attributes in the partition, this sufficiently hides λ_j. The resulting scheme is then

$$\begin{aligned}
\text{SK} = (K = h^{\alpha - rb}, K' = h^r, \{K_{1,\text{att}} = h^{\boxed{r f'_{n_c-1}(x_{\text{att}})} + \boxed{r_{\text{att}} f_n(x_{\text{att}})}},\\
K_{2,\text{att}} = h^{r_{\text{att}}}\}_{\text{att}\in\mathcal{S}}),\\
\text{CT} = (C = M \cdot e(g,h)^{\alpha s}, C' = g^s, \{C_{1,j} = B^{\lambda_j} \cdot \boxed{F'_{n_c-1}(x_{\text{att}_j})^{s_j}},\\
C_{2,j} = \boxed{F_n(x_{\text{att}_j})^{s_j}}, C_{3,j} = g^{s_j}\}_{j\in[n_1]}),
\end{aligned}$$

where $F'_{n_c-1}(x_{\text{att}}) = \prod_{i=0}^{n_c-1} (B'_i)^{x_{\text{att}}^i} = g^{f'_{n_c-1}(x_{\text{att}})}$, and $B'_i = g^{b'_i}$ for all $i \in [0, n_c - 1]$. Note that this scheme is a generalization of RW13, because setting $n = n_c = n_k = 1$ yields RW13.

3.4 More efficient decryption

Generalizing the polynomial allows for an improved decryption efficiency. To understand why this yields a significant improvement, we briefly review the W11b scheme. We consider the keys and ciphertexts, which are of the form:

$$\begin{aligned}
\text{SK} = (K = h^{\alpha - rb}, K' = h^r, \{K_{\text{att}} = h^{r f_n(x_{\text{att}})}\}_{\text{att}\in\mathcal{S}}),\\
\text{CT} = (C = M \cdot e(g,h)^{\alpha s}, C' = g^s, \{C_j = B^{\lambda_j} F_n(\rho(j))^s\}_{j\in[n_1]}),
\end{aligned}$$

where $r, s \in \mathbb{Z}_p$ are randomly chosen integers, $B = g^b$ is a public key, α is the master key, and $\rho(j)$ is the j-th attribute of the policy of length n_1, and λ_j is a sharing of s with respect to the policy. To decrypt, one computes

$$C / e(C', K) \cdot \prod_{j\in\Upsilon} e(C_j, K')^{\varepsilon_j} / \prod_{j\in\Upsilon} e(C', K_{\rho(j)})^{\varepsilon_j},$$

where ε_j for $j \in \Upsilon \subseteq [n_1]$ are integers that allow us to reconstruct the secret s. Each such product of pairings can be computed more efficiently by using

the bilinearity property on the shared arguments, e.g., $\prod_j e(K', C_j)^{\varepsilon_j}$ can be computed more efficiently by first multiplying C_j and then taking a pairing:

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

This requires only two pairing operations instead of $2|\Upsilon|+1$. While decryption is very efficient, the drawback of W11b is that it is bounded in both the keys and ciphertexts. Because F_n embeds an n-degree polynomial, its randomized variant only provides sufficient randomness for $n+1$ attributes shared between the keys and ciphertexts. In contrast, RW13 uses an implicit 1-degree polynomial for the hash. To provide unboundedness, each instance of the hash—which in itself only provides sufficient randomness for one attribute—is randomized. As a result, both the keys and ciphertexts consist of components of the form $(g^{t_i}, F_1(x_{\text{att}})^{t_i})$, such that each randomizer part g^{t_i} needs to be paired with the part involving the hash during decryption. Hence, a linear number of pairing operations is required during decryption instead of a constant. By generalizing the 1-degree polynomial, we can use the same randomizer for multiple attributes. Thus, we can achieve a similar speed-up in decryption efficiency as W11b, whilst benefiting from the unboundedness of RW13.

Note that this also illustrates why it is important that the BB hash is used for both the keys and the ciphertexts. For example, in the GPSW06 large-universe scheme [28, §5], the randomness provided by the hash is used only for the ciphertexts. As a result, the keys require a fresh randomizer for each attribute, and therefore, decryption costs at least one pairing operation per attribute.

4 Our construction

We now present the complete description of our scheme in the selective security setting (see Appendix G for a fully secure version). In this scheme, we also introduce the mappings ι and τ, which map the attributes of the keys and ciphertexts, respectively, into arbitrary partitions of maximum sizes n_k and n_c.

Definition 6 (GLUE). *GLUE is defined as follows.*

- Setup(λ)*: On input the security parameter λ, the setup generates three groups $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ of prime order p with generators $g \in \mathbb{G}, h \in \mathbb{H}$, and chooses a pairing $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. It also defines the universe of attributes $\mathcal{U} = \mathbb{Z}_p$, chooses $n_k \in \mathbb{N}$ and $n_c \in \mathbb{N}$ as the maximum partition sizes of the keys and ciphertexts, respectively, and sets $n = n_k + n_c - 1$. It then generates random $\alpha, b, b_i, b'_{i'} \in_R \mathbb{Z}_p$ for all $i \in [0, n], i' \in [0, n_c - 1]$. It outputs $\text{MSK} = (\alpha, b, \{b_i, b'_{i'}\}_{i \in [0,n], i' \in [0,n_c-1]})$ as its master secret key and publishes the master public key as*

$$\text{MPK} = (g, h, A = e(g,h)^\alpha, B = g^b, \{B_i = g^{b_i}\}_{i \in \overline{[n]}}, \{B'_i = g^{b'_i}\}_{i \in \overline{[n_c-1]}}).$$

– KeyGen($\mathrm{MSK}, \mathcal{S}$)*: On input set of attributes* $\mathcal{S}$*, the key generation computes* $m = \left\lceil \frac{|\mathcal{S}|}{n_k} \right\rceil$*, defines* $\iota\colon \mathcal{S} \to [m]$ *such that* $|\iota^{-1}(l)| \leq n_k$ *for each* $l \in [m]$*, generates random integers* $r, r_1, ..., r_m \in_R \mathbb{Z}_p$*, and outputs the secret key as*

$$\mathrm{SK}_{\mathcal{S}} = (K = h^{\alpha - rb}, K' = h^r, \iota,$$
$$\{K_{1,\mathrm{att}} = h^{r_{\iota(\mathrm{att})}(\sum_{i=0}^{n} b_i x_{\mathrm{att}}^i) + r(\sum_{i=0}^{n_c - 1} b'_i x_{\mathrm{att}}^i)}\}_{\mathrm{att} \in \mathcal{S}}, \{K_{2,l} = h^{r_l}\}_{l \in [m]}).$$

– Encrypt($\mathrm{MPK}, \mathbb{A}, M$)*: A message* $M \in \mathbb{G}_T$ *is encrypted under policy* $\mathbb{A} = (\mathbf{A}, \rho)$ *with* $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ *and* $\rho\colon [n_1] \to \mathcal{U}$ *by computing* $m' = \max\left(\left\lceil \frac{n_1}{n_c} \right\rceil, \max_{j \in [n_1]} |\rho^{-1}(\rho(j))|\right)$ *and defining* $\tau\colon [n_1] \to [m']$ *such that* $|\tau^{-1}(l')| \leq n_c$ *for each* $l' \in [m']$ *and if* $j, j' \in [n_1]$ *with* $j \neq j'$ *such that* $\rho(j) = \rho(j')$*, then* $\tau(j) \neq \tau(j')$*, i.e., multiple occurrences of the same attribute are mapped to different partitions. (Note that this works because* m' *is defined to be at least as large as the maximum number of occurrences of each attribute.) It then generates random integers* $s, s_1, ..., s_{m'}, v_2, ..., v_{n_2} \in_R \mathbb{Z}_p$ *and outputs the ciphertext as*

$$\mathrm{CT}_{\mathbb{A}} = (C = M \cdot A^s, C' = g^s, \tau, \{C_{1,j} = B^{\lambda_j} \cdot \prod_{i=0}^{n_c - 1} (B'_i)^{s_{\tau(j)} \rho(j)^i},$$
$$C_{2,j} = \prod_{i=0}^{n} B_i^{s_{\tau(j)} \rho(j)^i}\}_{j \in [n_1]}, \{C_{3,l'} = g^{s_{l'}}\}_{l' \in [m']}),$$

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

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

$$C / \Big(e(C', K) \cdot \textstyle\prod_{j \in \Upsilon} \left(e(C_{1,j}, K') / e(C_{3,\tau(j)}, K_{1,\rho(j)}) \cdot e(C_{2,j}, K_{2,\iota(\rho(j))}) \right)^{\varepsilon_j} \Big).$$

This can be computed more efficiently as

$$C / \Big(e(C', K) \cdot e(\textstyle\prod_{j \in \Upsilon} C_{1,j}^{\varepsilon_j}, K') \cdot \Big(\prod_{l' \in [m']} e(C_{3,l'}, \prod_{j \in \Upsilon \cap \tau^{-1}(l')} K_{1,\rho(j)}^{-\varepsilon_j}) \cdot \prod_{l \in [m]} e(\prod_{j \in \Upsilon \cap \rho^{-1}(\iota^{-1}(l))} C_{2,j}^{-\varepsilon_j}, K_{2,l}) \Big) \Big),$$

which costs, on average, $2 + \left\lceil \frac{|\Upsilon|}{n_k} \right\rceil + \left\lceil \frac{|\Upsilon|}{n_c} \right\rceil$ *pairing operations.*

The scheme is correct, i.e., we have $C / e(C', K) = M \cdot e(g, h)^{\alpha s} \cdot e(g, h)^{-\alpha s + rsb} = M \cdot e(g, h)^{rsb}$ *and*

$$\prod_{j \in \Upsilon} \left(e(C_{1,j}, K') / e(C_{3,\tau(j)}, K_{1,\rho(j)}) \cdot e(C_{2,j}, K_{2,\iota(j)}) \right)^{\varepsilon_j}$$
$$= \prod_{j \in \Upsilon} (e(g, h)^{r \lambda_j b + r s_{\tau(j)} \sum_{i=0}^{n_c - 1} b'_i \rho(j)^i}$$

$$\cdot e(g,h)^{-r_{\iota(\rho(j))}s_{\tau(j)}(\sum_{i=0}^{n} b_i\rho(j)^i)-rs_{\tau(j)}\sum_{i=0}^{n_c-1} b'_i\rho(j)^i}$$
$$\cdot e(g,h)^{r_{\iota(\rho(j))}s_{\tau(j)}\sum_{i=0}^{n} b_i\rho(j)^i})^{\varepsilon_j}$$
$$= \prod_{j\in\Upsilon} e(g,h)^{r\varepsilon_j\lambda_j b} = e(g,h)^{rb\sum_{j\in\Upsilon}\varepsilon_j\lambda_j} = e(g,h)^{rsb},$$

which yields the plaintext, i.e., $M \cdot e(g,h)^{rsb}/e(g,h)^{rsb} = M$.

Unique representation of attributes. In the scheme, we assume that any attribute string $\text{att} \in \{0,1\}^*$ can be uniquely represented in $\mathbb{Z}_p$. In practice, this can be done by using a collision-resistant hash function $\mathcal{H}\colon \{0,1\}^* \to \mathbb{Z}_p$ [49].

4.1 The associated pair encoding scheme

To prove security, we define the pair encoding scheme associated with our scheme in Definition 6, for which we use the variables $n_c, n_k, n, \mathcal{S}, \iota, \rho, \tau$, $n_1, n_2, \lambda_i, m, m'$ from Definition 6, as follows.

Definition 7 (PES for GLUE).

- Param(par) $\to 2n_c + n_k + 3$. *Let* $\mathbf{b} = (b, b_0, ..., b_n, b'_0, ..., b'_{n_c-1})$, *where* $n = n_k + n_c - 1$.
- EncKey($\mathcal{S}$) $\to (\mathbf{r}, k', \{k_{1,\text{att}}\}_{\text{att}\in\mathcal{S}})$. *Let* $\mathbf{r} = (r, \{r_l\}_{l\in[m]})$, $k' = \alpha - rb$ *and* $k_{1,\text{att}} = r_{\iota(\text{att})}(\sum_{i=0}^{n} b_i x_{\text{att}}^i) + r(\sum_{i=0}^{n_c-1} b'_i x_{\text{att}}^i)$.
- EncCt($(\mathbf{A},\rho)$) $\to (\mathbf{s}, \hat{\mathbf{s}}, \{c_{1,j}, c_{2,j}\}_{j\in[n_1]})$. *Let* $\mathbf{s} = (s, \{s_{l'}\}_{l'\in[m']})$ *and* $\hat{\mathbf{s}} = (\hat{v}_2, ..., \hat{v}_{n_2})$, *and* $c_{1,j} = \mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal + s_{\tau(j)}\sum_{i=0}^{n_c-1} b'_i\rho(j)^i$ *and* $c_{2,j} = s_{\tau(j)}\sum_{i=0}^{n} b_i\rho(j)^i$.

In Section 5, we prove security of the PES.

Theorem 1. *The PES for GLUE in Definition 7 satisfies the symbolic property (Definition 5).*

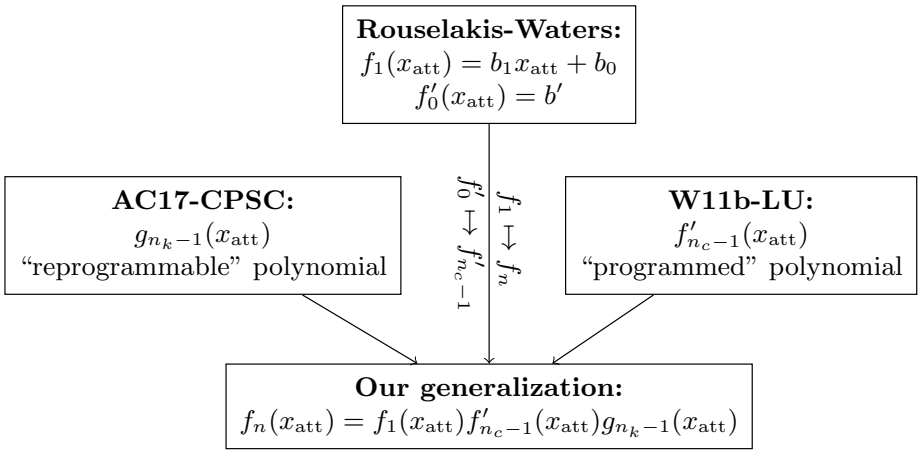
Therefore, instantiating the PES in the AC17 [3] framework yields a fully secure ABE scheme conform Definitions 2 and 3.

5 The security proof

While the construction of the scheme already provides some idea on why it may be secure, the proof requires some additional insights. First, we briefly review some important aspects in the Rouselakis-Waters proof, to gain some deeper understanding of the structure of the selective property proof. Then, we show how existing techniques can be combined to generalize the selective proof.

On a high level, the selective proof consists of the splitting of the n-degree polynomial, which provides randomness for the keys *and* ciphertexts, into a product of three polynomials f_1, f'_{n_c-1} and g_{n_k-1}. We use g_{n_k-1} for the keys,

Fig. 2: A high-level overview of the polynomials used in the combined proofs and our generalized selective proof.



and $f_1 f'_{n_c-1}$ for the challenge ciphertext. For the key polynomial g_{n_k-1}, we use Agrawal and Chase's [4] techniques. In their selective proof of the CP-ABE scheme with short ciphertexts, they embed a polynomial in the public keys such that this polynomial can be reprogrammed to some polynomial associated with the set of attributes of the key. We call such polynomials "reprogrammable". For the ciphertext polynomials f_1, f'_{n_c-1}, we use a combination of the proofs of RW13 [47] and W11b [53]. Roughly, in these proofs, they embed polynomials in the public keys, such that these polynomials are associated with the attributes that occur in the challenge access policy. We call such polynomials "programmed". These techniques ensure that the polynomials evaluate to the right values when the set and policy attributes are evaluated. Figure 2 depicts the relationship between the existing proofs and ours. A similar approach can be taken in the co-selective proof, by swapping the roles of the two polynomials.

5.1 The Rouselakis-Waters proof

We briefly review the Rouselakis-Waters selective security proof. They use the commonly used "program-and-cancel" technique [28,54], in which the challenge access policy $\mathbb{A} = (\mathbf{A}, \rho)$ is split in two disjoint subsets with respect to the set of attributes $\mathcal{S}$ associated with the queried key. One subset $\Upsilon = \{j \in [n_1] \mid \rho(j) \in \mathcal{S}\}$ contains all the rows of matrix $\mathbf{A}$ for which the corresponding attribute is in the set, while the other set $\overline{\Upsilon} = \mathcal{S} \setminus \Upsilon$ contains the rest of the rows. In the simulation of the key, it then uses the property that if $\mathcal{S}$ does not satisfy $\mathbb{A}$, there exists a vector $\mathbf{w} = (1, w_2, ..., w_{n_2}) \in \mathbb{Z}_p^{n_2}$ such that $\mathbf{A}_j \mathbf{w}^\intercal = 0$ for all $j \in \Upsilon$. Furthermore, they introduce the layering and individual randomness technique to ensure that the challenge ciphertext can be simulated. Roughly, they embed the attributes that occur in the policy in the public keys as well as their corre-

sponding row in the matrix in a layered fashion, using an individual randomness for each layer. During the key query and challenge phases, the randomizer embeds a single individual randomness to select the correct attribute layer, such that the key and ciphertext components can be simulated.

The selective symbolic security proof can be analogously structured, such that the substitutions ensure that all polynomials $k_{1,\text{att}}$, $c_{1,j}$ and $c_{2,j}$ evaluate to $\mathbf{0}$. In general, an attribute layer in the public keys is represented as a matrix $\mathbf{1}_{i,j}^{d_1 \times d_2} \in \mathbb{Z}_p^{d_1 \times d_2}$. Then, the appropriate attribute layer can be selected by the vector that represents the associated individual randomness, i.e., $\mathbf{1}_i^{d_1} \in \mathbb{Z}_p^{d_1}$. Multiplying this vector with the matrix yields $\overline{\mathbf{1}}_j^{d_2} \in \mathbb{Z}_p^{d_2}$. Effectively, only one remaining entry needs to evaluate to $\mathbf{0}$. This is done by embedding the policy in the public keys in a certain way. Specifically, in $c_{1,j} = \mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal + s_j b'$, where $\mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal = \sum_{k \in [n_2]} A_{j,k} v_k$ (where $v_1 = sb$) and $s_j b'$ are supposed to cancel out one another. This can be ensured by embedding all rows $\mathbf{A}_j$ of the policy in b', and using individual randomness (represented as a vector) to select the appropriate row. More concretely, b' could be substituted by $-\sum_{j \in [n_1], k \in [n_2]} A_{j,k} \mathbf{1}_{j,k}^{d_1 \times d_2}$ and s_j by $\mathbf{1}_j^{d_1}$ such that $s_j b' = -\sum_{k \in [n_2]} A_{j,k} \overline{\mathbf{1}}_k^{d_2}$. Then, if v_k is substituted by $\overline{\mathbf{1}}_{(0,k)}^{d_2}$, s by $\mathbf{1}_0^{d_1}$ and b by $\mathbf{1}_{0,(0,1)}^{d_1 \times d_2}$, then $\mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal = \sum_{k \in [n_2]} A_{j,k} \overline{\mathbf{1}}_{(0,k)}^{d_2}$. Similarly, $c_{2,j}$ evaluates to $\mathbf{0}$ by defining b_0 and b_1 such that for each attribute layer associated with row j, the 1-degree polynomial $F_{1,j}(x) = x - \rho(j)$ is embedded. The individual randomness ensures that this polynomial is selected with s_j. To ensure that the key $k_{1,\text{att}} = r_{\text{att}} f_1(x_{\text{att}}) + r b'$ evaluates to $\mathbf{0}$, we embed the vector $\mathbf{w}$ in r and r_{att}, which ensures that all attribute layers that are also in the set $\mathcal{S}$ go to $\mathbf{0}$. For those attribute layers that are not in the set $\mathcal{S}$, i.e., $\overline{\Upsilon}$, we ensure that layers $F_{1,j}(x_{\text{att}}) \mathbf{A}_j \mathbf{w}^\intercal$ in $r_{\text{att}} f_1(x_{\text{att}})$ cancel out layers $\mathbf{A}_j \mathbf{w}^\intercal$ in $r b'$. Roughly, this is done by embedding $\frac{1}{F_{1,j}(x_{\text{att}})}$ in r_{att} for all $j \in \overline{\Upsilon}$, such that the $\mathbf{A}_j \mathbf{w}^\intercal$ in the two summands cancel out one another.

5.2 Generalizing the Rouselakis-Waters proof

We generalize the Rouselakis-Waters proof by layering the policy embedded in the public keys in a partition-wise fashion instead of attribute-wise. In this way, the ciphertext-specific variable $s_{l'}$, which is used for all attributes in the same partition, can select all attributes associated within the l'-th partition. As such, in the computation of $c_{1,j}$ and $c_{2,j}$, $s_{\tau(j)} f'_{n_c-1}(\rho(j))$ needs to cancel out $\mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal$ and $s_{\tau(j)} f_n(\rho(j))$ needs to go to $\mathbf{0}$. To this end, we need to substitute f'_{n_c-1} in such a way that it outputs exactly $-\sum_{k \in [n_2]} A_{j,k} \overline{\mathbf{1}}_{(0,k)}^{d_2}$ when $s_{\tau(j)} f'_{n_c-1}(\rho(j))$ is computed. Similarly, the key-specific variable r_l needs to be constructed such that $k_{1,\text{att}}$ goes to $\mathbf{0}$, which happens when $r_{\iota(\text{att})} f_n(x_{\text{att}})$ cancels out $r f'_{n_c-1}(x_{\text{att}})$.

To accomplish this, we define f_n and f'_{n_c-1} as mentioned before, i.e., $f_n(x_{\text{att}}) = f_1(x_{\text{att}}) f'_{n_c-1}(x_{\text{att}}) g_{n_k-1}(x_{\text{att}})$. Roughly, we substitute f_1 in the same way as in the Rouselakis-Waters proof, while we use the polynomials f'_{n_c-1} and g_{n_k-1} to ensure that $c_{1,j}$ and $c_{2,j}$, and $k_{1,\text{att}}$ evaluate to $\mathbf{0}$, respectively. Because we are

given the challenge access structure a priori, i.e., as input to EncB, we can program these as required in the substitutions of the polynomials f_1 and f'_{n_c-1} in the public keys. Concretely, we substitute $b_0, ..., b_n$ such that

$$f_n(x_{\text{att}}) \ : \sum_{j\in[n_1],k\in[n_2]} A_{j,k} F_{n,j,k}(x_{\text{att}})$$
$$= \sum_{j\in[n_1],k\in[n_2]} A_{j,k} \underbrace{F_{1,j}(x_{\text{att}}) F'_{n_c-1,j}(x_{\text{att}}) \hat{G}_{n_k-1,j,k}(x_{\text{att}})}_{F_{n,j,k}(x_{\text{att}})},$$

where $F_{1,j}(x_{\text{att}}) = (x_{\text{att}} - \rho(j))$ and

$$F'_{n_c-1,j}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} d'_{i,j} x_{\text{att}}^i = \prod_{j'\in\chi_j\setminus\{j\}} \frac{x_{\text{att}} - \rho(j')}{\rho(j) - \rho(j')},$$

with $\chi_j = \{j' \in [n_1] \mid \tau(j') = \tau(j)\}$. We refer to these polynomials as the "programmed" polynomials. These ensure that $F_{n,j}(\rho(j')) = 0$ for all $j' \in \chi_j$, $F'_{n_c-1,j}(\rho(j)) = 1$ and $F'_{n_c-1,j'}(\rho(j)) = 0$ for all $j' \in \chi_j \setminus \{j\}$. Then, $c_{1,j}$ and $c_{2,j}$ evaluate to $\mathbf{0}$, if we substitute

$$f'_{n_c-1}(x_{\text{att}}) \ : \sum_{j\in[n_1],k\in[n_2]} A_{j,k} F'_{n_c-1,j}(x_{\text{att}}) \mathbf{1}^{d_1\times d_2}_{(1,\tau(j)),(0,k)}.$$

In contrast, the set of attributes associated with a key is given after the public keys have been established, i.e., as input to EncR, so we need to somehow achieve that we can program the polynomial $\hat{G}_{n_k-1,j}$ after the public keys are generated. We do this by setting

$$\hat{G}_{n_k-1,j,k}(x_{\text{att}}) = \sum_{i=0}^{n_k-1} \mathbf{1}^{d_1\times d_2}_{(1,\tau(j)),(1,i,j,k)} x_{\text{att}}^i,$$

such that $\hat{G}_{n_k-1,j,k}$ constitutes a "reprogrammable" polynomial. It can be reprogrammed by ensuring that r_l consists of the coefficients $u_{i,j,l}$ of some target polynomial(s), i.e., by multiplying

$$\left(\sum_{i=0}^{n_k-1} \mathbf{1}^{d_1\times d_2}_{(1,\tau(j)),(1,i,j,k)} x_{\text{att}}^i\right)\left(\sum_{i=0}^{n_k-1} u_{i,j,l} \overline{\mathbf{1}}^{d_2}_{(1,i,j,k)}\right)$$
$$= \sum_{i=0}^{n_k-1} u_{i,j,l} \mathbf{1}^{d_1}_{(1,\tau(j))} x_{\text{att}}^i.$$

We use this to "reprogram" polynomial $\hat{G}_{n_k-1,j,k}(x_{\text{att}})$ for all $j \in \overline{\Upsilon}$, which is well-defined, because $\rho(j) \notin \mathcal{S}$. This then yields $F'_{n_c-1,j}(x_{\text{att}})$ and cancels out the $F'_{n_c-1,j}(x_{\text{att}})$ in $rf'_{n_c-1}(x_{\text{att}})$ part in $k_{1,\text{att}}$ for all $j \in \overline{\Upsilon}$. Note that, like in Rouselakis-Waters, we have $\mathbf{A}_j \mathbf{w}^\intercal = 0$ for all $j \in \Upsilon$, so those layers

automatically go to $\mathbf{0}$ in the computation of $k_{1,\mathrm{att}}$. Hence, for each partition $\Psi_l = \{\mathrm{att} \in \mathcal{S} \mid \iota(\mathrm{att}) = l\}$ with $l \in [m]$, we define the polynomial

$$\begin{aligned} G_{n_k-1,j,l}(x_{\mathrm{att}}) &= \sum_{i=0}^{n_k-1} u_{i,j,l} x_{\mathrm{att}}^i \\ &= \sum_{\mathrm{att}' \in \Psi_l} \frac{1}{F_{1,j}(x_{\mathrm{att}'})} \prod_{\mathrm{att}'' \in \Psi_l \setminus \{\mathrm{att}'\}} \frac{x_{\mathrm{att}} - x_{\mathrm{att}''}}{x_{\mathrm{att}'} - x_{\mathrm{att}''}}, \end{aligned}$$

for each $j \in \overline{\Upsilon}$, such that $G_{n_k-1,j,l}(x_{\mathrm{att}}) = \frac{1}{F_{1,j}(x_{\mathrm{att}})}$ for all $\mathrm{att} \in \Psi_l$.

Putting it all together, we substitute b_0,...,b_n with coefficients such that the polynomial f_n is substituted by

$$\sum_{j \in [n_1], k \in [n_2]} A_{j,k} F_{n,j,k}(x_{\mathrm{att}}) = \sum_{j \in [n_1], k \in [n_2]} A_{j,k} \sum_{i=0}^{n} d_{i,j,k} x_{\mathrm{att}}^i,$$

where

$$d_{i,j,k} = \sum_{i' \in \overline{[n_k-1]}, i'' \in \overline{[n_c-1]} : i'+i''=i} d'_{i',j} \mathbf{1}^{d_1 \times d_2}_{(1,\tau(j)),(1,i'',j,k)}.$$

5.3 The selective symbolic property

We prove the selective symbolic property, using m, m', τ, ι as in Section 4 and $F_{n,j,k}, d_{i,j,k}, F'_{n_c-1,j}, d'_{i,j}, G_{n_k-1,j,l}, u_{i,j,l}$ and χ_j as in Section 5.2. For simplicity of notation, we write the second index of $\mathbf{1}^{d_1 \times d_2}$ and the index of $\overline{\mathbf{1}}^{d_2}$ as a tuple $(1, i, j, k)$ (with $i \in \overline{[n_k]}, j \in [n_1], k \in [n_2]$) such that it represents a unique integer in $[n_2 + 1, (((n_k + 1)n_1 + 1)n_2)]$. Note that we use $(0, k)$ to indicate the first n_2 columns, which are associated with only k and not (i, j). For the first index of $\mathbf{1}^{d_1 \times d_2}$ and the index of $\mathbf{1}^{d_1}$, we start counting at 0. Note that, therefore, $d_1 = n_2 + 1$ and $d_2 = ((n_k + 1)n_1 + 1)n_2$. The substitutions are, for all $i \in \overline{[n]}, i' \in \overline{[n_k]}, l \in [m], l' \in [m'], k \in [2, n_2]$:

$$\begin{gathered} b \;:\; \mathbf{1}^{d_1 \times d_2}_{0,(0,1)}, \quad b_i \;:\; \sum_{j \in [n_1], k \in [n_2]} A_{j,k} d_{i,j,k} \\ b'_{i'} \;:\; \sum_{j \in [n_1], k \in [n_2]} A_{j,k} d'_{i',j} \mathbf{1}^{d_1 \times d_2}_{(1,\tau(j)),(0,k)}, \quad s \;:\; \mathbf{1}^{d_1}_0 \\ s_{l'} \;:\; -\mathbf{1}^{d_1}_{(1,l')}, \quad \alpha \;:\; \mathbf{1}^{d_1}_0, \quad \hat{v}_k \;:\; \overline{\mathbf{1}}^{d_2}_{(0,k)}, \quad r \;:\; \sum_{k' \in [n_2]} w_{k'} \overline{\mathbf{1}}^{d_2}_{(0,k')} \\ r_l \;:\; - \sum_{i' \in \overline{[n_k-1]}, j' \in \overline{\Upsilon}, k' \in [n_2]} w_{k'} u_{i',j',l} \overline{\mathbf{1}}^{d_2}_{(1,i',j',k')}. \end{gathered}$$

Then, $c_{1,j}, c_{2,j}, k'$ and $k_{1,\mathrm{att}}$ indeed go to $\mathbf{0}$ (see Appendix C.1).

5.4 Co-selective symbolic property

We prove that the co-selective symbolic property also holds. For this proof, the roles of the reprogrammable and the programmed polynomial are reversed, because we are allowed to use an attribute set $\mathcal{S}$ in the programming of the public keys and secret keys, and the policy $\mathbb{A}$ only for the ciphertext. Similarly as in the selective case, we use the Rouselakis-Waters proof as inspiration for the structure of the proof. In particular, we use the selective security proof of the KP-ABE variant of the Rouselakis-Waters scheme (which is analogous to the co-selective proof of the CP-ABE variant).

In this proof, we substitute the polynomials with:

$$f_n(x_{\text{att}}) \ : \ \sum_{l\in[m]} \left(\underbrace{\hat{F}_{n_c-1,1,l}(x_{\text{att}}) G_{n_k,l}(x_{\text{att}})}_{G_{n,l}(x_{\text{att}})} - \hat{F}_{n_c-1,2,l}(x_{\text{att}}) \right),$$

$$f'_{n_c-1}(x_{\text{att}}) \ : \ \hat{F}_{n_c-1,2,0}(x_{\text{att}}),$$

where we define $G_{n,l}(x_{\text{att}}) = \sum_{i=0}^{n} \tilde{u}_{i,l} x_{\text{att}}^i$, and

$$G_{n_k,l}(x_{\text{att}}) = \sum_{i=0}^{n_k} u_{i,l} x_{\text{att}}^i = \prod_{\text{att}'\in\Psi_l} (x_{\text{att}} - x_{\text{att}'})$$

is a programmed polynomial with $G_{n_k,l}(x_{\text{att}}) = 0$ for $\text{att} \in \Psi_l$, and

$$\hat{F}_{n_c-1,1,l}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} \mathbf{1}_{(1,i,l),l}^{d_1\times d_2} \text{ and } \hat{F}_{n_c-1,2,l}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} \mathbf{1}_{(2,i),l}^{d_1\times d_2}$$

are the reprogrammable polynomials, to be reprogrammed to

$$F_{n_c-1,1,j,l}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} \tilde{d}_{i,j,l} x_{\text{att}}^i = \frac{1}{G_{n_k,l}(\rho(j))} F'_{n_c-1,j}(x_{\text{att}}),$$

$$F'_{n_c-1,j}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} d'_{i,j} x_{\text{att}}^i,$$

respectively, for $j \in \overline{\Upsilon}$. Note that $F_{n_c-1,1,j,l}(\rho(j)) = \frac{1}{G_{n_k,l}(\rho(j))}$ if $j \in \overline{\Upsilon}$ and $F_{n_c-1,1,j',l}(\rho(j)) = 0$ for $j' \in \chi_j$. Concretely, we have

$$\tilde{u}_{i,l} = \sum_{i'\in\overline{[n_c-1]}, i''\in\overline{[n_k]}: i'+i''=i} u_{i'',l} \mathbf{1}_{(1,i',l),l}^{d_1\times d_2}.$$

Then, for $i \in \overline{[n_c-1]}, i' \in [n_c, n], l \in [m], l' \in [m'], k \in [n_2]$ we make the following substitutions:

$$b \ : \ \mathbf{1}_{0,0}^{d_1\times d_2}, \quad b_i \ : \ \sum_{l\in[m]} \left(\tilde{u}_{i,l} - \mathbf{1}_{(2,i),l}^{d_1\times d_2} \right), \quad b_{i'} \ : \ \sum_{l\in[m]} \tilde{u}_{i,l}$$

$$b'_i \;:\; \mathbf{1}^{d_1 \times d_2}_{(2,i),0}, \quad \alpha \;:\; \mathbf{1}^{d_1}_0, \quad \hat{v}_k \;:\; w_k \overline{\mathbf{1}}^{d_2}_0, \quad r \;:\; \overline{\mathbf{1}}^{d_2}_0, \quad r_l \;:\; \overline{\mathbf{1}}^{d_2}_l$$
$$s \;:\; \mathbf{1}^{d_1}_0, \quad s_{l'} \;:\; - \sum_{i \in \overline{[n_c-1]}, j \in \hat{\chi}_{l'} \cap \overline{\Upsilon}, k \in [n_2]} A_{j,k} w_k \left(d'_{i,j} \mathbf{1}^{d_1}_{(2,i)} + \tilde{d}_{i,j} \right),$$

where $\tilde{d}_{i,j} = \sum_{l \in [m]} \tilde{d}_{i,j,l} \mathbf{1}^{d_1}_{(1,i,l)}$, $d_1 = n_c(m+1)+1$ and $d_2 = m+1$. For simplicity, we use tuple notations $(1,i,l)$ and $(2,i)$ for the first index of $\mathbf{1}^{d_1 \times d_2}$ and the index of $\mathbf{1}^{d_1}$ for all $i \in \overline{[n_c-1]}, l \in [m]$, which map injectively into the intervals $[2, n_c m + 1]$ and $[n_c m + 2, d_1]$, respectively, and index 0 maps to the first row. Then, $c_{1,j}, c_{2,j}, k'$ and $k_{1,\mathrm{att}}$ indeed go to $\mathbf{0}$ (see Appendix C.2).

6 Performance analysis

We analyze the efficiency of our schemes. An important aspect in this analysis are the parameters n_k and n_c, which are chosen during the setup (e.g., by a practitioner). On a high level, the key generation, encryption and decryption of the selectively secure version of GLUE incur the following costs:

- KeyGen: $2 + |\mathcal{S}| + \left\lceil \frac{|\mathcal{S}|}{n_k} \right\rceil$ exponentiations in $\mathbb{H}$;
- Encrypt: 1 exponentiation in $\mathbb{G}_T$, $1 + \left\lceil \frac{n_1}{n_c} \right\rceil$ exponentiations, n_1 multi-exponentiations with $n_c + 1$ bases and n_1 multi-exponentiations with $n_k + n_c$ bases in $\mathbb{G}$;
- Decrypt: roughly $2 + \left\lceil \frac{|\Upsilon|}{n_k} \right\rceil + \left\lceil \frac{|\Upsilon|}{n_c} \right\rceil$ pairing operations.

The efficiency of these algorithms depends on the one hand on the efficiency of these operations, and on the other hand on the choices of n_k and n_c. By analyzing these rough costs from a mathematical point of view, the trade-off between the encryption and decryption efficiency is optimal when $n_k = n_c$ (which follows from the arithmetic mean-harmonic mean inequality). However, when the set of attributes $\mathcal{S}$ is large, and n_k is small, it may occur that all matching attributes are in different partitions. As such, choosing n_k to be larger, e.g., $n_k = 10$, ensures that the matching attributes are in the same key partitions with a large probability, and therefore the actual number of pairing operations is higher. In general, it holds that, the larger the partition sizes, the fewer pairing operations are needed during decryption. Unfortunately, the drawback is that encryption becomes more expensive, meaning that we may want to use the online/offline version of the scheme in practice. In Appendix K, we give more details on how a suitable partition size may be chosen. For our analysis, we consider three parameter settings: $(n_k, n_c) \in \{(3,3), (5,5), (10,5)\}$. Furthermore, for the variant that supports OSW-type negations, we consider $|\mathcal{S}| \in \{1, 5\}$.

On the comparability of the schemes. For a fair comparision, we optimize all the schemes in the same way when instantiating the schemes in the asymmetric setting [46]. Specifically, we optimize the decryption and encryption

efficiency. For the analysis of the RW13 [47], HW14 [33], Att19-I-CP and Att19-I-CP-OO [11] schemes, we have used the performance analysis of our associated schemes for $n_k = n_c = 1$ (which have the same encodings). We also compare our monotone schemes with AHM+16 [12] and ABGW17 [7] (see Appendix H for our instantiations of the schemes). In particular, we have placed all schemes in the fully-secure and prime-order setting, so they yield the most comparable results. To place the costs based on our theoretical analyses in the fully-secure setting, we multiply the costs for each element and operation in $\mathbb{G}$ and $\mathbb{H}$ by a factor 2. This overhead corresponds to the most efficient instantiation of the schemes in the AC17 framework [3]. For all schemes, we also assume that the input access policies are Boolean formulas, so that for decryption, it is ensured that $\varepsilon_j \in \{0, 1\}$ [39].

Estimates based on benchmarks in RELIC. We estimate the computational costs of the schemes by obtaining benchmarks of various algorithms and extrapolating the results by analyzing the descriptions of the schemes. We analyze the efficiency in this way for two reasons. First, it allows us to analyze the efficiency of many scheme configurations without having to implement each one, which is a cumbersome and error-prone effort. Second, it allows us to compare the schemes more accurately and more fairly. Currently, the simplest and most popular way [47,2,52] to benchmark schemes is by using Charm [5]. However, Charm only supports curves that do not provide sufficient security anymore, and de la Piedra et al. [46] show that benchmarking the schemes on these curves yields inaccurate and unfair comparisons. To compare the schemes more accurately and fairly, we estimate[4] the costs of the schemes by applying their approaches [46]. In particular, we have run benchmarks in RELIC [8], a cryptographic library for efficient implementations of pairing-based cryptography on state-of-the-art elliptic curves. This library has implementations for exponentiations, including fixed-base and multi-base variants. In fixed-base exponentiation, the base g in g^x is fixed after setup, and as such, a precomputation table can be made to speed up the computation [21]. In a multi-base exponentiation, the product of multiple exponentiations, e.g., $g_1^{x_1} g_2^{x_2}$, is computed more efficiently [43]. We have run these benchmarks on a 1.6 GHz Intel i5-8250U processor for the BLS12-446 curve [16], which provides approximately 132-134 bits of security [31,30]. These benchmarks can be found in Appendix I and are used in our analysis.

Comparison. Tables 3a and 3b show the performances of all unbounded schemes using a BB hash that support MSPs and NMSPs. The tables illustrate that the decryption algorithms of our regular schemes are significantly faster than the established schemes. While the encryption costs increase compared to the other schemes, our online/offline versions also provide a solution in this regard, incurring minimal online costs. This comes with a slight trade-off in the ciphertext size

[4] Although approximated theoretically, we expect our estimates to be close to the costs of actual implementations (Appendix J).

Table 3: Rough estimates of the storage costs of the secret keys and the ciphertexts in kilobytes (KB), where 1 KB = 1024 bytes, and the (online) computational costs incurred by the key generation, encryption and decryption algorithms of $\text{GLUE}_{(n_k,n_c)}$ (and its online/offline (suffixed with "OO") and OSW-non-monotone (suffixed with "N") variants) and the other unbounded CP-ABE schemes, expressed in milliseconds (ms), for 10 and 100 attributes. Note that the offline key generation and encryption costs of each online/offline scheme are equal to the key generation and encryption costs of its regular version. The code used to generate these benchmarks is available at https://github.com/mtcvenema/glue.

		Storage costs					**Computational costs**					
			SK		CT		KeyGen		Encrypt		Decrypt	
	Scheme	|MPK|	10	100	10	100	10	100	10	100	10	100
Regular	RW13 [47]	1.42	4.86	44.58	4.05	33.58	26.0	238.7	32.9	305.9	46.2	375.2
	AHM+16 [12] ($n_k = 2$)	1.75	5.3	45.02	6.45	55.67	16.5	122.9	40.8	368.3	43.7	317.4
	ABGW17 [7]	1.42	2.65	22.51	3.94	33.47	14.2	120.5	32.3	305.2	27.9	192.4
	$\text{GLUE}_{(3,3)}$	2.08	3.53	30.02	3.39	26.36	18.9	160.7	59.8	571.4	24.3	133.9
	$\text{GLUE}_{(5,5)}$	2.74	3.09	26.93	3.17	24.83	16.5	144.2	82.3	800.4	17.0	82.8
	$\text{GLUE}_{(10,5)}$	3.28	2.87	24.72	3.17	24.83	15.4	132.3	102.1	998.4	15.1	64.5
O/O	HW14 [33]	1.42	5.23	48.29	4.79	41.0	0	0	0	0	51.5	416.2
	$\text{GLUE}_{(3,3)}$	2.08	3.9	33.73	5.62	48.62	0	0	0	0	33.6	202.6
	$\text{GLUE}_{(5,5)}$	2.74	3.46	30.64	6.88	61.94	0	0	0	0	27.6	157.5
	$\text{GLUE}_{(10,5)}$	3.28	3.24	28.43	8.74	80.49	0	0	0	0	24.2	123.4

(a) Schemes supporting MSPs only.

		Storage costs					**Computational costs**									
			SK		CT		KeyGen		Encrypt		Decrypt					
	Scheme	|MPK|	10	100	10	100	10	100	10	100	10	100	10	100	10	100
Regular	Att19-I-CP [11]	1.42	10.89	100.01	6.37	55.76	59.0	541.8	66.6	637.5	51.7	380.7	55.3	367.9	216.2	1779.0
	GLUE-$\text{N}_{(3,3)}$	2.08	7.59	63.66	5.04	41.19	44.8	385.9	90.0	865.1	29.8	139.4	62.3	374.9	109.5	745.5
	GLUE-$\text{N}_{(5,5)}$	2.74	6.49	55.95	4.6	38.1	40.1	352.8	111.4	1086.0	22.4	88.3	55.3	367.9	55.3	382.5
	GLUE-$\text{N}_{(10,5)}$	3.28	5.94	50.44	4.6	38.1	37.7	329.2	131.2	1284.0	20.6	70.0	78.5	599.4	78.5	614.1
O/O	Att19-I-CP-OO	1.42	12.01	111.15	7.11	63.18	0	0	0	0	61.0	461.3	64.6	448.5	225.5	1859.6
	GLUE-$\text{N}_{(3,3)}$	2.08	8.7	74.79	7.27	63.46	0	0	0	0	46.7	275.2	79.3	510.6	126.4	881.3
	GLUE-$\text{N}_{(5,5)}$	2.74	7.6	67.08	8.31	75.21	0	0	0	0	41.8	235.9	74.7	515.5	74.7	530.1
	GLUE-$\text{N}_{(10,5)}$	3.28	7.05	61.58	10.17	93.77	0	0	0	0	36.8	185.9	94.7	715.4	94.7	730.0

(b) Schemes supporting OSW-type negations. The decryption costs are for non-negated, and negated policies with $|\mathcal{S}| \in \{1, 5\}$, respectively.

and the decryption efficiency compared to the regular version, but overall, our online/offline schemes outperform the established schemes in all algorithms. Importantly, the decryption of our schemes supporting negations with parameters $n_k = n_c = 5$ outperforms the only other unbounded OSW-type non-monotone scheme. Importantly, decryption is faster by a factor 4 for non-negated attributes, and faster by a factor 4-5 for negated attributes and $|\mathcal{S}| = 5$, bringing down the costs from almost two seconds to 382 ms. As a result, our schemes could provide

a more attractive building block for OSWOT-type non-monotone schemes, as they support more efficient decryption algorithms for negated and non-negated attributes, and for small and large sets of attributes for each label. Furthermore, owing to the online/offline extensions, the key generation and encryption algorithms do not need to suffer from heavy online computations. Instead, encrypting users need to store only 3.17-10.17 kilobytes per one intermediate ciphertext of the first type and sufficient of the second type for ten attributes (depending on the instantiation). This means that, with just a megabyte of space, a user can store at least 100 intermediate ciphertexts for a total of 1000 attributes. For computing devices such as computers and smartphones, which have an abundance of storage space nowadays, this is a more than acceptable trade-off. Similarly, key generation authorities can store intermediate keys for at least 286 users and 2860 attributes with just a megabyte of space. Thus, with gigabytes, an authority can precompute keys for hundreds of thousands of users and millions of attributes.

7 Applying multiple instantiations of GLUE in practice

The flexible efficiency trade-offs that GLUE provides can be exploited in practice. In particular, practitioners can choose one suitable instantiation of GLUE, or choose multiple instantiations of GLUE to support different computational devices. Interestingly, by using the direct sum with parameter reuse transformation of Attrapadung [11], GLUE would be able to support multiple instances of itself simultaneously, such that the size of the master public key is upper-bounded in the maximum size of the public keys of all instances. This may be useful in settings in which the devices have varying computational resources. For instance, in the WLAN use case considered by ETSI [26], the decryption devices may be any mobile device in a network, including more constrained devices such as smartwatches. For those devices, it is more beneficial to use a scheme with fast decryption, e.g., $\text{GLUE}_{(5,5)}$, while for faster devices, it is sufficient to employ a scheme with slower decryption, e.g., RW13. In WLAN systems, the access point sends an encrypted e.g., WPA2-PSK key to the connecting device, which can decrypt it if is satisfies the policy. Because this exchange is interactive, the connecting device and access point could first negotiate on the particular instance of GLUE for which the connecting device has a secret key before encrypting the WPA2-PSK key. In non-interactive systems, e.g., cloud settings [26], it may be more desirable to use multiple instances in parallel. Powerful devices could, for instance, use multiple instances to support less powerful devices that only use the more efficient instances. For example, powerful decryption devices could have keys for both $\text{GLUE}_{(5,5)}$ and RW13, while less powerful encryption devices use RW13 or an online/offline variant of GLUE to encrypt.

8 Future work

For future work, it would be interesting to investigate the following. First, we have proven our scheme secure in the AC17 framework, which yields full secu-

rity under a q-type assumption. Although frameworks exist that prove security generically under static assumptions [9,22,10], these use a strong security notion called the master-key hiding property. Like other unbounded ABE using a BB hash, ours does not satisfy this property [9]. To achieve such strong notions of security, more intricate proof techniques need to be devised, such as [23]. Second, we have analyzed the efficiency of the schemes on the BLS12-446 curve. Presumably, the encryption and decryption costs can improve if curves such as KSS16-339 [35] are used, which provide faster arithmetic in $\mathbb{G}$ and provide more efficient products of pairing operations [25]. GLUE (and RW13) may also benefit from fixed-base multi-base exponentiations [43], which RELIC does not support. Finally, while we have given the first steps towards realizing more efficient schemes supporting OSWOT-type negations (Appendix F), we have not explicitly specified these schemes. Our analysis in Section 6 indicates that any such schemes would benefit from the efficiency of our schemes, including those supporting OSW-type negations (see Appendix L for more details).

9 Conclusion

We have proposed GLUE, a new unbounded large-universe scheme with flexible efficiency trade-off. This scheme is a generalization of RW13 [47] and W11b [53], in that it supports polynomials of any degree for the Boneh-Boyen hash. To optimally use the randomness provided by the hash, we use the partitioning approach (previously also used by AHM+16 [12]), splitting the sets of attributes and the policies in partitions of maximum sizes n_k and n_c, respectively. This allows for a decreased number of pairing operations required during decryption compared to RW13 (and related variants). Roughly, the pairing costs decrease by a factor $n_k = n_c$ (if chosen to be equal). Along the way, we have also introduced new proof techniques. These ensure that the randomness provided by the BB hash can be used for both the keys and ciphertexts in the unbounded setting. Finally, we have shown that our schemes indeed outperform existing schemes using a BB hash in the decryption, and notably, all schemes supporting OSW-type negations. Because our non-monotone schemes are unbounded and faster than 1.2 seconds in all algorithms on a laptop, even for large policies and sets, they are more suitable for practice than existing non-monotone schemes.

References

1. Agrawal, S., Chase, M.: A study of pair encodings: Predicate encryption in prime order groups. In: TCC. pp. 259–288. Springer (2016)
2. Agrawal, S., Chase, M.: FAME: fast attribute-based message encryption. In: CCS. pp. 665–682. ACM (2017)
3. Agrawal, S., Chase, M.: Simplifying design and analysis of complex predicate encryption schemes. In: EUROCRYPT. pp. 627–656. Springer (2017)
4. Agrawal, S., Chase, M.: Simplifying design and analysis of complex predicate encryption schemes. Cryptology ePrint Archive, Report 2017/233 (2017)

5. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptogr. Eng. **3**(2), 111–128 (2013)
6. Ambrona, M.: Generic negation of pair encodings. In: Garay, J.A. (ed.) PKC. LNCS, vol. 12711, pp. 120–146. Springer (2021)
7. Ambrona, M., Barthe, G., Gay, R., Wee, H.: Attribute-based encryption in the generic group model: Automated proofs and new constructions. In: CCS. pp. 647–664. ACM (2017)
8. Aranha, D.F., Gouvêa, C.P.L., Markmann, T., Wahby, R.S., Liao, K.: RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic (2020)
9. Attrapadung, N.: Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In: EUROCRYPT. pp. 557–577. Springer (2014)
10. Attrapadung, N.: Dual system encryption framework in prime-order groups via computational pair encodings. In: ASIACRYPT. pp. 591–623. Springer (2016)
11. Attrapadung, N.: Unbounded dynamic predicate compositions in attribute-based encryption. In: EUROCRYPT. pp. 34–67. Springer (2019)
12. Attrapadung, N., Hanaoka, G., Matsumoto, T., Teruya, T., Yamada, S.: Attribute based encryption with direct efficiency tradeoff. In: ACNS. pp. 249–266. Springer (2016)
13. Attrapadung, N., Libert, B.: Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In: PKC. pp. 384–402. Springer (2010)
14. Attrapadung, N., Libert, B., de Panafieu, E.: Expressive key-policy attribute-based encryption with constant-size ciphertexts. In: PKC. pp. 90–108. Springer (2011)
15. Attrapadung, N., Tomida, J.: Unbounded dynamic predicate compositions in ABE from standard assumptions. In: ASIACRYPT. pp. 405–436. Springer (2020)
16. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: SCN. pp. 257–267. Springer (2002)
17. Beimel, A.: Secure Schemes for Secret Sharing and Key Distribution. Phd thesis, Ben Gurion University (1996)
18. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: S&P. pp. 321–334. IEEE (2007)
19. Boneh, D., Boyen, X.: Efficient selective-id secure identity-based encryption without random oracles. In: EUROCRYPT. pp. 223–238. Springer (2004)
20. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT. pp. 440–456. Springer (2005)
21. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation. In: EUROCRYPT. pp. 200–207. Springer (1992)
22. Chen, J., Gay, R., Wee, H.: Improved dual system ABE in prime-order groups via predicate encodings. In: EUROCRYPT. pp. 595–624. Springer (2015)
23. Chen, J., Gong, J., Kowalczyk, L., Wee, H.: Unbounded ABE via bilinear entropy expansion, revisited. In: EUROCRYPT. pp. 503–534. Springer (2018)
24. Chen, J., Wee, H.: Dual system groups and its applications — compact hibe and more. Cryptology ePrint Archive, Report 2014/265 (2014)
25. Clarisse, R., Duquesne, S., Sanders, O.: Curves with fast computations in the first pairing group. In: CANS. pp. 280–298. Springer (2020)
26. ETSI: ETSI TS 103 458 (V1.1.1). Technical specification, European Telecommunications Standards Institute (ETSI) (2018)
27. ETSI: ETSI TS 103 532 (V1.1.1). Technical specification, European Telecommunications Standards Institute (ETSI) (2018)

28. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: CCS. ACM (2006)
29. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Report 2006/309 (2006)
30. Guillevic, A.: A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In: PKC. pp. 535–564. Springer (2020)
31. Guillevic, A., Singh, S.: On the alpha value of polynomials in the tower number field sieve algorithm. Cryptology ePrint Archive, Report 2019/885 (2019)
32. Hamburg, M.: Spatial encryption. Cryptology ePrint Archive, Report 2011/389 (2011)
33. Hohenberger, S., Waters, B.: Online/offline attribute-based encryption. In: PKC. pp. 293–310. Springer (2014)
34. Hu, C.T., Ferraiolo, D.F., Kuhn, D.R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (ABAC) definition and considerations (2019), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=927500
35. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing. LNCS, vol. 5209, pp. 126–135. Springer (2008)
36. Kamara, S., Lauter, K.E.: Cryptographic cloud storage. In: FC. pp. 136–149. Springer (2010)
37. Kowalczyk, L., Wee, H.: Compact adaptively secure ABE for \mathsf nc^1 from k-lin. In: EUROCRYPT. pp. 3–33. Springer (2019)
38. Lewko, A., Sahai, A., Waters, B.: Revocation systems with very small private keys. In: IEEE S & P. pp. 273–285 (2010)
39. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: EUROCRYPT. pp. 568–588. Springer (2011)
40. Lewko, A.B., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: EUROCRYPT. pp. 62–91. Springer (2010)
41. Lewko, A.B., Waters, B.: Unbounded HIBE and attribute-based encryption. In: EUROCRYPT. pp. 547–567. Springer (2011)
42. Lin, H., Luo, J.: Compact adaptively secure ABE from k-lin: Beyond nc^1 and towards NL. In: EUROCRYPT. pp. 247–277. Springer (2020)
43. Möller, B.: Algorithms for multi-exponentiation. In: SAC. pp. 165–180. Springer (2001)
44. Okamoto, T., Takashima, K.: Fully secure unbounded inner-product and attribute-based encryption. In: ASIACRYPT. pp. 349–366. Springer (2012)
45. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: CCS. pp. 195–203. ACM (2007)
46. de la Piedra, A., Venema, M., Alpár, G.: ABE squared: Accurately benchmarking efficiency of attribute-based encryption. TCHES **2022**(2), 192—-239 (2022)
47. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption. In: CCS. pp. 463–474. ACM (2013)
48. Sahai, A., Seyalioglu, H., Waters, B.: Dynamic credentials and ciphertext delegation for attribute-based encryption. In: CRYPTO. pp. 199–217. Springer (2012)
49. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT. pp. 457–473. Springer (2005)
50. Santos, N., Rodrigues, R., Gummadi, K.P., Saroiu, S.: Policy-sealed data: A new abstraction for building trusted cloud services. In: USENIX Security Symposium. pp. 175–188. USENIX Association (2012)

51. Tomida, J., Kawahara, Y., Nishimaki, R.: Fast, compact, and expressive attribute-based encryption. In: PKC. pp. 3–33. Springer (2020)
52. Venema, M., Alpár, G., Hoepman, J.: Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. In: To appear at Des. Codes Cryptogr. (2022)
53. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. Cryptology ePrint Archive, Report 2008/290 (2008)
54. Waters, B.: Ciphertext-policy attribute-based encryption - an expressive, efficient, and provably secure realization. In: PKC. pp. 53–70. Springer (2011)
55. Yamada, S., Attrapadung, N., Hanaoka, G., Kunihiro, N.: A framework and compact constructions for non-monotonic attribute-based encryption. In: PKC. pp. 275–292. Springer (2014)

A Discussion on the W11b proof

Intuitively, the W11b [53] proof does not generalize to the unbounded setting for the following reason. Usually, selective proofs of CP-ABE schemes [54,47] embed only a vector—for which the length is equal to the number of columns in the policy matrix—in the randomizer of the secret key that is used to hide the master key α. Subsequently, the randomizers used to randomize the attribute public keys are derived (multiplicatively) from this "general randomizer". In contrast, in the proof of the W11b scheme, this general randomizer is extended, and additionally embeds information for the (bounded number of) attributes that would normally be embedded in the "attribute randomizers". Because this is done in an additive fashion, it cannot be split multiplicatively for the sake of our unbounded proof. In addition, because this general randomizer can embed information for only a bounded number of attributes, we cannot use the same strategy in the unbounded setting.

B Attack on first attempt at scheme

Suppose we have key $(h^{\alpha-rb}, h^r, h^{r' f_n(x_{\mathrm{att}_1})}, h^{rb'}, h^{r'})$ and ciphertext $(M \cdot A^s, g^s, g^{s'}, B^{\lambda_1}(B')^{s'}, F_n(x_{\mathrm{att}_1})^{s'}, B^{\lambda_2}(B')^{s'}, F_n(x_{\mathrm{att}_2})^{s'})$ such that $\lambda_1 + \lambda_2 = s$, i.e., the policy requires that a secret key needs att_1 and att_2 to decrypt. Then, we can decrypt the ciphertext, even though the set $\mathcal{S} = \{\mathrm{att}_1\}$ does not satisfy the policy:

$$\begin{aligned} M \cdot A^s / \Big(& e(g^s, h^{\alpha-rb}) e(B^{\lambda_1}(B')^{s'} B^{\lambda_2}(B')^{s'}, h^r)^{-1} \\ & \cdot e(g^{s'}, h^{r' f_n(x_{\mathrm{att}_1})} h^{rb'})^{-2} \cdot e(F_n(x_{\mathrm{att}_1})^{s'})^2, h^{r'} \Big) \\ = M \cdot e(g,h)^{\alpha s} / \Big(& e(g,h)^{\alpha s - rsb} \cdot e(g,h)^{rsb + 2rs'b'} \\ & \cdot e(g,h)^{-2r's' f_n(x_{\mathrm{att}_1}) - 2rs'b'} \cdot e(g,h)^{2r's' f_n(x_{\mathrm{att}_1})} \Big) = M. \end{aligned}$$

C More details on the proof of the regular scheme

C.1 Selective security

The polynomials indeed go to $\mathbf{0}$:

$$\begin{aligned} c_{1,j} &= \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal + s_{\tau(j)} f'_{n_c - 1}(\rho(j)) \\ &= \sum_{k \in [n_2]} A_{j,k} \overline{\mathbf{1}}^{d_2}_{(0,k)} - \sum_{j' \in \chi_j, k \in [n_2]} A_{j',k} F'_{n_c - 1, j'}(\rho(j)) \overline{\mathbf{1}}^{d_2}_{(0,k)} = \mathbf{0}^\intercal, \end{aligned}$$

$$c_{2,j} = s_{\tau(j)} f_n(\rho(j)) = \sum_{j' \in \chi_j, k \in [n_2]} A_{j',k} F_{n,j',k}(\rho(j)) = \mathbf{0}^\intercal,$$

$$k' = \mathbf{1}_0^{d_1} - w_1 \mathbf{1}_0^{d_1} = \mathbf{0},$$

and $k_{1,\mathrm{att}} = r_{\iota(\mathrm{att})} f_n(x_{\mathrm{att}}) + r f'_{n_c-1}(x_{\mathrm{att}}) = \mathbf{0}$ follows from the fact that $-r_{\iota(\mathrm{att})} f_n(x_{\mathrm{att}}) = r f'_{n_c-1}(x_{\mathrm{att}})$. That is, $r_{\iota(\mathrm{att})} f_n(x_{\mathrm{att}})$ is

$$\begin{aligned}
&- \left(\sum_{\substack{i' \in [n_k], j' \in \overline{\Upsilon}, \\ k' \in [n_2]}} w_{k'} u_{i',j',l} \overline{\mathbf{1}}^{d_2}_{(1,i',j',k')} \right) \left(\sum_{\substack{j \in [n_1], \\ k \in [n_2]}} A_{j,k} F_{n,j,k}(x_{\mathrm{att}}) \right) \\
&= - \sum_{j \in \overline{\Upsilon}} \mathbf{A}_j \mathbf{w}^\intercal F_{1,j}(x_{\mathrm{att}}) F'_{n_c-1,j}(x_{\mathrm{att}}) G_{n_k-1,j,\iota(\mathrm{att})}(x_{\mathrm{att}}) \mathbf{1}^{d_1}_{(1,\tau(j))} \\
&= - \sum_{j \in \overline{\Upsilon}} \mathbf{A}_j \mathbf{w}^\intercal F'_{n_c-1,j}(x_{\mathrm{att}}) \mathbf{1}^{d_1}_{(1,\tau(j))}
\end{aligned}$$

and $r f'_{n_c-1}(x_{\mathrm{att}})$ is

$$\begin{aligned}
&\left(\sum_{k' \in [n_2]} w_{k'} \overline{\mathbf{1}}^{d_2}_{(0,k')} \right) \left(\sum_{j \in [n_1], k \in [n_2]} A_{j,k} F'_{n_c-1,j}(x_{\mathrm{att}}) \mathbf{1}^{d_1 \times d_2}_{(1,\tau(j)),(0,k)} \right) \\
&= \sum_{j \in [n_1]} \mathbf{A}_j \mathbf{w}^\intercal F'_{n_c-1,j}(x_{\mathrm{att}}) \mathbf{1}^{d_1}_{(1,\tau(j))} = -r_{\iota(\mathrm{att})} f_n(x_{\mathrm{att}}).
\end{aligned}$$

C.2 Co-selective security

The polynomials indeed go to $\mathbf{0}$:

For $c_{1,j}$, we have that $\mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal = \mathbf{A}_j \mathbf{w}^\intercal \overline{\mathbf{1}}^{d_2}_0$, which is $\mathbf{0}$ if $\rho(j) \in \mathcal{S}$, and otherwise, it is canceled by

$$\begin{aligned}
s_{\tau(j)} f'_{n_c-1}(\rho(j)) &= - \sum_{i \in \overline{[n_c-1]}, j' \in \hat{\chi}_{\tau(j)} \cap \overline{\Upsilon}, k \in [n_2]} A_{j',k} w_k d'_{i,j'} \rho(j)^i \overline{\mathbf{1}}^{d_2}_0 \\
&= - \sum_{j' \in \chi_j \cap \overline{\Upsilon}} \mathbf{A}_{j'} \mathbf{w}^\intercal F_{n_c-1,2,j'}(\rho(j)) \overline{\mathbf{1}}^{d_2}_0 = -\mathbf{A}_j \mathbf{w}^\intercal \overline{\mathbf{1}}^{d_2}_0.
\end{aligned}$$

For $c_{2,j}$, we have that $s_{\tau(j)} f_n(\rho(j))$ is equal to

$$s_{\tau(j)} \sum_{l \in [m]} \left(\hat{F}_{n_c-1,1,l}(\rho(j)) G_{n_k,l}(\rho(j)) - \hat{F}_{n_c-1,2,l}(\rho(j)) \right),$$

where the parts of $s_{\tau(j)}$ associated with $\mathbf{1}^{d_1}_{(2,i)}$ and $\mathbf{1}^{d_1}_{(1,i,l)}$ reprogram $\hat{F}_{n_c-1,1,l}(\rho(j))$ and $\hat{F}_{n_c-1,2,l}(\rho(j))$, respectively, yielding

$$\sum_{\substack{j' \in \chi_j \cap \overline{\Upsilon}, \\ l \in [m]}} \mathbf{A}_{j'} \mathbf{w}^\intercal \mathbf{1}^{d_1}_l \left(F_{n_c-1,j',l}(\rho(j)) G_{n_k,l}(\rho(j)) - F'_{n_c-1,j'}(\rho(j)) \right),$$

which is equal to $\mathbf{0}$ if $j \in \Upsilon$, because it then holds for all $j' \in \chi_j \cap \overline{\Upsilon}$ that $F_{n_c-1,j',l}(\rho(j)) = F'_{n_c-1,j'}(\rho(j)) = 0$. Otherwise, it yields

$$\sum_{l\in[m]} \mathbf{A}_j \mathbf{w}^\intercal \mathbf{1}_l^{d_1} \left(\frac{G_{n_k,l}(\rho(j))}{G_{n_k,l}(\rho(j))} - 1 \right) = \mathbf{0}.$$

For k', we have $\mathbf{1}_0^{d_1} - \mathbf{1}_{0,0}^{d_1 \times d_2} \overline{\mathbf{1}}_0^{d_2} = 0$, and for $k_{1,\mathrm{att}}$, we have

$$r_{\iota(\mathrm{att})} f_n(x_{\mathrm{att}}) = \left(G_{n,\iota(\mathrm{att})}(x_{\mathrm{att}}) - \hat{F}_{n_c-1,2,\iota(\mathrm{att})}(x_{\mathrm{att}}) \right) \overline{\mathbf{1}}_{\iota(\mathrm{att})}^{d_2},$$

where $G_{n,\iota(\mathrm{att})}(x_{\mathrm{att}}) = \hat{F}_{n_c-1,1,\iota(\mathrm{att})}(x_{\mathrm{att}}) G_{n_k,\iota(\mathrm{att})}(x_{\mathrm{att}}) = 0$, because $G_{n_k,\iota(\mathrm{att})}(x_{\mathrm{att}}) = 0$, and thus yields

$$-\hat{F}_{n_c-1,2,\iota(\mathrm{att})}(x_{\mathrm{att}}) \overline{\mathbf{1}}_{\iota(\mathrm{att})}^{d_2} = -\sum_{i=0}^{n_c-1} \mathbf{1}_{(2,i)}^{d_1} x_{\mathrm{att}}^i.$$

This is canceled by

$$r f_{n_c-1}(x_{\mathrm{att}}) = \sum_{i=0}^{n_c-1} \mathbf{1}_{(2,i),0}^{d_1 \times d_2} x_{\mathrm{att}}^i \overline{\mathbf{1}}_0^{d_2} = \sum_{i=0}^{n_c-1} \mathbf{1}_{(2,i)}^{d_1} x_{\mathrm{att}}^i.$$

D The KP-ABE version

We present the KP-ABE version of the scheme in Section 4.

Definition 8 (The unbounded KP-ABE scheme). *Our construction of the unbounded key-policy attribute-based encryption scheme is defined as follows.*

- Setup(λ)*: Taking as input the security parameter λ, the setup generates three groups $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ of prime order p with generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, and chooses a pairing $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. The setup also defines the universe of attributes $\mathcal{U} = \mathbb{Z}_p$ and $n \in \mathbb{N}$ is related to the maximum partition size. In particular, it chooses $n_k \in \mathbb{N}$ as the maximum partition size of the keys, and $n_c = n + 1 - n_k$ as the maximum partition size of the ciphertexts. It then generates random $\alpha, \mathbf{b} = (b, b_0, b_1, ..., b_n, b'_0, ..., b'_{n_k-1}) \in_R \mathbb{Z}_p$. It outputs* $\mathrm{MSK} = (\alpha, \mathbf{b})$ *as its master secret key and publishes the master public key as*
$$\mathrm{MPK} = (g, h, A = e(g,h)^\alpha, B = g^b, B_0 = g^{b_0}, ..., B_n = g^{b_n}, B'_0 = g^{b'_0}, ..., B'_{n_c-1} = g^{b_{n_k-1}}).$$
- KeyGen(MSK, $\mathbb{A}$)*: On input an access policy $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ and $\rho\colon [n_1] \to \mathcal{U}$, the algorithm computes $m = \left\lceil \frac{n_1}{n_k} \right\rceil$, defines $\iota\colon [n_1] \to [m]$ such that $|\iota^{-1}(l)| \leq n_k$ for each $l \in [m]$ and for all $j, j' \in [n_1]$ with $j \neq j'$*

and $\rho(j) = \rho(j')$*, we have* $\iota(j) \neq \iota(j')$*. It then generates random integers* $r, r_1, ..., r_m, v_2, ..., v_{n_2} \in_R \mathbb{Z}_p$ *and computes the secret key as*

$$\mathrm{SK}_{\mathbb{A}} = (\{K_{1,j} = h^{\lambda_j + r_{\iota(j)} \sum_{i=0}^{n_k - 1} b'_i \rho(j)^i}, K_{2,j} = h^{r_{\iota(j)} \sum_{i=0}^{n} b_i \rho(j)^i)}\}_{j \in [n_1]}, \\ \{K_{3,l} = h^{r_l}\}_{l \in [m]}),$$

where $\lambda_j = \mathbf{A}_j(\alpha, v_2, ..., v_{n_2})^\intercal$.

– Encrypt(MPK, $\mathcal{S}, M$)*: A message* $M \in \mathbb{G}_T$ *is encrypted under set of attributes* $\mathcal{S}$ *by computing* $m' = \left\lceil \frac{n_1}{n_c} \right\rceil$*, and defining* $\tau \colon \mathcal{S} \to [m']$ *such that* $|\tau^{-1}(i)| \leq n_c$ *for each* $i \in [m']$*. The algorithm then generates random integers* $s, s_1, ..., s_{m'} \in_R \mathbb{Z}_p$ *and computes the ciphertext as*

$$\mathrm{CT}_{\mathbb{A}} = (C = M \cdot A^s, C' = g^s, \\ \{C_{1,\mathrm{att}} = \textstyle\prod_{i=0}^{n} B_i^{s_{\tau(j)} x_{\mathrm{att}}^i} \cdot \prod_{i=0}^{n_k - 1} (B'_i)^{s x_{\mathrm{att}}^i}\}_{\mathrm{att} \in \mathcal{S}}, \{C_{2,l'} = g^{s_{l'}}\}_{l' \in [m']}).$$

– Decrypt($\mathrm{SK}_{\mathcal{S}}, \mathrm{CT}_{\mathbb{A}}$)*: 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* $\sum_{j \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$*. Then the plaintext* M *is retrieved by computing*

$$C / \left(\prod_{j \in \Upsilon} \left(e(C', K_{1,j}) / e(C_{1,\rho(j)}, K_{3,\iota(j)}) \cdot e(C_{2,\tau(\rho(j))}, K_{2,j}) \right)^{\varepsilon_j} \right).$$

Note that the scheme is correct, i.e., we have

$$\prod_{j \in \Upsilon} \left(e(C', K_{1,j}) / e(C_{1,\rho(j)}, K_{3,\iota(j)}) \cdot e(C_{2,\tau(\rho(j))}, K_{2,j}) \right)^{\varepsilon_j} \\ = \prod_{j \in \Upsilon} (e(g,h)^{\lambda_j s + r_{\iota(j)} s \sum_{i=0}^{n_k - 1} b'_i \rho(j)^i} \\ \cdot e(g,h)^{-r_{\iota(j)} s_{\tau(j)} \sum_{i=0}^{n} b_i \rho(j)^i - r_{\iota(j)} s \sum_{i=0}^{n_k - 1} b'_i \rho(j)^i} \\ \cdot e(g,h)^{r_{\iota(j)} s_{\tau(\rho(j))} (\sum_{i=0}^{n} b_i \rho(j)^i)})^{\varepsilon_j} = e(g,h)^{\sum_{j \in \Upsilon} \lambda_j s} = e(g,h)^{\alpha s},$$

such that $C / e(g,h)^{\alpha s} = M$.

E Online/offline extension

We give the online/offline variant [33] of GLUE. For the key generation, we show how a user can generate the secret key (as in Definition 6) from the online and offline parts in a final step, which can be performed at any time between receiving the keys and decrypting. In contrast, in [33], this step is taken in each run of the decryption algorithm, rather than only once.

E.1 Definition and security model

In [33], Hohenberger and Waters define the notion of online/offline ABE, which can be used to speed up the online execution time of the key generation and encryption algorithms. We adapt the definitions of Hohenberger and Waters, i.e., we include a final step to be executed after the keys have been received by the user. In this step, the online/offline secret keys are combined such that secret keys can be generated that are indistinguishable from secret keys in a regular run of the key generation algorithm.

Definition 9 (Online/offline ABE). *An ABE scheme with optional online/offline key generation and encryption consists of nine algorithms:*

- $\text{Setup}(\lambda) \to (\text{MPK}, \text{MSK})$*: This algorithm is the same as in Definition 2.*
- $\text{Regular.KeyGen}(\text{MSK}, \mathcal{S}) \to \text{SK}_{\mathcal{S}}$*: On input the master secret key* MSK *and some attribute set* $\mathcal{S}$*, this probabilistic algorithm generates a secret key* $\text{SK}_{\mathcal{S}}$*.*
- $\text{Offline.KeyGen}(\text{MSK}) \to \text{ISK}$*: On input the master secret key* MSK*, this optional probabilistic algorithm generates an intermediate secret key* ISK*.*
- $\text{Online.KeyGen}(\text{MSK}, \text{ISK}, \mathcal{S}) \to \text{OO.SK}_{\mathcal{S}}$*: On input the master secret key* MSK*, intermediate secret key* ISK *and some attribute set* $\mathcal{S}$*, this optional probabilistic algorithm generates an online/offline secret key* $\text{OO.SK}_{\mathcal{S}}$*.*
- $\text{FinalStep.KeyGen}(\text{OO.SK}_{\mathcal{S}}) \to \text{SK}_{\mathcal{S}}$*: On input an online/offline secret key* $\text{OO.SK}_{\mathcal{S}}$ *for some attribute set* $\mathcal{S}$*, this optional probabilistic algorithm generates a (regular) secret key* $\text{SK}_{\mathcal{S}}$*.*
- $\text{Regular.Encrypt}(\text{MPK}, \mathbb{A}, M) \to \text{CT}_{\mathbb{A}}$*: On input the master public key* MPK*, some policy* $\mathbb{A}$ *and message* M*, this probabilistic algorithm generates a ciphertext* $\text{CT}_{\mathbb{A}}$*.*
- $\text{Offline.Encrypt}(\text{MPK}) \to \text{ICT}$*: On input the master public key* MPK*, this optional probabilistic algorithm generates an intermediate ciphertext* ICT*.*
- $\text{Online.Encrypt}(\text{MPK}, \text{ICT}, \mathbb{A}, M) \to \text{OO.CT}_{\mathbb{A}}$*: On input the master public key* MPK*, intermediate ciphertext* ICT*, some policy* $\mathbb{A}$ *and message* M*, this optional probabilistic algorithm generates an online/offline ciphertext* $\text{OO.CT}_{\mathbb{A}}$*.*
- $\text{Decrypt}(\text{MPK}, (\text{OO.})\text{SK}_{\mathcal{S}}, (\text{OO.})\text{CT}_{\mathbb{A}}) \to M$*: On input the master public key* MPK*, the (online/offline) secret key* $(\text{OO.})\text{SK}_{\mathcal{S}}$*, and the (online/offline) ciphertext* $(\text{OO.})\text{CT}_{\mathbb{A}}$*, if* $\mathcal{S}$ *satisfies* $\mathbb{A}$*, then it returns* M*. Otherwise, it returns an error message.*

We also adjust the security model in [33] to match our definition of online/offline ABE. Note that, as in the definition in [33], the attacker is not allowed to access intermediate keys and ciphertexts.

Definition 10 (Full CPA-security for online/offline ABE). *We define the security game between challenger and attacker as follows:*

- ***Setup phase:*** *The challenger runs* $\text{Setup}(\lambda)$ *to obtain* MPK *and* MSK*, and sends the master public key* MPK *to the attacker. The challenger also initializes an empty list* L *and a counter* $c_L = 0$*.*

- ***First query phase:*** *The attacker can make the following queries:*
  - ***Regular key query:*** *The attacker queries secret keys for $\mathcal{S}$, and obtains* $\mathrm{SK}_{\mathcal{S}} \leftarrow \mathrm{KeyGen}(\mathrm{MSK}, \mathcal{S})$ *in response.*
  - ***Intermediate secret key query:*** *The attacker queries an intermediate secret key. The challenger generates* $\mathrm{ISK} \leftarrow \mathrm{Offline.KeyGen}(\mathrm{MSK})$*, stores* (c_L, ISK) *in list L and updates the counter* $c_L \leftarrow c_L + 1$.
  - ***Online/offline secret key query:*** *The attacker queries an online/offline secret key for $\mathcal{S}$ and set of indices $\mathcal{I} \subseteq [0, c_L]$ such that none of the indices in $\mathcal{I}$ have been queried before. (Otherwise, there is no entry in the table for one or more indices, and then, the challenger returns an error message.) The challenger selects intermediate secret keys* (i, ISK_i) *for all $i \in \mathcal{I}$ from the list, deletes these entries from the list and generates* $\mathrm{OO.SK}_{\mathcal{S}} \leftarrow \mathrm{Online.KeyGen}(\mathrm{MSK}, \{\mathrm{ISK}_i\}_{i \in \mathcal{I}}, \mathcal{S})$.
- ***Challenge phase:*** *The attacker specifies some policy $\mathbb{A}^*$ such that for all $\mathcal{S}$ in the first phase, we have $\mathcal{S}$ does not satisfy $\mathbb{A}^*$, and generates two messages M_0 and M_1 of equal length. The attacker sends these to the challenger and chooses whether it wants to be queried on a regular or an online/offline ciphertext:*
  - ***Regular challenge:*** *The challenger flips a coin, i.e.,* $\beta \in_R \{0,1\}$*, encrypts M_β under $\mathbb{A}^*$, i.e.,* $\mathrm{CT}_{x^*} \leftarrow \mathrm{Encrypt}(\mathrm{MPK}, \mathbb{A}^*, M_\beta)$*, and sends the resulting ciphertext* $\mathrm{CT}_{\mathbb{A}^*}$ *to the attacker.*
  - ***Online/offline challenge:*** *The challenger first generates intermediate ciphertexts* $\mathrm{ICT} \leftarrow \mathrm{Offline.Encrypt}(\mathrm{MPK})$*. Then, the challenger flips a coin, i.e.,* $\beta \in_R \{0,1\}$*, online encrypts M_β under $\mathbb{A}^*$ with* ICT*, i.e.,* $\mathrm{OO.CT}_{\mathbb{A}^*} \leftarrow \mathrm{Online.Encrypt}(\mathrm{MPK}, \mathrm{ICT}, \mathbb{A}^*, M_\beta)$*, and sends the resulting ciphertext* $\mathrm{OO.CT}_{\mathbb{A}^*}$ *to the attacker.*
- ***Second query phase:*** *This phase is identical to the first query phase, with the additional restriction that the attacker can only query $\mathcal{S}$ such that it does not satisfy $\mathbb{A}^*$.*
- ***Decision phase:*** *The attacker outputs a guess β' for β.*

The advantage of the attacker is defined as $\| \Pr[\beta' = \beta] - \frac{1}{2} |$*. An online/offline scheme is fully secure if all polynomial-time attackers have at most a negligible advantage in this security game.*

E.2 Online/offline version of GLUE

Definition 11 (GLUE-OO).

- $\mathrm{Setup}(\lambda) \rightarrow (\mathrm{MPK}, \mathrm{MSK})$*: This algorithm is the same as in Definition 6.*
- $\mathrm{Regular.KeyGen}(\mathrm{MSK}, \mathcal{S}) \rightarrow \mathrm{SK}_{\mathcal{S}}$*: This algorithm is the same as in Definition 6.*
- $\mathrm{Offline.KeyGen}(\mathrm{MSK}) \rightarrow \mathrm{ISK}$*: The algorithm generates two types of "intermediate secret keys".*
  - *First type: The algorithm generates random integer r and stores* $(K = h^{\alpha - rb}, K' = h^r, r)$.

- *Second type: The algorithm generates random integers* $r', z_j \in_R \mathbb{Z}_p$ *for all* $j \in [n_k]$*, and then stores* $(\{\hat{K}_{1,j} = h^{z_j}, z_j\}_{j\in[n_k]}, K_2 = h^{r'}, r')$.

– Online.KeyGen(MSK, ISK, $\mathcal{S}$) $\to$ $\mathrm{SK}_{\mathcal{S}}$*: On input set of attributes* $\mathcal{S}$*, the algorithm computes* $m = \left\lceil \frac{|\mathcal{S}|}{n} \right\rceil$*, defines* $\iota\colon \mathcal{S} \to [m]$ *such that* $|\iota^{-1}(l)| \leq n$ *for each* $l \in [m]$*, and further defines* $\hat{\iota}\colon \mathcal{S} \to [n_k]$ *such that it is injective on each subdomain* $\iota^{-l}(l)$ *for all* $l \in [m]$*. It takes one intermediate secret key of the first type, and* m *of the second type:*

$$(K = h^{\alpha - rb}, K' = h^r, r), (\{\hat{K}_{1,j,l} = g^{z_{j,l}}, z_{j,l}\}_{j\in[n_k]}, K_{2,l} = h^{r_l}, r_l)_{l\in[m]},$$

sets $\hat{K}_{1,\mathrm{att}} = \hat{K}_{1,\hat{\iota}(\mathrm{att}),\iota(\mathrm{att})}$*, then computes*

$$\hat{K}_{3,\mathrm{att}} = \left(r_{\iota(\mathrm{att})} \sum_{i=0}^{n} b_i x_{\mathrm{att}}^i + r \sum_{i=0}^{n_c-1} b'_i x_{\mathrm{att}}^i \right) - z_{\hat{\iota}(\mathrm{att}),\iota(\mathrm{att})}.$$

and outputs the secret key as

$$\mathrm{SK}_{\mathcal{S}} = (K, K', \iota, \{\hat{K}_{1,\mathrm{att}}, \hat{K}_{3,\mathrm{att}}\}_{\mathrm{att}\in\mathcal{S}}, \{K_{2,l}\}_{l\in[m]}).$$

– FinalStep.KeyGen(OO.$\mathrm{SK}_{\mathcal{S}}$) $\to$ $\mathrm{SK}_{\mathcal{S}}$*: The user can generate the secret keys as in Definition 6 from* $(K, K', \iota, \{\hat{K}_{1,\mathrm{att}}, K_{2,l}, \hat{K}_{3,\mathrm{att}}\}_{\mathrm{att}\in\mathcal{S}, l\in[m]})$*, by computing for each* $\mathrm{att} \in \mathcal{S}$ *the secret key component* $K_{1,\mathrm{att}}$ *as in Definition 6:* $K_{1,\mathrm{att}} = \hat{K}_{1,\mathrm{att}} \cdot h^{\hat{K}_{3,\mathrm{att}}}$.

– Regular.Encrypt(MPK, $\mathbb{A}, M$) $\to$ $\mathrm{CT}_{\mathbb{A}}$*: This algorithm is the same as in Definition 6.*

– Offline.Encrypt(MPK) $\to$ $\mathrm{CT}_{\mathbb{A}}$*: The algorithm generates two types of "intermediate ciphertexts".*

- *First type: It selects* $s \in_R \mathbb{Z}_p$ *and stores* $(\hat{C} = A^s, C' = g^s, s)$.
- *Second type: It selects* $s', \hat{\lambda}_1, ..., \hat{\lambda}_{n_c} \in_R \mathbb{Z}_p$*,* $(\hat{x}_{j,1}, ..., \hat{x}_{j,n}) \in_R \mathbb{Z}_p^n$*, sets* $\hat{x}_{j,0} = 1$ *for all* $j \in [n_c]$*, and stores*

$$(\{\hat{C}_{1,j} = B^{\hat{\lambda}_j} \cdot \prod_{i=0}^{n_c-1} (B'_i)^{s'\hat{x}_{j,i}}, \hat{C}_{2,j} = \prod_{i=0}^{n} B_i^{s'\hat{x}_{j,i}},$$
$$\hat{\lambda}_j, \{\hat{x}_{j,i}\}_{i\in\overline{[n]}}\}_{j\in[n_c]}, C_3 = g^{s'}, s').$$

– Online.Encrypt(MPK, ICT, $\mathbb{A}, M$) $\to$ $\mathrm{CT}_{\mathbb{A}}$*: On input policy* $\mathbb{A} = (\mathbf{A}, \rho)$*, the algorithm selects one "intermediate ciphertext"* $(\hat{C}, C', s)$ *of the first type, and then* $m' = \left\lceil \frac{n_1}{n_c} \right\rceil$ *"intermediate ciphertexts" of the second type*

$$(\{\hat{C}_{1,j,l'}, \hat{C}_{2,j,l'}, \hat{\lambda}_{j,l'}, \{\hat{x}_{j,i,l'}\}_{i\in[0,n]}\}_{j\in[n_c]}, C_{3,l'}, s_{l'})$$

(for all $l' \in [m']$*). It defines* τ *and* λ_j *as in Definition 6, and further defines* $\hat{\tau}\colon [n_1] \to [n_c]$ *such that* $\hat{\tau}$ *is injective on each subdomain* $\tau^{-1}(l')$ *with* $l' \in$

$[m']$, *i.e., each attribute gets mapped to a unique tuple* $(\hat{C}_{1,j,l'}, ..., s_{l'})$. *It encrypts message* M *by setting* $C = M \cdot \hat{C}$ *and sets for all* $j \in [n_1], i \in [n]$*:*

$$\hat{C}_{1,j} = \hat{C}_{1,\hat{\tau}(j),\tau(j)}, \;\; \hat{C}_{2,j} = \hat{C}_{2,\hat{\tau}(j),\tau(j)}$$
$$\hat{C}_{4,j} = \lambda_j - \hat{\lambda}_{\hat{\tau}(j),\tau(j)}, \;\; \hat{C}_{5,j,i} = s_{\hat{\tau}(j)}(\rho(j)^i - \hat{x}_{\hat{\tau}(j),i,\tau(j)})$$

The user publishes the ciphertext as

$$\mathrm{CT}_{\mathbb{A}} = (C, C', \tau, \{\hat{C}_{1,j}, \hat{C}_{2,j}, \hat{C}_{4,j}, \hat{C}_{5,j,i}\}_{i\in[n],j\in[\hat{n}_1]}, \{C_{3,l'}\}_{l'\in[\hat{m}']}),$$

Note that the ciphertext increases by $n_1(n+1)$ *elements in* $\mathbb{Z}_p$ *compared to regular ciphertexts.*

– Decrypt$(\mathrm{MPK}, (\mathrm{OO.})\mathrm{SK}_{\mathcal{S}}, (\mathrm{OO.})\mathrm{CT}_{\mathbb{A}}) \to M$*:If* $\mathcal{S}$ *satisfies* $\mathbb{A}$*, determine* ε_j *and* Υ *as in Definition 6, set* $\Upsilon_l = \{j \in \Upsilon \mid \iota(\rho(j)) = l\}$ *for all* $l \in [m]$ *and compute* $C/\left(e(C', K) \cdot C_1 \cdot C_2 \cdot C_3\right)$*, where*

$$C_1 = e\left(\prod_{j\in\Upsilon} \hat{C}_{1,j}^{\varepsilon_j} \cdot (g^b)^{\sum_{j\in\Upsilon} \varepsilon_j \hat{C}_{4,j}} \cdot \prod_{i=1}^{n_c-1} (g^{b_i})^{\sum_{j\in\Upsilon} \varepsilon_j \hat{C}_{5,j,i}}, K'\right)$$
$$C_2 = \prod_{j\in\Upsilon} e(C_{3,\tau(j)}^{-\varepsilon_j}, K_{1,\rho(j)})$$
$$C_3 = \prod_{l\in[m]} e\left(\prod_{j\in\Upsilon_l} \hat{C}_{2,j}^{\varepsilon_j} \cdot \prod_{i=1}^{n} (g^{b_i})^{\sum_{j\in\Upsilon_l} \varepsilon_j \hat{C}_{5,j,i}}, K_{2,l}\right).$$

Correctness. Correctness of the decryption algorithm follows from simply showing that $\hat{C}_{1,j} \cdot (g^b)^{\hat{C}_{4,j}} \cdot \prod_{i=1}^{n_c-1} (g^{b_i})^{\hat{C}_{5,j,i}} = C_{1,j}$, and $\hat{C}_{2,j} \cdot \prod_{i=1}^{n} (g^{b_i})^{\hat{C}_{5,j,i}} = C_{2,j}$, where $C_{1,j}$ and $C_{2,j}$ are as in Definition 6. Then, the correctness proof is identical to that in this definition. Clearly, this is the case, because

$$\hat{C}_{1,j} \cdot (g^b)^{\hat{C}_{4,j}} \cdot \prod_{i=1}^{n_c-1} (g^{b_i})^{\hat{C}_{5,j,i}}$$
$$= g^{\hat{\lambda}_{\hat{\tau}(j),\tau(j)} b + s_{\hat{\tau}(j)} \sum_{i=0}^{n_c-1} b'_i \hat{x}_{\hat{\tau}(j),i,\tau(j)}} \cdot g^{b(\lambda_j - \hat{\lambda}_{\hat{\tau}(j),\tau(j)})}$$
$$\cdot \prod_{i=1}^{n_c-1} g^{s_{\tau(j)} b_i (\rho(j)^i - \hat{x}_{\hat{\tau}(j),i,\tau(j)})} = g^{\lambda_j b + s_{\tau(j)} \sum_{i=0}^{n_c-1} b'_i \rho(j)^i} = C_{1,j},$$

and

$$\hat{C}_{2,j} \cdot \prod_{i=1}^{n} (g^{b_i})^{\hat{C}_{5,j,i}} = g^{s_{\hat{\tau}(j)} \sum_{i=0}^{n} b_i \hat{x}_{\hat{\tau}(j),i,\tau(j)}} \cdot \prod_{i=1}^{n} (g^{b_i})^{s_{\hat{\tau}(j)}(\rho(j)^i - \hat{x}_{\hat{\tau}(j),i,\tau(j)})}$$
$$= g^{s_{\tau(j)} \sum_{i=0}^{n} b_i \rho(j)^i} = C_{2,j}.$$

E.3 Security proof

We prove selective security of the online/offline variant of the unbounded CP-ABE scheme in Section E. For this, we provide a security reduction to the selective/full security of the basic scheme, mirroring the proof in [33]. For simplicity, we assume that, in all algorithms, the online/offline algorithms are queried. For the regular algorithms, we relay the responses of the challenger of the regular scheme.

Theorem 2. *The online/offline scheme is secure if the scheme in Definition 6 is secure.*

Proof. Suppose $\mathcal{A}_{\text{OO-CP-ABE}}$ is a polynomial-time attacker that can break the online/offline variant of the scheme with non-negligible advantage ε. We show how we can break the "regular" variant by interacting with this attacker.

- **Initialization phase:** Let $\mathbb{A} = (\mathbf{A}, \rho)$, where $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ with $n_1, n_2 < q$, denote the access structure generated by attacker $\mathcal{A}_{\text{CP-ABE}}$ for which $\mathcal{A}_{\text{OO-CP-ABE}}$ can selectively break the normal scheme. The attacker $\mathcal{A}_{\text{CP-ABE}}$ sends it to the challenger.
- **Setup phase:** The public parameters generated by the challenger (in the CP-ABE security game), i.e., $\text{MPK} = (g, e(g,h)^\alpha, g^{\mathbf{b}})$ and sent to attacker $\mathcal{A}_{\text{CP-ABE}}$ are passed along to attacker $\mathcal{A}_{\text{OO-CP-ABE}}$.
- **First query phase:** Attacker $\mathcal{A}_{\text{OO-CP-ABE}}$ queries secret keys for sets $\mathcal{S}$, which $\mathcal{A}_{\text{CP-ABE}}$ relays to the challenger. The challenger generates

$$\text{SK}_{\mathcal{S}} = (K = h^{\alpha - rb}, K' = h^r, \iota,$$
$$\{K_{1,\text{att}} = h^{r_{\iota(\text{att})}(\sum_{i=0}^{n} b_i x_{\text{att}}^i) + r\sum_{i=0}^{n_c-1} b'_i x_{\text{att}}^i}\}_{\text{att}\in\mathcal{S}}, \{K_{2,l} = h^{r_l}\}_{l\in[m]}).$$

  Attacker $\mathcal{A}_{\text{CP-ABE}}$ then selects random $r_{2,\text{att}} \in_R \mathbb{Z}_p$ for all $\text{att} \in \mathcal{S}$ and computes $\hat{K}_{1,\text{att}} = K_{1,\text{att}} \cdot g^{-r_{2,\text{att}}}$. Then $\mathcal{A}_{\text{CP-ABE}}$ sends

$$\text{SK}'_{\mathcal{S}} = (K, K', \iota, \{\hat{K}_{1,\text{att}}, \hat{K}_{3,\text{att}} = r_{2,\text{att}}\}_{\text{att}\in\mathcal{S}}, \{K_{2,l}\}_{l\in[m]})$$

  to $\mathcal{A}_{\text{OO-CP-ABE}}$. Note that this key has the same distribution as the keys generated by the scheme, because $z_{j,l}$ in the scheme is a randomly generated integer, and the final step in the key generation yields a normal secret key.
- **Challenge phase:** Attacker $\mathcal{A}_{\text{CP-ABE}}$ generates two random messages M_0, M_1 and sends them to the challenger, who flips a coin $\beta \in_R \{0,1\}$ and encrypts M_β. The challenger sends

$$\text{CT}_{\mathbb{A}} = (C, C', \{C_{1,j}, C_{2,j}\}_{j\in\{1,\ldots,n_1\}}, \{C_{3,l'} = g^{s_{l'}}\}_{l'\in\{1,\ldots,m'\}}),$$

  to attacker $\mathcal{A}_{\text{CP-ABE}}$, who picks random $\hat{C}_{4,j}, \hat{C}_{5,j,i} \in_R \mathbb{Z}_p$ for each $j \in [n_1], i \in [n]$ and computes

$$\text{CT}'_{\mathbb{A}} = (C, C', \{\hat{C}_{1,j} = C_{1,j} \cdot (g^b)^{-\hat{C}_{4,j}} \cdot \textstyle\prod_{i=1}^{n_c-1}(g^{b'_i})^{-\hat{C}_{5,j,i}},$$
$$\hat{C}_{2,j} = C_{2,j} \cdot \textstyle\prod_{i=0}^{n}(g^{b_i})^{-\hat{C}_{5,j,i}}, \hat{C}_{4,j}, \hat{C}_{5,j,i}\}_{j\in[n_1]}, \{C_{3,l'}\}_{l'\in[m']}),$$

  which is well-formed as can be observed from the proof of correctness. Attacker $\mathcal{A}_{\text{CP-ABE}}$ then sends $\text{CT}'_{\mathbb{A}}$ to attacker $\mathcal{A}_{\text{OO-CP-ABE}}$.

- **Second key query phase:** The second query phase is the same as the first key query phase.
- **Decision phase:** Attacker $\mathcal{A}_{\text{OO-CP-ABE}}$ outputs a guess β' on β, which attacker $\mathcal{A}_{\text{CP-ABE}}$ relays to the challenger.

Attacker $\mathcal{A}_{\text{CP-ABE}}$ guesses correctly whenever $\mathcal{A}_{\text{OO-CP-ABE}}$ guesses correctly, so the attacker does this with exactly the same advantage ε as well.

F Two non-monotone versions of GLUE

We give two pair encoding schemes based on our generalized unbounded scheme. The first scheme is the generalized unbounded analog to TKN20 [51]. In this scheme, we replace the full-domain hash of TKN20 by a Boneh-Boyen hash. By extension, this scheme can be extended with an online/offline version. The second scheme is a generalized unbounded version of the unbounded CP-ABE scheme supporting OSW-type negations (Att19-I-CP) [11], which is the non-monotone extension of RW13 [47]. In this scheme, like in RW13, we replace their instantiation of the Boneh-Boyen hash, i.e., a 1-degree polynomial, with a generalized Boneh-Boyen hash, i.e., an n-degree polynomial. For future work, it would be interesting to consider whether the two new schemes can be combined, thus achieving the support of OSWOT-type negations [15]. In such a scheme, a generalized Boneh-Boyen hash could be used for both the label universe and the attribute universe. In this way, the same randomizer can be used for multiple labels, as well multiple attributes with the same label. Due to the sole use of Boneh-Boyen hashes, the scheme can be extended to an online/offline version, such that the key generation and encryption costs can be minimized.

F.1 Generalized unbounded analog to TKN20

By applying the ciphertext-policy augmentation (confined to OR) to the direct sum of an AND-composition of our generalized unbounded scheme and the IBE predicate, and an AND-composition of our generalized unbounded scheme and the NIBE predicate (all from [11]), we obtain a provably secure scheme supporting OT-type negations (analog to TKN20 [51]). Here, we use our generalized unbounded scheme for the label universe. In this way, we can avoid the use of a hash function modeled as a random oracle in TKN20 [51], and thus benefit from our online/offline extensions. The variables $n_c, n_k, n, \mathcal{S}, \rho, \iota, \tau$, $n_1, n_2, \lambda_i, m, m'$ are as in Definition 6. We also include a function ρ_2 that maps the row to 1 if the attribute is not negated in the policy, and to 2 if it is negated, and $\rho_1 \colon [n_1] \to \{0,1\}^*$ and $\kappa \colon \mathcal{S} \to \{0,1\}^*$ map the attributes in the policy and set, respectively, to labels (represented as string).

Definition 12 (Generalized unbounded ABE with OT-NMSPs).

- Param(par) $\to 2n + 2n_c + 3$. *Let*

$$\mathbf{b} = (b, b_{0,0}, ..., b_{0,n}, b_{1,0}, ..., b_{1,n}, b'_{0,0}, ..., b'_{0,n_c-1}, b'_{1,0}, ..., b'_{1,n_c-1}),$$

where $n = n_c + n_k - 1$. *We define*

$$f_{n,\beta}(y_{\text{lab}}) = \sum_{i=0}^{n} b_{\beta,i} y_{\text{lab}}^i, \quad f'_{n_c-1,\beta}(y_{\text{lab}}) = \sum_{i=0}^{n_c-1} b'_{\beta,i} y_{\text{lab}}^i$$

for $\beta \in \{0,1\}$, *and*

$$g(x_{\text{att}}, y_{\text{lab}}) = f_{n,1}(y_{\text{lab}}) x_{\text{att}} + f_{n,0}(y_{\text{lab}})$$
$$g'(x_{\text{att}}, y_{\text{lab}}) = f'_{n_c-1,1}(y_{\text{lab}}) x_{\text{att}} + f'_{n_c-1,0}(y_{\text{lab}}).$$

- $\text{EncKey}(\mathcal{S}, \kappa) \to (\mathbf{r}, k', \{k_{1,\text{att}}\}_{\text{att}\in\mathcal{S}})$. *Let* $\mathbf{r} = (r, \{r_l\}_{l\in[m]})$, $k' = \alpha - rb$, *put additional restriction on* ι *so that, for all* $\text{att} \neq \text{att}' \in \mathcal{S}$ *with* $\kappa(\text{att}) = \kappa(\text{att}')$, *we ensure* $\iota(\text{att}) \neq \iota(\text{att}')$, *and*

$$k_{1,\text{att}} = r_{\iota(\text{att})} g(x_{\text{att}}, y_{\kappa(\text{att})}) + r g'(x_{\text{att}}, y_{\kappa(\text{att})})$$

  where for any label lab, *we denote* y_{lab} *as its representation in* $\mathbb{Z}_p$.
- $\text{EncCt}((\mathbf{A}, \rho)) \to (\mathbf{s}, \hat{\mathbf{s}}, \{c_{1,j}, c_{2,j}, \bar{c}_{1,j'}, \bar{c}_{2,j'}, \bar{c}_{3,j'}\}_{j\in\Phi, j'\in\overline{\Phi}})$, *where* $\rho = (\rho, \rho_1, \rho_2)$. *Let* $\Phi = \{j \in [n_1] \mid \rho_2(j) = 1\}$ *and* $\overline{\Phi} = [n_1] \setminus \Phi$. *Let* $\mathbf{s} = (s, \{s_{l'}\}_{l'\in[m']})$ *and* $\hat{\mathbf{s}} = (\hat{v}_2, ..., \hat{v}_{n_2})$. *Then, we distinguish between whether* $j \in \Phi$ *or not.*
  - *For* $j \in \Phi$: $c_{1,j} = \mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal + s_{\tau(j)} g'(\rho(j), y_{\rho_1(j)})$ *and* $c_{2,j} = s_{\tau(j)} g(\rho(j), y_{\rho_1(j)})$.
  - *For* $j \in \overline{\Phi}$: $\bar{c}_{1,j} = \mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal + s_{\tau(j)} f'_{n_c-1,0}(y_{\rho_1(j)})$, $\bar{c}_{2,j} = s_{\tau(j)} g(\rho(j), y_{\rho_1(j)})$ *and* $\bar{c}_{3,j} = \rho(j)\mathbf{A}_j(sb, \hat{\mathbf{s}})^\intercal + s_{\tau(j)} f'_{n_c-1,0}(y_{\rho_1(j)})$.
- $\text{Pair}(\mathcal{S}, (\mathbf{A}, \rho, \rho_1, \rho_2))$: *For* $\mathcal{S}$ *that satisfies* $\mathbb{A}$, *we have some* $\Upsilon \subseteq [n_1]$ *such that* $\{\varepsilon_j \in \mathbb{Z}_p\}_{j\in\Upsilon}$ *exists with* $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$ *(Definition 1). We also split* Υ *in two subsets* $\Upsilon' = \Upsilon \cap \Phi$ *and* $\overline{\Upsilon}' = \Upsilon \setminus \Upsilon'$. *We retrieve* αs *by computing*
  - *For all* $j \in \Upsilon'$:

$$\varepsilon_j (r c_{1,j} - s_{\tau(j)} k_{1,\rho(j)} + r_{\iota(\rho(j))} c_{2,j}) = \varepsilon_j r \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal$$

  - *For all* $j \in \overline{\Upsilon}'$, *we set* $x_{\text{att}} = \kappa^{-1}(y_{\rho_1(j)})$, *and compute*

$$\frac{\varepsilon_j}{x_{\text{att}} - \rho(j)} \left(r(x_{\text{att}} \bar{c}_{1,j} - \bar{c}_{3,j}) - s_{\tau(j)} k_{1,\rho(j)} + r_{\iota(\rho(j))} c_{2,j} \right) = \varepsilon_j r \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal$$

  Then, we retrieve $\alpha s = sk' - \sum_{j\in\Upsilon} \varepsilon_j r \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal$.

Performance analysis of the selectively secure instantiation. For the most efficient decryption algorithm, we push $\mathbf{s}$ and $\mathbf{r}$ in $\mathbb{H}$, and the polynomials in $\mathbb{G}$. Then, the costs are:

- KeyGen: $m + 2$ exponentiations in $\mathbb{H}$, and $|\mathcal{S}|$ exponentiations in $\mathbb{G}$;
- Encrypt: $m' + 1$ exponentiations in $\mathbb{H}$, $|\Phi|(2n + 2n_c + 3) + |\overline{\Phi}|(2n + 2n_c + 4)$ exponentiations in $\mathbb{G}$ and 1 exponentiation in $\mathbb{G}_T$;
- Decrypt: $2 + \left\lceil \frac{|\Upsilon|}{n_c} \right\rceil + \left\lceil \frac{|\Upsilon|}{n_k} \right\rceil$ pairing operations and $4|\overline{\Upsilon}'|$ exponentiations in $\mathbb{G}$.

F.2 Generalized unbounded ABE supporting OSW-type negations

By applying the generic negation in [6] and the direct sum transformation and ciphertext-policy augmentation (confined to OR) in [11] to the PES in Section 4.1, we obtain a provably secure generalized unbounded scheme supporting MSPs with OSW-type negations. Note that this yields a generalized variant of Att19-CP-I [11]. The variables $n_c, n_k, n, \mathcal{S}, \rho, \iota, \tau$, $n_1, n_2, \lambda_i, m, m'$ are as in Definition 6. In this definition, we also include a function ρ_2 that maps the row to 1 if the attribute is not negated in the policy, and to 2 if it is negated.

For this scheme, we require Lagrange interpolation. That is, given $n+1$ points $(x, f_n(x))$, with $x \in \mathcal{S}$ and $|\mathcal{S}| = n+1$, on a polynomial. Then, we can reconstruct the the point $f_n(z)$ by computing

$$f_n(z) = \sum_{x \in S} \Lambda_{S,x} f_n(x) \pmod p,$$

where

$$\Lambda_{S,x,z} = \prod_{y \in S \setminus \{x\}} \frac{z-y}{x-y} \pmod p.$$

Definition 13 (GLUE-N). *GLUE-N, which supports OSW-type negations, is defined as:*

- Param(par) $\to 2n + 2n_c + 3$. *Let*

$$\mathbf{b} = (b, b'', b^{(3)}, b_0, ..., b_n, \bar{b}_0, ..., \bar{b}_n, b'_0, ..., b'_{n_c-1}, \bar{b}'_0, ..., \bar{b}'_{n_c-1}),$$

  where $n = n_c + n_k - 1$ with $n_k \geq n_c$, and

$$f_n(x_{\text{att}}) = \sum_{i=0}^{n} b_i x_{\text{att}}^i, f'_{n_c-1}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} b'_i x_{\text{att}}^i,$$
$$\bar{f}_n(x_{\text{att}}) = \sum_{i=0}^{n} \bar{b}_i x_{\text{att}}^i, \bar{f}'_{n_c-1}(x_{\text{att}}) = \sum_{i=0}^{n_c-1} \bar{b}'_i x_{\text{att}}^i.$$

- EncKey($\mathcal{S}$) $\to (\mathbf{r}, k', k'', \bar{k}'', \{k_{1,\text{att}}, \bar{k}_{1,\text{att}}, \bar{k}_{2,\text{att}}\}_{\text{att} \in \mathcal{S}})$. *Let* $\mathbf{r} = (r, r', \bar{r}, \{r_l, \bar{r}_{\text{att}}, \bar{r}'_l\}_{\text{att} \in \mathcal{S}, l \in [m]})$, $k' = \alpha - r'b$,

$$k'' = r'b'' + rb^{(3)}, \quad \bar{k}'' = r'b'' + \bar{r}b^{(3)},$$
$$k_{1,\text{att}} = r_{\iota(\text{att})} f_n(x_{\text{att}}) + r f'_{n_c-1}(x_{\text{att}}),$$
$$\bar{k}_{1,\text{att}} = \bar{r}'_{\iota(\text{att})} \bar{f}'_{n_c-1}(x_{\text{att}}) + \bar{r}_{\iota(\text{att})} \bar{b}_0, \quad \bar{k}_{2,\text{att}} = \bar{r}_{\iota(\text{att})} \bar{f}_n(x_{\text{att}}),$$

  such that $\sum_{l \in [m]} \bar{r}'_l = \bar{r}$. Note that we require that each partition is full, i.e., $|\iota^{-1}(l)| = n_k$ for all $l \in [m]$. If needed, this can be done by using dummy attributes [45].
- EncCt($(\mathbf{A}, \rho, \rho_2)$) $\to (\mathbf{s}, \hat{\mathbf{s}}, \{c_{1,j}, c_{2,j}, c_{3,j}, \bar{c}_{2,j'}, \bar{c}_{3,j'}\}_{j \in \Phi, j' \in \overline{\Phi}})$. *Let* $\Phi = \{j \in [n_1] \mid \rho_2(j) = 1\}$ *and* $\overline{\Phi} = [n_1] \setminus \Phi$. *Let* $\mathbf{s} = (s, \{s_{l'}\}_{l' \in [m']}, \{s'_j\}_{j \in [n_1]})$ *and* $\hat{\mathbf{s}} = (\hat{v}_2, ..., \hat{v}_{n_2})$. *We set* $c_{1,j} = \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal + s'_j b''$, *and*

- *For $j \in \Phi$:*

$$c_{2,j} = s'_j b^{(3)} + s_{\tau(j)} f'_{n_c-1}(\rho(j)), c_{3,j} = s_{\tau(j)} f_n(\rho(j)).$$

- *For $j \in \overline{\Phi}$:*

$$\bar{c}_{2,j} = s'_j b^{(3)} + s_{\tau(j)} \bar{f}'_{n_c-1}(\rho(j)), \bar{c}_{3,j} = s_{\tau(j)} \bar{f}_n(\rho(j)).$$

We require that each partition that has at least one negated attribute in it is full and only contains negated attributes that occur in a conjunction, i.e., for all $j \in [n_1]$ with $\rho_2(j) = 2$, we have $|\chi_j| = n_c$, where $\chi_j = \{j' \in [n_1] \mid \tau(j') = \tau(j) \wedge \rho_2(j') = 2\}$. If needed, this can be done by using dummy attributes (not issued for keys) [45].

– $\mathrm{Pair}(\mathcal{S}, (\mathbf{A}, \rho, \rho_2))$*: For $\mathcal{S}$ that satisfies $\mathbb{A}$, we have some $\Upsilon \subseteq [n_1]$ such that $\{\varepsilon_j \in \mathbb{Z}_p\}_{j \in \Upsilon}$ exists with $\sum_{j \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$ (Definition 1). We also split Υ in two subsets $\Upsilon' = \Upsilon \cap \Phi$ and $\overline{\Upsilon}' = \Upsilon \setminus \Upsilon'$. We retrieve αs by first computing for each ciphertext partition $l' \in [m']$ with some row $j \in \overline{\Upsilon}'$ with $\tau(j) = l'$ and $l \in [m]$:*

$$\sum_{\mathrm{att} \in \Psi_l} \Lambda_{\Omega_{j,l}, x_{\mathrm{att}}, 0} s_{\tau(j)} \bar{k}_{2,\mathrm{att}} + \sum_{j' \in \chi_j} \Lambda_{\Omega_{j,l}, \rho(j'), 0} \bar{r}_l \bar{c}_{3,j'} = \bar{r}_l s_{\tau(j)} \bar{f}_n(0),$$

where $\Psi_l = \{\mathrm{att} \in \mathcal{S} \mid \iota(\mathrm{att}) = l\}$, and $\Omega_{j,l} = \{x_{\mathrm{att}} \mid \mathrm{att} \in \Psi_l\} \cup \{\rho(j') \mid j' \in [n_1], \tau(j) = \tau(j')\}$. Then, we use it to retrieve for all $\mathrm{att} \in \Psi_l$:

$$s_{\tau(j)} \bar{k}_{1,\mathrm{att}} - \bar{r}_l s_{\tau(j)} \bar{f}_n(0) = s_{\tau(j)} \bar{r}'_{\iota(\mathrm{att})} \bar{f}'_{n_c-1}(x_{\mathrm{att}}),$$

which we use to recover for each $j \in \overline{\Upsilon}'$ and $l \in [m]$:

$$\sum_{\mathrm{att} \in \Psi_l} \Lambda_{\Psi'_l, x_{\mathrm{att}}, \rho(j)} s_{\tau(j)} \bar{r}'_l \bar{f}'_{n_c-1}(x_{\mathrm{att}}) = s_{\tau(j)} \bar{r}'_l \bar{f}'_{n_c-1}(\rho(j)),$$

where $\Psi'_l = \{x_{\mathrm{att}} \mid \mathrm{att} \in \Psi_l\}$. Then, we retrieve

$$\sum_{l \in [m]} s_{\tau(j)} \bar{r}'_l \bar{f}'_{n_c-1}(\rho(j)) = s_{\tau(j)} \bar{r} \bar{f}'_{n_c-1}(\rho(j))$$

for each $j \in \overline{\Upsilon}'$, so in turn we can retrieve

$$r' c_{1,j} - s'_j \bar{k}'' + \bar{r} c_{2,j} - s_{\tau(j)} \bar{r} \bar{f}'_{n_c-1}(\rho(j)) = r' \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal.$$

Then, for all $j \in \Upsilon'$, we compute

$$r' c_{1,j} - s'_j k'' + r c_{2,j} - s_{\tau(j)} k_{1,\rho(j)} + r_{\iota(\rho(j))} c_{3,j} = r' \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal.$$

Finally, we retrieve

$$sk' - \sum_{j \in \Upsilon} \varepsilon_j r' \mathbf{A}_j (sb, \hat{\mathbf{s}})^\intercal = \alpha s.$$

Performance analysis of the selectively secure instantiation. If we assume that $|\mathcal{S}|$ can take on any positive value, then it is best to put all non-lone variables in $\mathbb{H}$ and the polynomials in $\mathbb{G}$. If $|\mathcal{S}|$ is always going to be small compared to $|\Upsilon|$, then it is better to put all ciphertext components in $\mathbb{G}$ and the key components in $\mathbb{H}$.

To analyze the decryption costs more properly, we re-order the computations for $j \in \overline{\Upsilon}'$:

$$\sum_{j\in\overline{\Upsilon}'} \varepsilon_j \Big(- \sum_{l\in[m],\mathrm{att}\in\Psi_l} \Lambda_{\Psi'_l,x_{\mathrm{att}},\rho(j)} \Big(- \sum_{\mathrm{att}'\in\Psi_l} \Lambda_{\Omega_{j,l},x_{\mathrm{att}'},0} s_{\tau(j)} \bar{k}_{2,\mathrm{att}'} + \sum_{j'\in\chi_j} \Lambda_{\Omega_{j,l},\rho(j'),0} \bar{r}_l \bar{c}_{3,j'} + s_{\tau(j)} \bar{k}_{1,\mathrm{att}} \Big) \Big)$$

$$= \sum_{l'\in[m']} s_{l'} \sum_{j\in\overline{\Upsilon}'\cap\tau^{-1}(l')} \left(\sum_{\mathrm{att}'\in\mathcal{S}} \left(\sum_{\mathrm{att}\in\Psi_{\iota(\mathrm{att}')}} \varepsilon_j \Lambda_{\Psi'_{\iota(\mathrm{att}')},x_{\mathrm{att}},\rho(j)} \Lambda_{\Omega_{j,\iota(\mathrm{att}')},x_{\mathrm{att}'},0} \right) \bar{k}_{2,\mathrm{att}'} - \sum_{\mathrm{att}\in\mathcal{S}} \varepsilon_j \Lambda_{\Psi'_{\iota(\mathrm{att})},x_{\mathrm{att}},\rho(j)} \bar{k}_{1,\mathrm{att}} \right)$$

$$- \sum_{l\in[m]} \bar{r}_l \sum_{j'\in[n_1]} \left(\sum_{j\in\overline{\Upsilon}'\cap\chi_{j'},\mathrm{att}\in\Psi_l} \varepsilon_j \Lambda_{\Psi'_l,x_{\mathrm{att}},\rho(j)} \Lambda_{\Omega_{j,l},\rho(j'),0}) \right) \bar{c}_{3,j'}$$

which costs $m + \left\lceil \frac{|\overline{\Upsilon}'|}{n_c} \right\rceil$ pairing operations, and approximately $m \left\lceil \frac{|\overline{\Upsilon}'|}{n_c} \right\rceil (2n_k + n_c)$ exponentiations in $\mathbb{G}$. If $|\mathcal{S}|$ is small compared to $|\overline{\Upsilon}'|$, we can also compute it as

$$\sum_{\mathrm{att}\in\mathcal{S}} \bar{k}_{2,\mathrm{att}'} \left(\sum_{j\in\overline{\Upsilon}',\mathrm{att}'\in\Psi_{\iota^{-1}(\mathrm{att})}} \varepsilon_j \Lambda_{\Psi'_{\iota(\mathrm{att})},x_{\mathrm{att}},\rho(j)} \Lambda_{\Omega_{j,\iota(\mathrm{att})},x_{\mathrm{att}},0} s_{\tau(j)} \right) - \sum_{\mathrm{att}\in\mathcal{S}} \bar{k}_{1,\mathrm{att}} \left(\sum_{j\in\overline{\Upsilon}'} \varepsilon_j \Lambda_{\Psi'_{\iota(\mathrm{att})},x_{\mathrm{att}},\rho(j)} s_{\tau(j)} \right)$$

$$- \sum_{l\in[m]} \bar{r}_l \sum_{j'\in[n_1]} \left(\sum_{j\in\overline{\Upsilon}'\cap\chi_{j'},\mathrm{att}\in\Psi_l} \varepsilon_j \Lambda_{\Psi'_l,x_{\mathrm{att}},\rho(j)} \Lambda_{\Omega_{j,l},\rho(j'),0}) \right) \bar{c}_{3,j'},$$

which costs $m + 2|\mathcal{S}|$ pairing operations, and approximately $m \left\lceil \frac{|\overline{\Upsilon}'|}{n_c} \right\rceil (2n_k + n_c)$ exponentiations in $\mathbb{G}$.

Then, the costs are (in the best case, assuming that the negations can be distributed optimally over the ciphertext partitions):

- KeyGen: $3 + 2m + |\mathcal{S}|$ exponentiations in $\mathbb{H}$ and $3 + 3|\mathcal{S}|$ exponentiations in $\mathbb{G}$;
- Encrypt: $|\Upsilon| + m' + 1$ exponentiations in $\mathbb{H}$, $n_1(n_c + n + 4)$ exponentiations in $\mathbb{G}$, and 1 exponentiation in $\mathbb{G}_T$;
- Decrypt: $\min\left(am + \left\lceil \frac{|\Upsilon|}{n_c} \right\rceil, a(m + 2|\mathcal{S}|) + \left\lceil \frac{|\Upsilon'|}{n_c} \right\rceil\right) + \left\lceil \frac{|\Upsilon'|}{n_k} \right\rceil + 4$ pairing operations, and $am \left\lceil \frac{|\overline{\Upsilon}'|}{n_c} \right\rceil (2n_k + n_c)$ exponentiations in $\mathbb{G}$, where $a = 1$ if $|\overline{\Upsilon}'| > 0$, and $a = 0$ otherwise.

For encrypt and decrypt, the costs are higher when the attributes associated with the negations cannot be distributed optimally over the ciphertext partitions. In the worst case, each negated attribute incurs 2 exponentiations in $\mathbb{H}$ and n_c+n+4 in $\mathbb{G}$ in Encrypt. For decryption, suppose that the policy consists of negated attributes only, and none of them can be placed in the same partition. Then, the decryption costs are $4 + |\Upsilon| + \left\lceil \frac{|\Upsilon'|}{n_k} \right\rceil$ pairing operations and $2|\Upsilon| \cdot |\mathcal{S}| + \left\lceil \frac{|\Upsilon'|}{n_k} \right\rceil \cdot |\Upsilon|$ exponentiations.

G Fully secure instantiation of GLUE

We instantiate our pair encoding scheme in Section 4.1 in the Agrawal-Chase [3] framework with the prime-order dual system groups of Chen and Wee [24] with $d = 1$ (i.e., the underlying assumption in the security proof is the SXDH assumption).

Definition 14 (Fully secure version of GLUE). *The scheme is defined as follows.*

- Setup(λ)*: Taking as input the security parameter λ, the setup generates three groups $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ of prime order p with generators $g \in \mathbb{G}, h \in \mathbb{H}$, and chooses a pairing $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. The setup also defines the universe of attributes $\mathcal{U} = \mathbb{Z}_p$. It also chooses $n_k \in \mathbb{N}$ and $n_c \in \mathbb{N}$ as the maximum partition sizes of the keys and ciphertexts, respectively, and sets $n = n_k + n_c - 1$. It then generates random $\alpha_1, \alpha_2, a_1, a_2, a_3, a_4, d_1, d_2, ..., d_5, b_{i,1}, b_{i,2}, b'_{i',1}, b'_{i',2} \in_R \mathbb{Z}_p$ for all $i \in [0, n], i' \in [0, n_c - 1]$ such that $d_1 d_4 \neq d_2 d_3$. It outputs*

$$\text{MSK} = (\alpha_1, \alpha_2, a_1, ..., a_4, d_1, ..., d_5, \{b_{i,1}, b_{i,2}\}_{i\in[0,n]}, \{b'_{i,1}, b'_{i,2}\}_{i\in[0,n_c-1]})$$

as its master secret key and publishes the master public key as

$$\begin{gathered}\text{MPK} = (g, A = e(g,g)^{\alpha_1 d_1 + \alpha_2 d_2}, g_1 = g^{d_1}, g_2 = g^{d_2},\\ B_1 = g^{a_1 d_1 + a_3 d_3}, B_2 = g^{a_1 d_2 + a_3 d_4},\\ \{B_{i,1} = g^{b_{i,1} d_1 + b_{i,3} d_3}, B_{i,2} = g^{b_{i,1} d_2 + b_{i,3} d_4}\}_{i\in[0,n]},\\ \{B'_{i,1} = g^{b'_{i,1} d_1 + b'_{i,3} d_3}, B'_{i,2} = g^{b'_{i,1} d_2 + b'_{i,3} d_4}\}_{i\in[0,n_c-1]}).\end{gathered}$$

- KeyGen(MSK, $\mathcal{S}$)*: On input a set of attributes $\mathcal{S}$, the algorithm computes $m = \left\lceil \frac{|\mathcal{S}|}{n_k} \right\rceil$, defines $\iota\colon \mathcal{S} \to [m]$ such that $|\iota^{-1}(i)| \leq n_k$ for each $i \in [m]$, and generates random integers $r, r_1, ..., r_m \in_R \mathbb{Z}_p$ and computes the secret key as*

$$\text{SK}_{\mathcal{S}} = (\{K_\beta = h^{\alpha_\beta - r\bar{b}_\beta}, K'_1 = h^{r d_4 d_5 d_6}, K'_2 = h^{-r d_3 d_5 d_6}, \iota,$$

$$\{K_{1,\mathrm{att},\beta} = h^{r_{\iota(\mathrm{att})}(\sum_{i=0}^{n}\bar{b}_{i,\beta}x_{\mathrm{att}}^{i})+r\sum_{i=0}^{n_c-1}\bar{b}'_{i,\beta}x_{\mathrm{att}}^{i}}\}_{\mathrm{att}\in\mathcal{S}},$$
$$\{K_{2,l,1} = h^{r_l d_4 d_5 d_6}, K_{2,l,2} = h^{r_l d_3 d_5 d_6}\}_{l\in[m]}\}_{\beta\in\{1,2\}}),$$

where for all $i \in [0,n], i' \in [0, n_c - 1]$*, we set*

$$d_6 = \frac{1}{d_1 d_4 - d_2 d_3},$$
$$\bar{b}_1 = d_6(a_1 d_4 d_5 - a_2 d_2 d_5), \bar{b}_2 = d_6(-a_1 d_3 d_5 + a_2 d_1 d_5),$$
$$\bar{b}_{i,1} = d_6(b_{i,1} d_4 d_5 - b_{i,2} d_2 d_5), \bar{b}_{i,2} = d_6(-b_{i,1} d_3 d_5 + b_{i,2} d_1 d_5),$$
$$\bar{b}'_{i,1} = d_6(b'_{i,1} d_4 d_5 - b'_{i,2} d_2 d_5), \bar{b}'_{i,2} = d_6(-b'_{i,1} d_3 d_5 + b'_{i,2} d_1 d_5).$$

– Encrypt$(M, \mathrm{MPK}, \mathbb{A})$*: A message* $M \in \mathbb{G}_T$ *is encrypted under* $\mathbb{A} = (\mathbf{A}, \rho)$ *with* $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ *and* $\rho\colon [n_1] \to \mathcal{U}$ *by computing* $m' = \left\lceil \frac{n_1}{n_c} \right\rceil$ *and defining* $\tau\colon [n_1] \to [m']$ *such that* $|\tau^{-1}(i)| \leq n_c$ *for each* $i \in [m']$ *and if* $i, j \in [m']$ *with* $i \neq j$ *such that* $\rho(i) = \rho(j)$*, then* $\tau(i) \neq \tau(j)$*, i.e., multiple occurrences of the same attribute are mapped to different partitions. The user then generates random integers* $s, s_1, ..., s_{m'}, v_2, ..., v_{n_2} \in_R \mathbb{Z}_p$ *and computes the ciphertext as*

$$\mathrm{CT}_{\mathbb{A}} = (C = M \cdot A^s, C'_1 = g^{sd_1}, C'_2 = g^{sd_2}, \tau,$$
$$\{C_{1,j,\beta} = B_\beta^{A_{j,1}s} g_\beta^{\bar{\lambda}_j} \cdot \prod_{i=0}^{n_c-1} (B'_{i,\beta})^{s_{\tau(j)}\rho(j)^i},$$
$$C_{2,j,\beta} = \prod_{i=0}^{n} B_{i,\beta}^{s_{\tau(j)}\rho(j)^i}\}_{j\in[n_1],\beta\in\{1,2\}}, \{C_{3,l',\beta} = g^{s_{l'}d_\beta}\}_{l'\in[m'],\beta\in\{1,2\}}),$$

such that $\bar{\lambda}_j = \sum_{k\in[2,n_2]} A_{j,k} v_k$*.*

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

$$C/\Big(e(C', K) \cdot \textstyle\prod_{j\in\Upsilon,\beta\in\{1,2\}} \Big(e(C_{1,j,\beta}, K'_\beta)/e(C_{3,\tau(j),\beta}, K_{1,\rho(j),\beta})$$
$$\cdot e(C_{2,j,\beta}, K_{2,\iota(\rho(j)),\beta})\Big)^{\varepsilon_j}\Big).$$

H AHM+16 and ABGW17 in the asymmetric setting

We give the definitions of the AHM+16 [12] and the ABGW17 [7] schemes in the asymmetric setting. In particular, we distribute the key and ciphertext components such that the encryption and decryption algorithms are optimized (possibly at the cost of the key generation efficiency). The general approach is fairly simple: we try to put as many ciphertext components in the first source group as possible. Alternatively, if the decryption algorithm requires exponentiations, then we place the components that need to be exponentiated in the first source group.

H.1 The AHM+16 scheme

We define the AHM+16 [12] scheme in the ciphertext-policy and selective security setting with optimized decryption as follows.

Definition 15 (The AHM+16 scheme). *The AHM+16 scheme is defined as follows.*

- Setup(λ)*: Taking as input the security parameter λ, the setup generates three groups $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ of prime order p with generators $g \in \mathbb{G}, h \in \mathbb{H}$, and chooses a pairing $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. The setup also defines the universe of attributes $\mathcal{U} = \mathbb{Z}_p$. It also chooses $n_k \in \mathbb{N}$ as the maximum partition sizes of the keys. It then generates random $\alpha, b, b_0, b_1, ..., b_{n_k}, b'_1, b'_2, b'_3 \in_R \mathbb{Z}_p$. It outputs $\mathrm{MSK} = (\alpha, b, b_0, b_1, ..., b_{n_k}, b'_1, b'_2, b'_3)$ as its master secret key and publishes the master public key as*

$$\mathrm{MPK} = (g, h, A = e(g,h)^{\alpha}, B = g^{b}, B'_1 = g^{b'_1}, B'_2 = g^{b'_2}, B'_3 = g^{b'_3}, \\ B_0 = g^{b_0}, ..., B_n = g^{b_n}).$$

- KeyGen($\mathrm{MSK}, \mathcal{S}$)*: On input a set of attributes $\mathcal{S}$, compute $m = \left\lceil \frac{|\mathcal{S}|}{n_k} \right\rceil$, define $\iota\colon \mathcal{S} \to [m]$ such that $|\iota^{-1}(i)| \leq n_k$ for each $i \in [m]$, generate random integers $r, r', r_1, ..., r_m \in_R \mathbb{Z}_p$ and compute the secret key as*

$$\mathrm{SK}_{\mathcal{S}} = (K = h^{\alpha - rb - r'b'_1}, K' = h^{r}, K'' = h^{r'}, K^{(3)} = h^{r'b'_3}, \iota, \\ \{K_{1,l} = h^{r'b'_2 + r_{\iota(\mathrm{att})}(\sum_{i=0}^{n_k} b_i x_{\mathrm{att}}^i)}, K_{2,l} = h^{r_l}\}_{l \in [m]}).$$

- Encrypt($M, \mathrm{MPK}, \mathbb{A}$)*: A message $M \in \mathbb{G}_T$ is encrypted under $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ and $\rho\colon [n_1] \to \mathcal{U}$ by generating random integers $s, s', s_1, ..., s_{n_1}, v_2, ..., v_{n_2} \in_R \mathbb{Z}_p$ and computing the ciphertext as*

$$\mathrm{CT}_{\mathbb{A}} = (C = M \cdot A^{s}, C' = g^{s}, C'' = g^{s'}, C^{(3)} = (B'_1)^{s}(B'_3)^{s'}, \\ \{C_{1,j} = B^{\lambda_j} \cdot (B'_2)^{s_j}, C_{2,j} = g^{s_j}, C_{3,j,0} = B_0^{s_j}\}_{j \in [n_1]}, \\ \{C_{3,j,i} = B_{i+1}^{s_j} B_1^{-s_j \rho(j)^i}\}_{i \in [n_k], j \in [n_1]}),$$

  such that λ_j denotes the j-th entry of $\mathbf{A} \cdot (s, v_2, ..., v_{n_2})^{\intercal}$.
- Decrypt($\mathrm{SK}_{\mathcal{S}}, \mathrm{CT}_{\mathbb{A}}$)*: Suppose that $\mathcal{S}$ satisfies $\mathbb{A}$, and suppose $\Upsilon = \{j \in [n_1] \mid \rho(j) \in \mathcal{S}\}$, such that $\{\varepsilon_j \in \mathbb{Z}_p\}_{j \in \Upsilon}$ exist with $\sum_{i \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$ (Definition 1). Let $f_{n_k,l}(x) = \prod_{\mathrm{att} \in \iota^{-1}(l)} \frac{x - x_{\mathrm{att}}}{-x_{\mathrm{att}}} = \sum_{i=0}^{n_k} c_{i,l} x^i$ denote, for each $l \in [m]$, the polynomial with roots in $\iota^{-1}(l)$. Then the plaintext M is retrieved by computing*

$$C / \Big(e(C', K) \cdot e((C^{(3)})^{\varepsilon_j}, K'') \cdot e(C''^{-\varepsilon_j}, K^{(3)}) \cdot \prod_{j \in \Upsilon} \Big(e(C_{1,j}^{\varepsilon_j}, K') \\ \cdot e(C_{2,j}^{-\varepsilon_j}, K_{1,\iota(\rho(j))}) \cdot e(\prod_{i \in [\overline{n_k}]} C_{3,j,i}^{-\varepsilon_j c_{i,\iota(\rho(j))}}, K_{2,\iota(\rho(j))}) \Big) \Big).$$

Performance analysis of the selectively secure instantiation.

- KeyGen: $4 + 2|\mathcal{S}|$ exponentiations in $\mathbb{H}$;
- Encrypt: $2 + 2n_1$ exponentiations in $\mathbb{G}$, $1 + n_1 n_k$ two-base exponentiations in $\mathbb{G}$, and 1 exponentiation in $\mathbb{G}_T$;
- Decrypt: roughly $4 + 2\left\lceil \frac{|\Upsilon|}{n_k} \right\rceil$ pairing operations, and $|\Upsilon|$ multi-base exponentiations with n_k bases in $\mathbb{G}$.

H.2 The ABGW17 scheme

We define the ABGW17 [7] scheme in the ciphertext-policy and selective security setting with optimized encryption as follows.

Definition 16 (The ABGW17 scheme). *The ABGW17 scheme is defined 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 $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. The setup also defines the universe of attributes $\mathcal{U} = \mathbb{Z}_p$. It then generates random $\alpha, b, b_0, b_1, b' \in_R \mathbb{Z}_p$. It outputs* $\mathrm{MSK} = (\alpha, b, b_0, b_1, b')$ *as its master secret key and publishes the master public key as*

$$\mathrm{MPK} = (g, h, A = e(g,h)^{\alpha}, B = g^{b}, B_0 = g^{b_0}, B_1 = g^{b_1}, B' = g^{b'}).$$

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

$$\mathrm{SK}_{\mathcal{S}} = (K = h^{\alpha - rb}, K' = h^{r}, \{K_{\mathrm{att}} = h^{\frac{rb'}{x_{\mathrm{att}} b_1 + b_0}}\}_{\mathrm{att} \in \mathcal{S}}).$$

- Encrypt($M, \mathrm{MPK}, \mathbb{A}$)*: A message $M \in \mathbb{G}_T$ is encrypted under access policy $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ and $\rho\colon [n_1] \to \mathcal{U}$ by generating random integers $s, s_1, ..., s_{n_1}, v_2, ..., v_{n_2} \in_R \mathbb{Z}_p$ and computing the ciphertext*

$$\mathrm{CT}_{\mathbb{A}} = (C = M \cdot A^{s}, \{C_{1,j} = B^{\lambda_j}(B')^{s_j},$$
$$C_{2,j} = (B_1^{\rho(j)} B_0)^{s_{\tau(j)}}, C_{3,j} = g^{\lambda_j}\}_{j \in [n_1]}),$$

  such that λ_j denotes the j-th entry of $\mathbf{A} \cdot (s, v_2, ..., v_{n_2})^{\intercal}$.
- Decrypt($\mathrm{SK}_{\mathcal{S}}, \mathrm{CT}_{\mathbb{A}}$)*: 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 $\sum_{i \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$ (Definition 1). Then the plaintext M is retrieved by computing*

$$C / \Big(e(\textstyle\prod_{j \in \Upsilon} C_{3,j}^{\varepsilon_j}, K) \cdot e(\prod_{j \in \Upsilon} C_{1,j}^{\varepsilon_j}, K') / \prod_{j \in \Upsilon} e(C_{2,j}^{\varepsilon_j}, K_{\rho(j)}) \Big).$$

Performance analysis of the selectively secure instantiation.

- KeyGen: $2 + |\mathcal{S}|$ exponentiations in $\mathbb{H}$;
- Encrypt: n_1 exponentiations in $\mathbb{G}$, $2n_1$ two-base exponentiations in $\mathbb{G}$, and 1 exponentiation in $\mathbb{G}_T$;
- Decrypt: roughly $2 + |\Upsilon|$ pairing operations.

Table 4: The performance of various algorithms on the BLS12-446 elliptic curve, expressed in the number of 10^3 cycles.

Algorithm	In $\mathbb{G}$	In $\mathbb{H}$	In $\mathbb{G}_T$
Exponentiation	926	2076	3101
Fixed-base exponentiation	485	945	-
Multi-base exponentiation	926+	2076+	-
($n+1$ bases)	$317n$	$1384n$	
Hash	1163	2933	-

Pairing operation	
Single	6259
Product of $n+1$	$6259 + 1462n$

Table 5: The actual versus estimated costs of the key generation and encryption algorithms of RW13 on the BLS12-446 elliptic curve, expressed in the number of 10^3 cycles.

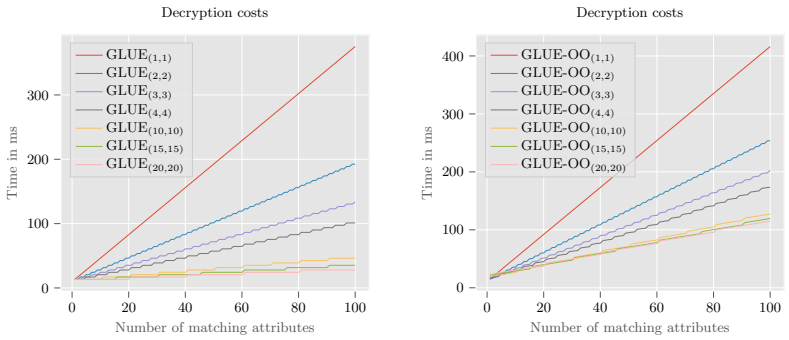
	KeyGen			Encrypt		
	1	10	100	1	10	100
Actual	1520	8376	76853	1697	9045	82362
Estimated	1504	8272	75952	1682	8792	79892
Difference	1%	1%	1%	1%	3%	3%

I Benchmarks in RELIC

Our benchmarks of several algorithms (e.g., for exponentiation) in RELIC are summarized in Table 4. The costs are expressed in the number of cycles, but can be effectively converted in milliseconds by considering the clock frequency of the device: 1.6 GHz.

J Accuracy of our estimates

Although our estimates in Section 6 are approximated theoretically by benchmarking various algorithms and extrapolating the costs in the schemes, we argue that they are realistic. To show this, we approximate the costs of RW13-OE (the variant of RW13 with an optimized encryption, which is the same as our variant of RW13) based on the benchmarks of various algorithms in Table 1 of [46]. We then compare the estimated costs with the actual costs. Note that we only compare the key generation and encryption costs, because our estimation of the decryption costs assumes that $\varepsilon_j \in \{0, 1\}$, which is not the case in the implementation of the access structures in [46]. Table 5 indeed shows that our estimates are very close to the actual costs of RW13. Note that, because the measurements in Table 1 of [46] are rounded, a part of the difference between the costs can be attributed to rounding errors.



(a) Regular version (b) Online/offline version

Fig. 3: Decryption costs of our regular and online/offline schemes supporting MSPs in milliseconds.

K On choosing suitable partition sizes

We briefly discuss how suitable partition sizes can be chosen (if nothing is known about the application for which it will be used). This can be done by analyzing the decryption costs of the algorithm for various choices of $n_k = n_c$ (where the encryption-decryption trade-off is the most optimal). In Figure 3, the decryption efficiency of the regular and online/offline variant of our schemes supporting MSPs are depicted. As these graphs show, the decryption costs of the regular variant decrease as n_k and n_c increase. For the online/offline variant, this is not the case. In fact, we see that for smaller sets of attributes, decryption for $n_k = n_c > 10$ is several milliseconds slower than the case that $n_k = n_c = 5$. Only for larger sets of attributes, these start to (slightly) outperform $n_k = n_c = 5$. Furthermore, the cases that $n_k = n_c \in \{1, 2\}$ are only faster for one or two attributes, and then they increase very fast in costs. As such, $n_k = n_c = 5$ seems to provide, on average, the most efficient decryption algorithm.

L Towards analyzing the efficiency of OSWOT-type non-monotone schemes

To obtain schemes supporting OSWOT-type negations, like proposed in [15], one can first apply a Boneh-Boyen or full-domain hash (as proposed in Section F.1) for the labeled universes, and then an unbounded scheme supporting NMSPs for the attributes in that universe. In this way, during decryption, the decrypting user only needs to compare the secret key attributes with the negated attribute in the policy that share the same label. For many conceivable attribute labels—e.g., name, address of residence, profession, age—we expect the user to have only a few attributes. In this case, any subset $\mathcal{S}' \subseteq \mathcal{S}$ with the same label is small, i.e., $|\mathcal{S}'| \approx 1$. To see how instantiating our schemes compare to instantiating

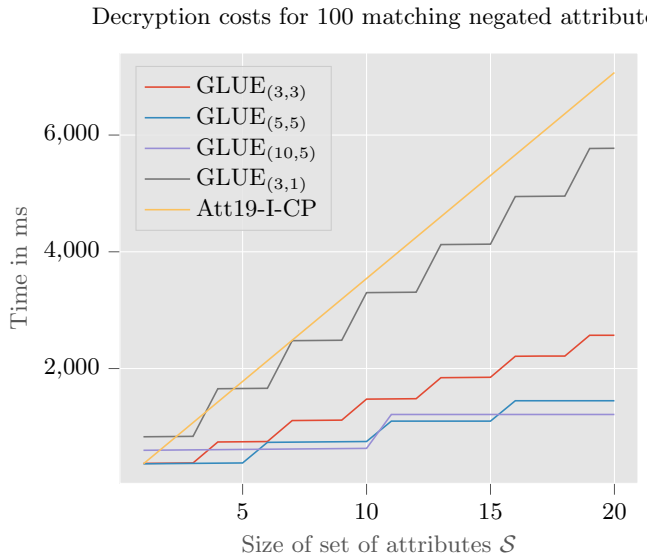


Fig. 4: Decryption costs of the schemes supporting OSW-type negations, for various $|\mathcal{S}|$, for 100 matching attributes.

Att19-I-CP [11], we also consider the specific cases in which the set of attributes $|\mathcal{S}| = 1$ in Table 3. On the other hand, for some labels, such as "departments at a hospital" or "courses provided by a university", users may possess many attributes. It is therefore paramount that a scheme also performs efficiently for the cases in which $\mathcal{S}'$ is larger, say, for $|\mathcal{S}|' \approx 5$. Thus, we also consider the specific case where $|\mathcal{S}| = 5$ in Table 3. In general, it is valuable to consider both small and large quantities of $|\mathcal{S}|$, as it illustrates the impact of OSW-type negations on the decryption efficiency (see Figure 4 for a more detailed efficiency analysis). Compared to OSW-type negations, using OSWOT-type negations would reduce the decryption costs for negations, but might increase the decryption costs for non-negated attributes and the encryption costs. To reduce the encryption costs, the online/offline versions of our schemes can ensure that the encryption costs (with only slide trade-off in the decryption efficiency), so the total efficiency of the scheme is high. Note, however, that this requires the use of a Boneh-Boyen hash for the labels, and not a full-domain hash.

M Performances in figures

We elaborate on the performance analysis of the schemes analyzed in Section 6.

Storage costs. In Figures 5 and 6, the storage costs of the keys and ciphertexts are depicted for various input sizes, for the regular schemes and the online/offline

schemes, respectively. For the online/offline schemes, we have listed the sizes of the intermediate keys that are stored by the authority that generates keys. We assume that the authority stores at least one intermediate key of the first type for every 10 attributes. The tables show that our regular scheme's ciphertexts are smaller than those of RW13, AHM+16 and ABGW17. On the other hand, the ciphertext sizes of the our online/offline scheme are larger by a factor 2-3 than those of HW14, and by a factor 4-5 compared to our regular scheme.

Computational costs. In Figure 7, the computational costs of the key generation and encryption algorithms are depicted for various input sizes. As expected, the encryption costs of our scheme are higher than those of the other schemes. If these are considered to be too high for some specific practical setting, one could choose to use the online/offline variant (which would increase the ciphertext sizes). In Figures 8, 9 and 10, the computational costs of the decryption algorithm are depicted, for various sizes of the set of matching attributes. These illustrate that, for larger sizes, our decryption algorithms perform even better compared to the other schemes than for the smaller sizes.

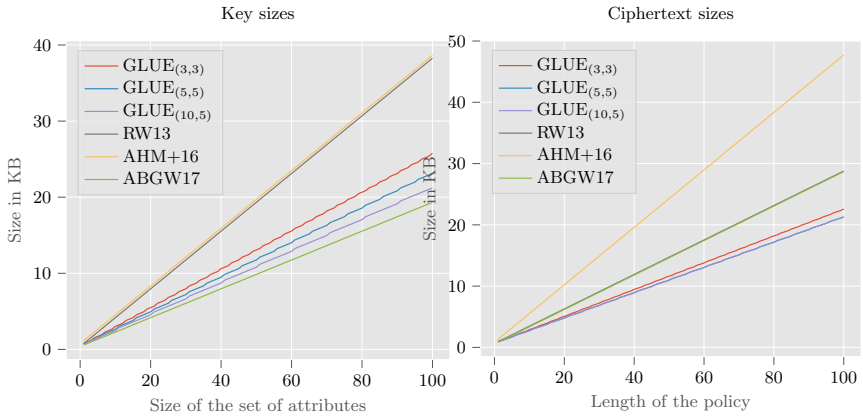


Fig. 5: Storage costs of the regular schemes (1 KB = 1024 bytes)

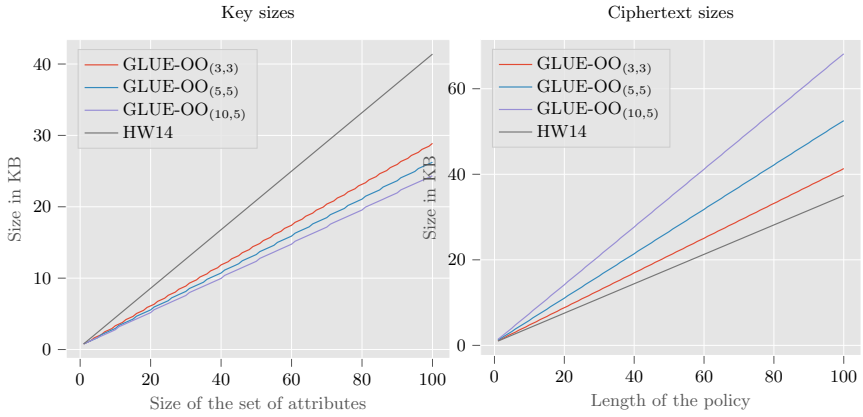
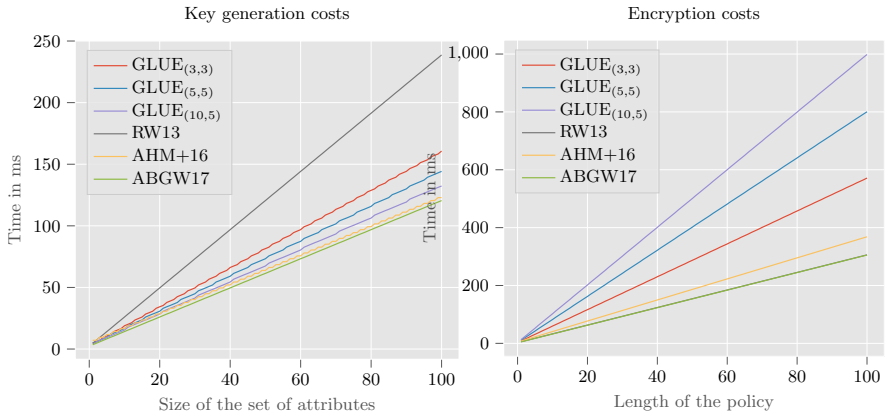


Fig. 6: Storage costs of the online/offline schemes (1 KB = 1024 bytes)



(a) Key generation (b) Encryption

Fig. 7: The computational costs of key generation and encryption of the regular schemes.

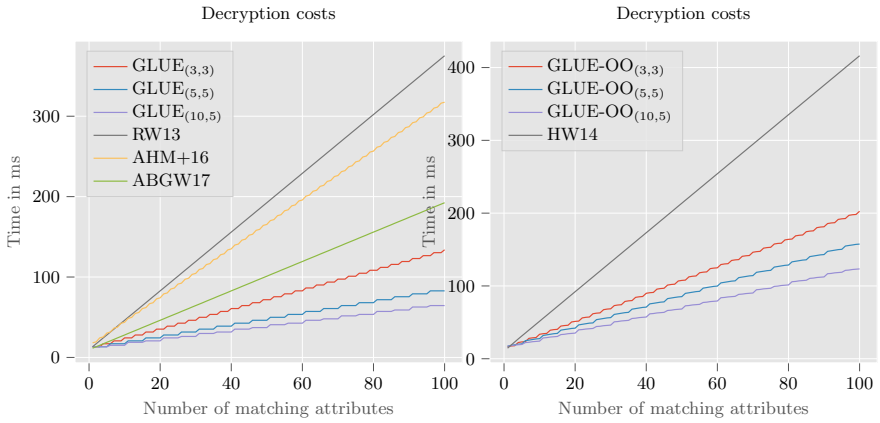
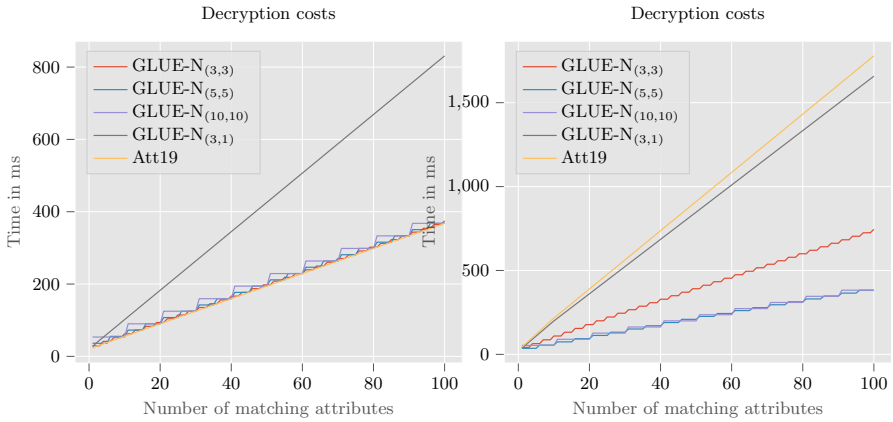
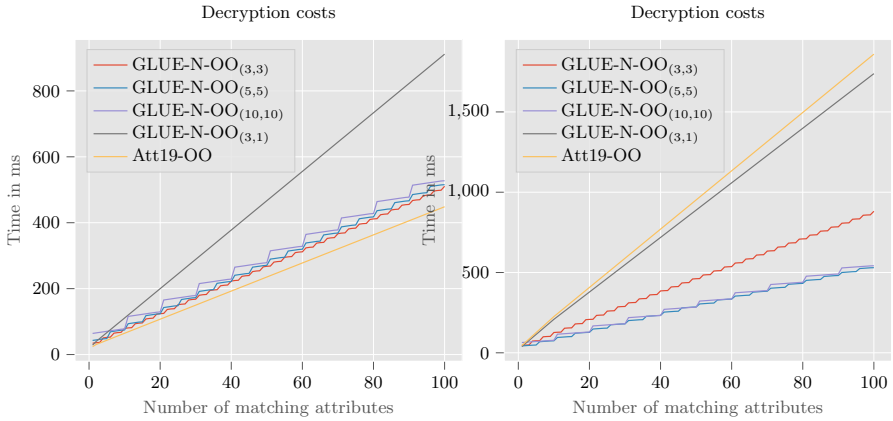


Fig. 8: Decryption costs of the schemes supporting MSPs



(a) $|\mathcal{S}| = 1$ (b) $|\mathcal{S}| = 5$

Fig. 9: Decryption costs of the regular schemes supporting NMSPs, all negations



(a) $|\mathcal{S}| = 1$ (b) $|\mathcal{S}| = 5$

Fig. 10: Decryption costs of the online/offline schemes supporting NMSPs, all negations

Table of Contents