TinyABE: Unrestricted Ciphertext-Policy Attribute-Based Encryption for Embedded Devices and Low-Quality Networks

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 (CP-ABE) has attracted much interest from the practical community to enforce access control in distributed settings such as the Internet of Things (IoT). In such settings, encryption devices are often constrained, having small memories and little computational power, and the associated networks are lossy. To optimize both the ciphertext sizes and the encryption speed is therefore paramount. In addition, the master public key needs to be small enough to fit in the encryption device's memory. At the same time, the scheme needs to be expressive enough to support common access control models. Currently, however, the state of the art incurs undesirable efficiency trade-offs. Existing schemes often have linear ciphertexts, and consequently, the ciphertexts may be too large and encryption may be too slow. In contrast, schemes with small ciphertexts have extremely large master public keys, and are generally computationally inefficient.
In this work, we propose TinyABE: a novel CP-ABE scheme that is expressive and can be configured to be efficient enough for settings with embedded devices and low-quality networks. In particular, we demonstrate that our scheme can be configured such that the ciphertexts are small, encryption is fast and the master public key is small enough to fit in memory. From a theoretical standpoint, the new scheme and its security proof are non-trivial generalizations of the expressive scheme with constant-size ciphertexts by Agrawal and Chase (TCC'16, Eurocrypt'17) and its proof to the unbounded setting. By using techniques of Rouselakis and Waters (CCS'13), we remove the restrictions that the Agrawal-Chase scheme imposes on the keys and ciphertexts, making it thus more flexible. In this way, TinyABE is especially suitable for IoT.

Keywords: attribute-based encryption · ciphertext-policy attribute-based encryption · short ciphertexts · efficient encryption

1 Introduction

Attribute-based encryption (ABE) is an advanced type of public-key encryption in which the key pairs are associated with attributes rather than individuals [46]. In *ciphertext-policy ABE* (CP-ABE), messages are encrypted under an

access policy [16]. Subsequently, the ciphertexts can be decrypted by a single secret key that is associated with a set of attributes that satisfies the policy. In contrast, in *key-policy ABE* (KP-ABE), ciphertexts are associated with sets of attributes and secret keys with access policies [30]. CP-ABE has proven to be a valuable primitive in the enforcement of fine-grained access control on a cryptographic level [16,35,47]. It allows the encrypting device to determine who gets access to the plaintext, without requiring an online trusted third party to act as an intermediary [52]. Instead, it requires a trusted entity to issue secret keys to eligible users, which can be used to access the data for which those are authorized. In this way, the device can directly and securely share its data via any (potentially untrusted) network. Recently, the European Telecommunications Standards Institute (ETSI) has published two specifications regarding the high-level requirements for ABE [26], and how ABE can increase data security and privacy [27]. In these specifications, ETSI focuses on several use cases, one of which considers data access control in the Internet of Things (IoT), in particular. An important requirement that ETSI imposes on ABE is that an IoT device should be able to encrypt, but not necessarily decrypt. To this end, the public keys and ciphertexts should be small, and encryption should be efficient. ETSI also requires the scheme to support expressive policies [26]. Such policies include Boolean formulas, consisting of AND and OR gates, over attributes; and the attributes may be strings or numerical values. The policies may also be large, because they could specify that a decryption key should be generated within a certain time interval, whose description may require several attributes.

According to RFC8576[3], IoT devices and networks are characterized by small memory, low computational power, and high packet loss rates. Unfortunately, many ABE schemes—including those considered by ETSI [26]—have ciphertexts sizes and encryption costs that grow linearly in the number of attributes [2,6]. As a result, these schemes are not suitable for IoT applications [28]. First, encryption may simply consume too much time, requiring almost one second per attribute [48]. Second, even for small policies, the ciphertexts may be so large that they have to be fragmented across more than one data packet during transmission. This results in an increased probability that at least one of the packets is dropped, and subsequently increases the expected time that it takes for the message to successfully arrive at the receiver [42]. Third, the ciphertext may not fit in memory. The computation of one ciphertext would therefore need to be split into parts, and the partial ciphertexts need to be streamed out of the device, like in [34]. This may further complicate issues with packet loss.

To mitigate issues with the size, ABE schemes with sufficiently short ciphertexts can be deployed. Several schemes with constant-size ciphertexts have been proposed [25,32,22,12,11,1]. However, many of these schemes have restricted policies [25,32,22], supporting only AND-gates or threshold functions, and therefore have a limited expressivity. Others are bounded [12,11,1], supporting only limited sizes for the sets or policies associated with the ciphertexts. More importantly, the efficiency of these bounded schemes depends heavily on the bounds. Hence,

[3] https://tools.ietf.org/html/rfc8576

choosing these bounds to be sufficiently high for some given practical setting is not a suitable option either.

In this work, we mitigate these limitations by proposing a scheme with a trade-off feature. Upon setup, the system parameters can be chosen such that the desired efficiency trade-off between the sizes of the keys and the ciphertexts, as well as the computational costs of the algorithms can be attained. In particular, one can optimize encryption so that it can be performed on IoT devices. Furthermore, one can configure the ciphertexts to be small enough for a specific setting, i.e., to fit in memory of IoT devices or in one Ethernet packet for some given number of attributes. One can also configure the master public key to be small enough to fit in memory. This makes TinyABE especially suitable for IoT.

1.1 Our contributions

Our main contribution is TinyABE, a new CP-ABE scheme that simultaneously can satisfy several desirable properties:

- **Expressivity:** The scheme supports monotone span programs (MSPs), which includes Boolean formulas consisting of both AND and OR gates;
- **Large-universeness:** Any string can be used as attribute;
- **Unboundedness:** No bounds are posed on the parameters, including the attribute sets associated with the keys and the policy lengths;
- **Configurable:** The system parameters can be chosen such that the scheme attains the required efficiency, for example
  - **Short ciphertexts:** The scheme can be configured such that the ciphertexts are sufficiently small for scenarios involving low-quality networks;
  - **Efficient encryption:** The scheme can be configured such that encryption is fast, even on resource-constrained devices.

We achieve this by making the expressive CP-ABE scheme with constant-size ciphertexts by Agrawal and Chase (AC16) [1] unbounded. As a result, our scheme is parametrized, and can be configured to provide the desired efficiency trade-off. Special cases of our scheme include AC16, and the CP-ABE scheme with constant-size ciphertexts by Attrapadung (Att19) [9]. TinyABE can thus be viewed as a generalization of AC16 to the unbounded setting. We also provide two secondary contributions:

- *Security proof:* We generalize Agrawal and Chase's [3] proof for AC16 [1] to the unbounded setting using Rouselakis and Waters' [45] techniques;
- *Performance analysis:* We analyze the efficiency of our scheme with a focus on practice. In particular, we obtain the most efficient encryption algorithm compared to other expressive and unbounded schemes;

2 High-level overview and details about TinyABE

Our construction. TinyABE is a generalization of the Agrawal-Chase scheme (AC16) [1,3] to the unbounded setting, using the partitioning techniques by Attrapadung et al. (AHM+16) [10], and by using the proof techniques by Rouselakis and Waters (RW13) [45]. By generalizing AC16, we can make it more

efficient. Although AC16 supports expressive policies and attains constant-size ciphertexts, it is bounded in parameters N_1 and N_2, where N_1 and N_2 denote the upper bounds on the number of rows and columns of the access structure, respectively. Importantly, the scheme's efficiency depends on these parameters. Whereas the ciphertext sizes are constant, the master public key grows by a factor $N_1 N_2$, and the secret keys grow by a factor $N_1^2 N_2$. As a result, the master public key is already so large for $N_1 = N_2 = 32$, i.e., 103 kilobytes (KB), that it does not fit in memory of many embedded devices. By making AC16 unbounded, the efficiency depends differently on these factors. We make the AC16 scheme unbounded by using a similar approach as AHM+16. Roughly, we partition the sets of rows and columns in smaller subsets of maximum sizes $\hat{n}_1$ and $\hat{n}_2$, respectively, and apply the AC16 scheme to the partitions. The master public key and secret keys then also grow in factors $\hat{n}_1 \hat{n}_2$ and $\hat{n}_1^2 \hat{n}_2$, respectively, but $\hat{n}_1$ and $\hat{n}_2$ can be much smaller to attain small ciphertexts. Although our ciphertexts are not constant-size, they shrink by a factor $\mathcal{O}(\min(\hat{n}_1, \hat{n}_2))$ compared to schemes with linear-size ciphertexts, such as RW13. Thus, even for small choices of $\hat{n}_1$ and $\hat{n}_2$, our ciphertexts are much smaller than RW13 ciphertexts. Whereas RW13 ciphertexts might only fit in memory or in one Ethernet packet for a maximum policy length of 33 or 3, respectively, TinyABE can support larger policy lengths. For example, in the same settings, it supports maximum policy lengths of 298 (for $\hat{n}_1 = \hat{n}_2 = 3$), and 100 (for $\hat{n}_1 = \hat{n}_2 = 13$), respectively, while the associated master public keys are only 2.3, and 19 KB, respectively.

Security proof: the AC17 framework. We formulate our scheme and proofs in the AC17 [3] framework, which considers a commonly-used abstraction of pairing-based encryption schemes: pair encoding schemes (PES) [8]. Essentially, a PES condenses a scheme to "what happens in the exponent". The AC17 framework simplifies security analysis, whilst achieving strong security guarantees, by reducing the effort of proving security to performing simple linear algebra [51]. In part, we use this framework, because we generalize AC16, and its only proofs in the full-security setting are given in this framework [3,9]. In contrast, other expressive CP-ABE schemes with constant-size ciphertexts [11,13] have larger keys than AC16 and are therefore less efficient. Furthermore, because we prove security in the AC17 framework, our scheme can be transformed into a scheme supporting negations in the policies [9,5], additionally allowing for the support of revocation systems [37].

Improving the partitioning approach. We improve on the partitioning approach used for the KP-ABE scheme of Attrapadung et al. (AHM+16) [10], which is unbounded, supports expressive policies, and can be configured to have small ciphertexts. Specifically, AHM+16 generalizes the first expressive KP-ABE scheme with constant-size ciphertexts of Attrapadung, Libert and de Panafieu (ALP11) [12] to the unbounded setting. Concretely, their approach consists of the partitioning of the attribute set (to be used during encryption) into subsets of maximum size n_k, where n_k is the bound on the attribute set inherited from

ALP11. Before our work, a CP-ABE scheme attaining similar characteristics remained an open problem. In fact, the first expressive CP-ABE schemes with constant-size ciphertexts [11,1] were proposed four years after the introduction of ALP11. Presumably, the reason for this delay is the difficulty in simultaneously achieving these properties in the ciphertext-policy setting. On the one hand, the entire access policy—which is two-dimensional—needs to be embedded in one ciphertext component. On the other hand, the decrypting user—who has an attribute set satisfying the policy—may not have keys for all attributes used in the access policy. These difficulties also translate to the unbounded setting: to make AC16 unbounded, we need to partition in two dimensions instead of one. In addition, we want to embed the entire policy in one ciphertext component, like AC16. This is unlike AHM+16, which embeds each partitioned subset in a separate ciphertext component, and thus still requires a linear number of operations during encryption. In contrast, the costs of computing our ciphertext component embedding the policy are essentially upper-bounded by a constant.

Performance analysis. We show that TinyABE offers advantages over other schemes by analyzing the storage and computational costs. In this analysis, we take into account the limitations of constrained devices and low-quality networks. To this end, we select two configurations of TinyABE, which we compare with RW13 and AC16. Our first configuration provides sufficiently small public keys and ciphertexts for IoT devices, whilst attaining an efficient encryption algorithm. For example, in Section 6.3, we estimate the encryption costs on some IoT devices. For policies of length 100, encryption with RW13 takes over a minute, while encryption with our scheme takes only 7.6 seconds. Moreover, while the master public key of AC16 is almost a megabyte in size, our master public key is only 2.25 kilobytes, and thus fits easily in memory of constrained devices. Our second configuration ensures that, for policy lengths of up to 100 attributes, the ciphertexts fit in one Ethernet packet, which has a maximum transmission unit of 1500 bytes. In contrast, RW13 ciphertexts are too large.

Expressive, large-universe, unbounded and efficient. TinyABE is simultaneously expressive and unrestricted while it is configurable. Therefore, it can be configured to be efficient enough for practical applications involving IoT devices and networks. Our scheme supports large universes, so it can efficiently support any strings as attributes, and does not require that, in the setup, public keys are generated for each attribute. The scheme is also unbounded[4], which implicitly ensures that it attains a better efficiency, even for large policies, compared to bounded schemes. In contrast, the efficiency of bounded schemes with constant-size ciphertexts [1,9] depends heavily on the choice of these bounds. Finally, because our scheme supports monotone span programs, it can enforce any fine-grained policies on encrypted data. Practitioners therefore do not need to restrict themselves to less expressive solutions in IoT settings anymore [36,28].

[4] Note that our scheme is also unbounded in that it satisfies the "multi-use" property, meaning that attributes may occur any number of times in the access policies.

Expressive and efficient CP-ABE scheme for IoT. Several schemes have been introduced over the years. Some can attain sufficiently short ciphertexts for some specific practical context. In particular, we consider schemes that can be configured to be small enough to fit e.g., in memory of constrained devices or in Ethernet packets, even for large policies. As Table 1a shows, all of the CP-ABE schemes of this kind incur a trade-off: either they are not expressive, or they impose bounds (and by extension, they are inefficient). In contrast, TinyABE is the first CP-ABE scheme to overcome these limitations. Furthermore, compared to expressive schemes with ciphertext sizes that grow at least in the size of the policy or set (see Table 1b), TinyABE can be configured to have a more efficient encryption. As such, it is feasible to implement ABE on IoT devices (see Section 6.3), which are mainly assumed to be required to encrypt and not decrypt.

3 Preliminaries

3.1 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.

3.2 Access structures

Definition 1 ((Monotone) access structures [15]). *Let $\{a_1, ..., a_n\}$ be a set of attributes. An access structure is a collection $\mathbb{A}$ of non-empty subsets of $\{a_1, ..., a_n\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets that are not in $\mathbb{A}$ are called the unauthorized sets. An access structure $\mathbb{A} \subseteq 2^{\{a_1,...,a_n\}}$ is monotone if for all B, C holds: if $B \in \mathbb{A}$ and $B \subseteq C$, then also $C \in \mathbb{A}$.*

We represent access policies $\mathbb{A}$ by linear secret sharing scheme (LSSS) matrices, which support monotone span programs [15,31]. In particular, Boolean formulas can be efficiently converted into LSSS matrices [38].

Definition 2 (Access structures represented by LSSS matrices [31]). *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 the rows of $\mathbf{A}$ to attributes in the universe. For some vector $\mathbf{v} = (s, v_2, ..., v_{n_2}) \in_R \mathbb{Z}_p^{n_2}$, the i-th secret generated by matrix $\mathbf{A}$ 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 thus, $\sum_{i \in \Upsilon} \varepsilon_i \lambda_i = s$ 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 $i \in \Upsilon$ [15].*

Table 1: Comparison of ABE schemes with short and linear ciphertexts, respectively. For each scheme, we list whether they are CP, the expressivity (expr.), whether they are large-universe (LU), and whether they support unbounded (unb) policies or sets. For the schemes with short ciphertexts, we also give the asymptotic complexity of the storage costs of their master public keys (MPK), secret keys (SK) and ciphertexts (CT). We consider a scheme to have short ciphertexts if their asymptotic sizes are smaller than linear in the number of attributes, i.e., $\mathcal{O}(|\mathcal{S}|)$ or $\mathcal{O}(|\mathbb{A}|)$. Note that we have only listed schemes that are structurally different, i.e., that have a different PES. For instance, the KP-ABE scheme in [45] has the same PES as the KP-ABE scheme in [39].

Scheme	**CP**	**Expr.**	**LU**	**Unb.**		**Sizes**		
				$\lvert\mathbb{A}\rvert$	$\lvert\mathcal{S}\rvert$	MPK	SK	CT
EMN+09 [25]	✓	AND	✗	✓	✓	$\mathcal{O}(\lvert\mathcal{U}\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
HLR10 [32]	✓	Threshold	✗	✓	✓	$\mathcal{O}(\lvert\mathcal{U}\rvert)$	$\mathcal{O}(\lvert\mathcal{U}\rvert)$	$\mathcal{O}(1)$
CZF11 [22]	✓	AND	✗	✓	✓	$\mathcal{O}(\lvert\mathcal{U}\rvert)$	$\mathcal{O}(\lvert\mathcal{U}\rvert)$	$\mathcal{O}(1)$
ALP11 [12]	✗	(N)MSP	✓	✓	✗	$\mathcal{O}(N_k)$	$\mathcal{O}(N_k\lvert\mathbb{A}\rvert)$	$\mathcal{O}(1)$
CCL+13 [21]	✗	Threshold	✗	✓	✓	$\mathcal{O}(\lvert\mathcal{U}\rvert)$	$\mathcal{O}(\lvert\mathcal{U}\rvert\lvert\mathbb{A}\rvert)$	$\mathcal{O}(1)$
Tak14 [49]	✗	NMSP	✓	✗	✗	$\mathcal{O}(N_k)$	$\mathcal{O}(N_k\lvert\mathbb{A}\rvert)$	$\mathcal{O}(1)$
AHY15 [11]	✓	(N)MSP	✓	✗	✗	$\mathcal{O}((N_kN_1)^2\lambda)$	$\mathcal{O}((N_kN_1)^4\lambda^2)$	$\mathcal{O}(1)$
AHM+16 [10]	✗	MSP	✓	✓	✓	$\mathcal{O}(n_k)$	$\mathcal{O}(n_k\lvert\mathbb{A}\rvert)$	$\mathcal{O}(\frac{\lvert\mathcal{S}\rvert}{n_k})$
AC16 [1,3]	✓	MSP	✓	✗	✗	$\mathcal{O}(N_1(N_2+N_k))$	$\mathcal{O}(\lvert\mathcal{S}\rvert N_1^2(N_2+N_k))$	$\mathcal{O}(1)$
Att19 [9]	✓	NMSP	✓	✗	✓	$\mathcal{O}(N_1N_2))$	$\mathcal{O}(\lvert\mathcal{S}\rvert N_1^2N_2)$	$\mathcal{O}(1)$
AT20 [13]	✓	(N)MSP	✓	✗	✓	$\mathcal{O}((N_2+N_k\lambda)^2)$	$\mathcal{O}((N_2+N_k\lambda)^4)$	$\mathcal{O}(1)$
LL20b [40]	✗	MSP	✓	✓	✗	$\mathcal{O}(N_k)$	$\mathcal{O}(N_k\lvert\mathbb{A}\rvert)$	$\mathcal{O}(1)$
TinyABE	✓	MSP	✓	✓	✓	$\mathcal{O}(\hat{n}_1(\hat{n}_2+n_k))$	$\mathcal{O}(\hat{n}_1^2(\hat{n}_2+\hat{n}_k)\frac{\lvert\mathcal{S}\rvert}{n_k})$	$\mathcal{O}(\min(\frac{n_1}{\hat{n}_1},\frac{n_2}{\hat{n}_2}))$

(a) ABE with short ciphertexts.

Scheme	**CP**	**LU**	**Unb.**	
			$\lvert\mathbb{A}\rvert$	$\lvert\mathcal{S}\rvert$
GPSW06 [30]	✗	✗	✓	✓
BSW07 [16]	✓	✓	✓	✓
Wat11-I [53]	✓	✗	✓	✓
Wat11-IV [53]	✓	✓	✓	✓
LW11 [39]	✗	✓	✓	✓
RW13 [45]	✓	✓	✓	✓
FAME [2]	✓	✓	✓	✓
ABGW17 [6]	✓	✓	✓	✓
TKN20 [50]	✓	✓	✓	✓
TinyABE	✓	✓	✓	✓

(b) ABE with linear-sized keys and ciphertexts that support MSPs.

Notes: $\mathcal{U}$ = universe; $\mathbb{A}$ = access policy; $\mathcal{S}$ = set of attributes;
(N)MSP = (non-)monotone span program, n_1, n_2 = number of rows, columns of $\mathbb{A}$;
N_1, N_2, N_k = maximum bounds on $n_1, n_2, |\mathcal{S}|$;
$\hat{n}_1, \hat{n}_2, n_k$ = maximum partition sizes of $n_1, n_2, |\mathcal{S}|$

3.3 Ciphertext-policy ABE

Definition 3 (Ciphertext-policy ABE [16]). *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*. It 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 message* M*, a policy* $\mathbb{A}$ *and the master public key* 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* $\text{CT}_{\mathbb{A}}$ *with access policy* $\mathbb{A}$*, and a secret key* $\text{SK}_{\mathcal{S}}$ *with attribute set* $\mathcal{S}$*. It succeeds and outputs a message* M' *if* $\mathcal{S}$ *satisfies* $\mathbb{A}$*. Otherwise, it fails.*

The scheme is correct if successful decryption of a ciphertext always yields the original message.

Large-universe and unbounded ABE. A scheme supports large universes if it does not impose bounds on the universe, which consists of all attributes that can be used in the scheme. We call the scheme unbounded, if it supports large universes and additionally does not impose bounds on the sets $\mathcal{S}$ and policies $\mathbb{A}$, or on the number of times $|\rho^{-1}(\text{att})|$ that one attribute att occurs in a policy.

3.4 Security model

Definition 4 (Full IND-CPA-security for CP-ABE [16]). *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 two equal-length messages* M_0 *and* M_1*, and an access structure* $\mathbb{A}^*$ *such that none of the sets* $\mathcal{S}_i$ *satisfies it, 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.*

3.5 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, so that $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$, with generators $g \in \mathbb{G}, h \in \mathbb{H}$ is 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).

3.6 Pair encoding schemes

Definition 5 (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 $\kappa = (p, \mathrm{par})$, where* par *specifies some parameters, is given by four deterministic polynomial-time algorithms as described below.*

- $\mathrm{Param}(\mathrm{par}) \to n$*: 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 6 (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*

- $\mathrm{EncB}(x) \to \mathbf{B}_1, ..., \mathbf{B}_n \in \mathbb{Z}_p^{d_1 \times d_2}$*;*

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

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

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

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.

4 Our construction: TinyABE

We present our construction. To this end, in Section 4.1, we give a step-by-step description on how these layering techniques can be applied, by first carefully reviewing the scheme. Roughly, we use the techniques of Attrapadung et al. [10,9] to remove the bounds on the attribute sets used in the key generation. Then, we apply the layering techniques to the ciphertext policy, by using the partitioning approach of Attrapadung et al. (AHM+16) [10]. However, unlike in AHM+16 [10], we need to partition in two "directions" due to the two-dimensional nature of access policies. In particular, for each policy, we split the set of rows in subsets of maximum size $\hat{n}_1$, and the set of columns in subsets of maximum size $\hat{n}_2$. Then, for each subset, we use a fresh "randomizer". These randomizers are appropriately applied to the ciphertext component of AC16 that embeds the policy. To this end, we identify which parts of this ciphertext component correspond to the rows and which to the columns:

$$C'' = \underbrace{\prod_{j \in [n_1], k \in [n_2]} g^{s A_{j,k} b_{j,k}}}_{\text{columns}} \underbrace{\prod_{i \in \overline{[n_k]}, j \in [n_1]} g^{s \rho(j)^i b'_{i,j}}}_{\text{rows}},$$

where $g^{b_{j,k}}$ and $g^{b'_{i,j}}$ denote public keys, and s is a random integer during encryption under access structure $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$. For example, for each partitioned subset $\mathcal{S}_l'$ of $[n_1]$, we use a fresh randomizer s_l to compute partial ciphertext $\prod_{i \in \overline{[n_k]}, j \in \mathcal{S}_l'} g^{s_l \rho(j)^i b'_{i,j}}$. In the scheme, we use mappings τ_1 and τ_2 to partition the rows and columns, respectively, in sets of maximum size $\hat{n}_1$ and $\hat{n}_2$. Furthermore, we define the mappings $\hat{\tau}_1$ and $\hat{\tau}_2$ to map each row and column, respectively, that are in the same partition to a unique set of public keys.

4.1 Removing the bounds from AC16

We show how to make AC16 [1] unbounded, by analyzing the scheme and showing, in steps, how the bounds can be removed by introducing more randomness.

The AC16 scheme. We briefly review the AC16 scheme [1]. Specifically, the secret keys SK and ciphertexts CT are of the form

$$\mathrm{SK} = (\{K_{1,j} = g^{r_j}, K_{2,j,k} = g^{r_j b_{j,k} - v_k}, K_{3,j,j',k} = g^{r_j b_{j',k}},$$
$$K_{4,j,\mathrm{att}} = g^{r_j \sum_{i \in \overline{[n_k]}} x_{\mathrm{att}}^i b'_{i,j}}, K_{5,i,j,j'} = g^{r_j b'_{i,j'}}\}_{\substack{i \in \overline{[n_k]}, j, j' \in [\hat{n}_1], j \neq j', \\ k \in [\hat{n}_2], \mathrm{att} \in \mathcal{S}}},)$$

$$\mathrm{CT} = \Bigg(C = M \cdot e(g,g)^{\alpha s}, C' = g^s,$$
$$C'' = \prod_{j \in [n_1], k \in [n_2]} g^{s A_{j,k} b_{j,k}} \prod_{i \in \overline{[n_k]}, j \in [n_1]} g^{s \rho(j)^i b'_{i,j}} \Bigg)$$

where $g^{b_{j,k}}$ and $g^{b'_{i,j}}$ denote public keys, $v_1 = \alpha$ is the master-key, $r_j, v_k \in_R \mathbb{Z}_p$ are randomly chosen integers during the key generation for set $\mathcal{S}$ with $|\mathcal{S}| \leq n_k$, and s is a randomly chosen integer during encryption for access structure $\mathbb{A} = (\mathbf{A}, \rho)$ with $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ such that $n_1 \leq N_1$ and $n_2 \leq N_2$ and $\rho \colon [n_1] \to \mathbb{Z}_p$, where $N_1, N_2 \in \mathbb{N}$ denote bounds on the policy size. Furthermore, x_{att} denotes the unique representation of an attribute att (represented as a string) in $\mathbb{Z}_p$, which can be generated with a collision-resistant hash function $\mathcal{H} \colon \{0,1\}^* \to \mathbb{Z}_p$.

Intuitively, decryption using a key SK for set $\mathcal{S}$ of a ciphertext CT with access policy $\mathbb{A}$ works by "singling out" each row $j \in \Upsilon = \{j' \in [n_1] \mid \rho(j') \in \mathcal{S}\}$, i.e., $e(g,g)^{r_j s (\sum_{k \in [\hat{n}_2]} A_{j,k} b_{j,k} + \sum_{i \in \overline{[n_k]}} \rho(j)^i b'_{i,j})}$ from C'' (and $K_{1,j}$). From this, $e(g,g)^{\sum_{k \in [\hat{n}_2]} A_{j,k} v_k}$ can be retrieved by using C', $K_{2,j,k}$ and $K_{4,j,\rho(j)}$. More concretely, this can be done because the secret keys are constructed in a specific way. That is, for each $j, j' \in [\hat{n}_1]$, it embeds the product $r_j b_{j',k}$, but only in the case that $j = j'$, it also embeds the secret v_k (where $v_1 = \alpha$). Similarly, for $j = j'$, only the secrets $r_j \sum_i x_{\mathrm{att}}^i b'_{i,j}$ are given for those attributes att that are in the set $\mathcal{S}$. For $j \neq j'$, we can reconstruct $r_j \sum_i x_{\overline{\mathrm{att}}}^i b'_{i,j'}$ for any attribute $\overline{\mathrm{att}}$. To decrypt, we have to retrieve $e(g,g)^{\alpha s}$, for which we would need to pair $K_{2,j,k}$ with C' to obtain $e(g,g)^{r_j b_{j,k} s - v_k s}$. Then, the question is how we can cancel out $e(g,g)^{r_j b_{j,k} s}$. Roughly, we want to "single out" the j-th row of the access policy in the ciphertext component C''. Then, we pair $K_{1,j} = g^{r_j}$ with C'', and cancel out all resulting components $e(g,g)^{r_j s (A_{j',k} b_{j',k} + \rho(j')^i b'_{i,j'})}$ for $j \neq j'$ by using $K_{3,j,j',k}$ and $K_{5,i,j,j'}$ and pairing them with C'. Note that we just argued that we can reconstruct these components (regardless of the fact that, possibly, $\rho(j') \notin \mathcal{S}$). This leaves us with components $e(g,g)^{r_j s (\sum_{k \in [\hat{n}_2]} A_{j,k} b_{j,k} + \sum_{i \in \overline{[n_k]}} \rho(j)^i b'_{i,j})}$. Then, we can only cancel $\prod_{i \in [n_k]} e(g,g)^{r_j s \rho(j)^i b'_{i,j}}$ if $\rho(j) \in \mathcal{S}$ (by pairing $K_{4,j,\mathrm{att}}$ with C'), which subsequently yields $\prod_{k \in [\hat{n}_2]} e(g,g)^{r_j s A_{j,k} b_{j,k}}$. By combining this with

$e(g,g)^{r_j b_{j,k} s - v_k s}$, we can obtain $e(g,g)^{\alpha s}$. For these last steps, we use the following LSSS property (Definition 2). If $\mathcal{S}$ satisfies $\mathbb{A}$, then there exist $\varepsilon_j \in \mathbb{Z}_p$ for all $j \in \Upsilon = \{j \in [n_1] \mid \rho(j) \in \mathcal{S}\}$ such that $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1,0,...,0)$ for rows $\mathbf{A}_j$ of matrix $\mathbf{A}$. Thus, computing $\prod_{k\in[\hat{n}_2]}(e(g,g)^{r_j b_{j,k} s - v_k s})^{\varepsilon_j A_{j,k}}$ yields $\prod_{j\in\Upsilon,k\in[\hat{n}_2]} e(g,g)^{r_j \varepsilon_j A_{j,k} b_{j,k} s} e(g,g)^{-\alpha s}$. We finally obtain $e(g,g)^{\alpha s}$ by raising $\prod_{k\in[\hat{n}_2]} e(g,g)^{r_j s A_{j,k} b_{j,k}}$ to the power ε_j for each $j \in \Upsilon$.

Removing the bound on set $\mathcal{S}$. First, we remove the bound n_k on set $\mathcal{S}$, which is simpler than removing the bounds on the access policy. In fact, this has already been done by Attrapadung [9], so we only briefly review his version of the scheme. Note that this method also resembles the method used in the AHM+16 scheme [10]. The general idea is that the set $\mathcal{S}$ is partitioned in arbitrary sets of maximum size n_k. For each partition, we use the randomness provided by r_j and the public keys to embed the partition like in the original scheme. However, because we have $m = \left\lceil \frac{|\mathcal{S}|}{n_k} \right\rceil$ partitions, we need m fresh sets of randomness $\{r_j\}_{j\in[\hat{n}_1]}$ for each partition. Hence, the keys look like this:

$$\mathrm{SK} = \{K_{1,j,l} = g^{r_{j,l}}, K_{2,j,k,l} = g^{r_{j,l} b_{j,k} - v_k}, K_{3,j,j',k,l} = g^{r_{j,l} b_{j',k}},$$
$$K_{4,j,\mathrm{att}} = g^{r_{j,\iota(\mathrm{att})} \sum_{i\in\overline{[n_k]}} x^i_{\mathrm{att}} b'_{i,j}}, K_{5,i,j,j',l} = g^{r_{j,l} b_{i,j'}}\}_{\substack{j,j'\in[\hat{n}_1], j\neq j', k\in[\hat{n}_2],, \\ i\in\overline{[n_k]}, l\in[m], \mathrm{att}\in\mathcal{S}}}$$

where $\iota\colon \mathcal{S} \to [m]$ maps the attributes of $\mathcal{S}$ into partitions. To ensure that the partitions are small enough, we place a restriction on ι, i.e., $|\iota^{-1}(l)| \leq n_k$ for all $l \in [m]$. Note here that the key component $K_{4,j,\mathrm{att}}$ only uses the randomizer $r_{j,l}$ corresponding to the partition in which x_{att} is mapped with ι. Furthermore, decryption is similar as in the AC16 scheme, though a little more care should be taken into picking the correct randomizer $r_{j,l}$.

Removing the bound from policy $\mathbb{A}$. It is considerably more difficult to remove the bounds on the access policy $\mathbb{A}$. Again, we need to introduce fresh randomness for each partition. However, we need to partition in two directions: the rows and the columns. Hence, we partition the access policy $\mathbb{A} = (\mathbf{A}, \rho)$ by splitting the rows in $m'_1 = \left\lceil \frac{n_1}{\hat{n}_1} \right\rceil$ partitions of maximum size $\hat{n}_1$, and the columns in $m'_2 = \left\lceil \frac{n_2}{\hat{n}_2} \right\rceil$ partitions of maximum size $\hat{n}_2$. In addition, we define the associated mappings $\tau_\beta\colon [n_\beta] \to [m'_\beta]$ for $\beta \in \{1,2\}$ that output the indices of the partitions in which the rows (for $\beta = 1$) and columns (for $\beta = 2$) are mapped. For each partition $l_\beta \in [m'_\beta]$, we introduce a randomizer s_{β,l_β}. In addition, we need to ensure that each row $j \in [n_1]$ in one partition uses a unique set of public parameters $\{g^{b_{j',k}}, g^{b'_{i,j'}}\}_{i\in\overline{[n_k]},k\in[\hat{n}_2]}$ (with $j' \in [\hat{n}_1]$). Similarly, we need to ensure that each column $k \in [n_2]$ in one partition uses a unique set $\{b_{j,k'}\}_{j\in[n_1]}$ (with $k' \in [\hat{n}_2]$). We thus define the corresponding mappings $\hat{\tau}_\beta\colon [n_\beta] \to [\hat{n}_\beta]$ such that $\hat{\tau}_\beta$ is injective on the subdomain $\tau_\beta^{-1}(l_\beta)$ for each $l_\beta \in [m'_\beta]$.

Then, we consider how we can apply any randomness in the ciphertext without causing incorrectness or insecurity. To this end, we analyze the AC16 ciphertext component C'', which is

$$\underbrace{\prod_{j\in[n_1],k\in[n_2]} g^{sA_{j,k}b_{j,k}}}_{C''_{\mathbf{A}}} \cdot \underbrace{\prod_{i\in\overline{[n_k]},j\in[n_1]} g^{s\rho(j)^i b'_{i,j}}}_{C''_{\rho}}.$$

For both parts $C''_{\mathbf{A}}$ and C''_{ρ}, we analyze with which randomness the randomness s needs to be replaced. As shown, the part associated with the access policy, i.e., $C''_{\mathbf{A}}$, is necessary to retrieve the secret $e(g,g)^{sA_{j,k}v_k}$, such that eventually $e(g,g)^{\alpha s}$ can be retrieved by computing $\prod_{j\in\Upsilon,k\in[n_2]} e(g,g)^{\varepsilon_j sA_{j,k}v_k}$. Note that, here, it is important that s is associated with $k=1$ to ensure correctness of the scheme. However, for $k>1$, we can use a different randomness. In short, for the $C''_{\mathbf{A}}$ part, we use the randomness associated with the m'_2 column partitions, which yields the transformation:

$$C''_{\mathbf{A}} \mapsto \prod_{j\in[n_1],k\in[n_2]} g^{s_{2,\tau_2(k)}A_{j,k}b_{\hat{\tau}_1(j),\hat{\tau}_2(k)}},$$

where we require that $s_{2,\tau_2(1)} = s$ to ensure correctness.

As shown, the part of C'' associated with the attribute mapping ρ, i.e., C''_{ρ}, ensures that the message is, albeit indirectly, sufficiently blinded. That is, in the "singling out" of row j, we could only obtain $\prod_k e(g,g)^{r_j sA_{j,k}b_{j,k}}$ (now: $\prod_k e(g,g)^{r_{j,\iota(\rho(j))}s_{2,\tau_2(k)}A_{j,k}b_{\hat{\tau}_1(j),\hat{\tau}_2(k)}}$) if we could cancel out $e(g,g)^{r_j s\rho(j)^i b'_{i,j}}$. This only worked if $\rho(j)\in\mathcal{S}$. In this case, using a fresh set $\{b'_{i,j}\}_{i\in\overline{[n_k]}}$ for each row j ensures that there is sufficient randomness for the entire partition. As such, it is straightforward that the C''_{ρ} part needs to be randomized for each partition of $\hat{n}_1$ rows, like in the removal of the bound on $\mathcal{S}$. For the row partitions, we had introduced the random integers s_{1,l_1} for each partition $l_1\in[m'_1]$, and substituting s for these yields:

$$C''_{\rho} \mapsto \prod_{i\in\overline{[n_k]},j\in[n_1]} g^{s_{1,\tau_1(j)}\rho(j)^i b'_{i,\hat{\tau}_1(j)}}.$$

Finally, we point out that the randomizers s_{1,l_1} and s_{2,l_2} are only used in combination with the public keys $b'_{i,j}$ and $b_{j,k}$, respectively. In our proofs, it becomes clear that we can therefore set $s_{2,l_2} = s_{1,l_2}$ for all $l_2\in[m'_2]$.

4.2 The scheme

We give our scheme in the selective-security setting. A fully secure variant can be obtained by applying the AC17 [3] transformation to our PES (Section 4.3).

Definition 7 (TinyABE). *TinyABE is defined as follows.*

– Setup(λ)*: On input the security parameter λ, the algorithm 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 $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. It sets the universe of attributes $\mathcal{U} = \mathbb{Z}_p$, and chooses $\hat{n}_1 \in \mathbb{N}$ and $\hat{n}_2 \in \mathbb{N}$ as the maximum number of rows and columns that fit into one partition, respectively. It also chooses $n_k \in \mathbb{N}$, which is the maximum partition size of the keys. It then generates random $\alpha, b_{j,k}, b'_{i,j} \in_R \mathbb{Z}_p$ for all $i \in \overline{[n_k]}, j \in [\hat{n}_1], k \in [\hat{n}_2]$. It outputs* $\text{MSK} = (\alpha, \{b_{j,k}, b'_{i,j}\}_{i \in \overline{[n_k]}, j \in [\hat{n}_1], k \in [\hat{n}_2]})$ *as the master secret key and publishes the domain parameters* $(p, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \hat{n}_1, \hat{n}_2, n_k)$ *and the master public key as*

$$\text{MPK} = (g, h, A = e(g,h)^\alpha, \{B_{j,k} = g^{b_{j,k}}, B'_{i,j} = g^{b'_{i,j}}\}_{i \in \overline{[n_k]}, j \in [\hat{n}_1], k \in [\hat{n}_2]}).$$

– 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 a partition mapping $\iota\colon \mathcal{S} \to [m]$ such that $|\iota^{-1}(l)| \leq n_k$ for each $l \in [m]$, and generates random integers $r_{j,l}, v_k \in_R \mathbb{Z}_p$ for each $j \in [\hat{n}_1], k \in [2, \hat{n}_2], l \in [m]$, setting $v_1 = \alpha$ and computes the secret key as*

$$\text{SK}_{\mathcal{S}} = (\{K_{1,j,l} = h^{r_{j,l}}, K_{2,j,k,l} = h^{r_{j,l} b_{j,k} - v_k}, K_{3,j,j',k,l} = h^{r_{j,l} b_{j',k}},$$
$$K_{4,j,\text{att}} = h^{r_{j,\iota(\text{att})} \sum_{i \in \overline{[n_k]}} x_{\text{att}}^i b'_{i,j}}, K_{5,i,j,j',l} = h^{r_{j,l} b_{i,j'}}\}_{\substack{j,j' \in [\hat{n}_1], j \neq j', k \in [\hat{n}_2], \\ i \in \overline{[n_k]}, l \in [m], \text{att} \in \mathcal{S}}}).$$

– Encrypt(MPK, $\mathbb{A}, M$)*: 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 $m'_1 = \left\lceil \frac{n_1}{\hat{n}_1} \right\rceil$ and $m'_2 = \left\lceil \frac{n_2}{\hat{n}_2} \right\rceil$, and defining partition mappings for each $\beta \in [2]$: $\tau_\beta\colon [n_\beta] \to [m'_\beta]$ such that $|\tau_\beta^{-1}(l_\beta)| \leq \hat{n}_\beta$ for each $l_\beta \in [m'_\beta]$. For τ_2, we require that $\tau_2(1) = 1$. Define $\hat{\tau}_\beta\colon [n_\beta] \to [\hat{n}_\beta]$ such that $\hat{\tau}_\beta$ is injective on the subset $\tau_\beta^{-1}(l_\beta)$ for each $l_\beta \in [m'_\beta]$. Then, generate random integers $s, s_{l'} \in_R \mathbb{Z}_p$ for each $l' \in [2, \max(m'_1, m'_2)]$, and specifically set $s_1 = s$, and compute the ciphertext as*

$$\text{CT}_{\mathbb{A}} = \big(C = M \cdot A^s, \{C_{l_1} = g^{s_{l_1}}\}_{l_1 \in [m'_1]}$$
$$C' = \prod_{j \in [n_1], k \in [n_2]} B_{\hat{\tau}_1(j), \hat{\tau}_2(k)}^{s_{\tau_2(k)} A_{j,k}} \prod_{i \in \overline{[n_k]}, j \in [n_1]} (B'_{i, \hat{\tau}_1(j)})^{s_{\tau_1(j)} \rho(j)^i}\big).$$

– Decrypt($\text{SK}_{\mathcal{S}}, \text{CT}_{\mathbb{A}}$)*: Suppose that $\mathcal{S}$ satisfies $\mathbb{A}$, let $\Upsilon = \{j \in [n_1] \mid \rho(j) \in \mathcal{S}\}$. Then, $\{\varepsilon_j \in \mathbb{Z}_p\}_{j \in \Upsilon}$ exist with $\sum_{i \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$ (Definition 2). Then, the plaintext M is retrieved by computing $C \cdot C_2 \cdot C_3 \cdot C_4 \cdot C_5 / C_1$, where*

$$C_1 = \prod_{j \in \Upsilon} e(C', K_{1, \hat{\tau}_1(j), \iota(\rho(j))}^{\varepsilon_j}), C_2 = \prod_{j \in \Upsilon, k \in [n_2]} e(C_{\tau_2(k)}, K_{2, \hat{\tau}_1(j), \hat{\tau}_2(k), \iota(\rho(j))}^{\varepsilon_j A_{j,k}}),$$
$$C_3 = \prod_{j \in \Upsilon, j' \in [n_1] \setminus \{j\}, k \in [n_2]} e(C_{\tau_2(k)}, K_{3, \hat{\tau}_1(j), \hat{\tau}_1(j'), \hat{\tau}_2(k), \iota(\rho(j))}^{\varepsilon_j A_{j',k}}),$$
$$C_4 = \prod_{j \in \Upsilon} e(C_{\tau_1(j)}, K_{4, \hat{\tau}_1(j), \rho(j)}^{\varepsilon_j}),$$
$$C_5 = \prod_{i \in \overline{[n_k]}, j \in \Upsilon, j' \in [n_1] \setminus \{j\}} e(C_{\tau_1(j')}, K_{5, i, \hat{\tau}_1(j), \hat{\tau}_1(j'), \iota(\rho(j))}^{\varepsilon_j \rho(j')^i}).$$

The scheme is correct (see Appendix A).

4.3 The associated pair encoding scheme

To prove security, we define the pair encoding of TinyABE, for which we use the variables $\hat{n}_1, \hat{n}_2, n_k, \mathcal{S}, \iota, \rho, \tau_1, \tau_2, \hat{\tau}_1, \hat{\tau}_2, n_1, n_2, \lambda_i, m, m'_1, m'_2$ from Definition 7.

Definition 8 (PES for TinyABE).

- Param(par) $\to \hat{n}_1(\hat{n}_2 + n_k)$. *Let* $\mathbf{b} = (\{b_{j,k}, b'_{i,j}\}_{i\in\overline{[n_k]}, j\in[\hat{n}_1], k\in[\hat{n}_2]})$.
- EncK($\mathcal{S}$) $\to \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \mathbf{b})$, *where*

$$\mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \mathbf{b}) = (\{k_{2,j,k,l} = r_{j,l}b_{j,k} - v_k, k_{3,j,j',k,l} = r_{j,l}b_{j',k},$$
$$k_{4,j,\mathrm{att}} = r_{j,\iota(\mathrm{att})} \sum_{i\in\overline{[n_k]}} x^i_{\mathrm{att}} b'_{i,j}, k_{5,i,j,j',l} = r_{j,l}b_{i,j'}\}_{\substack{i\in\overline{[n_k]}, j,j'\in[\hat{n}_1], j\neq j', \\ k\in[\hat{n}_2], l\in[m], \mathrm{att}\in\mathcal{S}}},)$$

  and $\mathbf{r} = (\{r_{j,l}\}_{j\in[n_1], l\in[m]})$ *are non-lone variables and* $\hat{\mathbf{r}} = (\{v_k\}_{k\in[2,n_2]})$ *are lone variables.*
- EncC($(\mathbf{A}, \rho)$) $\to \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}) = (c')$, *where*

$$c' = \sum_{j\in[n_1], k\in[n_2]} s_{\tau_2(k)} A_{j,k} b_{\hat{\tau}_1(j),\hat{\tau}_2(k)} + \sum_{i\in\overline{[n_k]}, j\in[n_1]} s_{\tau_1(j)} \rho(j)^i b'_{i,\hat{\tau}_1(j)}$$

  and we have non-lone variables $\mathbf{s} = (\{s_{l'}\}_{l'\in[\max(m'_1, m'_2)]})$.

Theorem 1. *The PES for TinyABE in Definition 8 satisfies the symbolic property (Definition 6).*

5 Security proof

We prove security of TinyABE (Definition 8) by proving symbolic security of our PES. This yields a fully secure scheme with the generic transformation in the AC17 [3] framework. In particular, we prove the selective and co-selective security properties (Definition 6) in Sections 5.2 and 5.3, respectively. Before that, we show in Section 5.1 how we combine the AC17 [4] symbolic proofs of AC16 [1], and the selective security proofs of RW13 [45].

5.1 "Unbounding" the AC17 proof of AC16

We elevate the AC17 [4] proof of the AC16 [1] (see Appendix B for a summary) to the unbounded setting by applying the techniques of Rouselakis and Waters [45]. (Note that, although we explain our methodology for the selective property, it is similar for the co-selective property.) Roughly speaking, with their layering techniques, we embed all row attributes that are mapped to the same public-key components $b'_{i,j}$ and $b_{j,k}$ for $j \in [\hat{n}_1]$ in these public-key components. Then, we use the individual randomness techniques of Rouselakis and Waters shared by

all row attributes in the same row partition to ensure that only the attribute layer associated with the partition is singled out in the challenge phase. We do something similar for the column partitions.

More specifically, for the challenge ciphertext polynomials to evaluate to $\mathbf{0}$ (as needed for the symbolic property (Definition 6)), we require the following substitutions. The public key $b'_{i,j}$ "embeds" all rows $j' \in [n_1]$ that are mapped to row $j \in [\hat{n}_1]$ with $\hat{\tau}_1$. For each of these rows (and their corresponding attribute), the individual randomness associated with its partition—to which it is mapped with τ_1—is used. Similarly, the substituted public key $b_{j,k}$ "embeds" all columns that are mapped to column $k \in [\hat{n}_2]$, which are randomized with the individual randomness for each corresponding partition. In this way, during the "challenge phase", the layers associated with the ciphertext partitions $l_1 \in [m'_1]$ and $l_2 \in [m'_2]$ (for the rows and columns, respectively) can be singled out for each public-key variable $b_{j,k}$ and $b'_{i,j}$. This works, because each public-key component only uses each partition randomness at most once (due to the restriction that $\hat{\tau}_\beta$ is injective on each subdomain $\tau_\beta^{-1}(l_\beta)$ with $l_\beta \in [m'_\beta]$).

For the "key query phase", the AC17 proof is adapted as follows. We embed another individual randomness, for each row $j' \in [n_1]$ that is mapped to row $j \in [\hat{n}_1]$, in the public-key variables $b'_{i,j}$, and we embed another individual randomness associated with the i-th coefficient. In particular, we embed the same polynomial as in the AC17 proof, but instead we embed one for each row j' that is mapped to j with $\hat{\tau}_1$. That is, we define an n_k-degree polynomial that yields 0 if $\rho(j')$ is plugged in. We embed the polynomials for which the roots are the attributes in the partition—like in the AC17 proof—for each row attribute. This ensures that the substituted encoding associated with $K_{4,j,\mathrm{att}}$ evaluates to $\mathbf{0}$.

5.2 The selective property

Our PES satisfies the selective security property. Let $\chi_{1,j} = \{j' \in [n_1] \mid \hat{\tau}_1(j') = \hat{\tau}_1(j)\}$ and $\chi_{2,k} = \{k' \in [n_2] \mid \hat{\tau}_2(k') = \hat{\tau}_2(k)\}$ for all $j \in [\hat{n}_1], k \in [\hat{n}_2]$. Let Υ as before and set $\overline{\Upsilon} = [n_1] \setminus \Upsilon$. Because $\mathcal{S}$ does not satisfy $\mathbb{A}$, there exists $\mathbf{w} = (1, w_2, ..., w_{n_2}) \in \mathbb{Z}_p^{n_2}$ such that $\mathbf{A}_j \mathbf{w}^\intercal = 0$ for all $j \in \Upsilon$ (Definition 2). Let $\hat{G}_{j,k}(x_{\mathrm{att}}) = \sum_{i \in [n_k]} (x_{\mathrm{att}}^i - \rho(j)^i) \mathbf{1}^{d_1 \times d_2}_{(i,j,k),\tau_1(j)}$ for all $j \in [n_1], k \in [n_2]$. Let $\Psi_l = \{\mathrm{att} \in \mathcal{S} \mid \iota(\mathrm{att}) = l\}$ be the l-th partition of $\mathcal{S}$ for all $l \in [m]$. Then, for each $l \in [m]$, we define $G_l(x_{\mathrm{att}}) = \prod_{\mathrm{att}' \in \Psi_l} (x_{\mathrm{att}} - x_{\mathrm{att}'}) = \sum_{i=0}^{n_k} u_{i,l} x_{\mathrm{att}}^i$. We make the following substitutions:

$$b'_{0,j} \;:\; \sum_{j' \in \chi_{1,j}, k' \in [n_2]} A_{j',k'} \left(\mathbf{1}^{d_1 \times d_2}_{(j',k'),\tau_1(j')} - \sum_{i' \in [n_k]} \rho(j')^{i'} \mathbf{1}^{d_1 \times d_2}_{(i',j',k'),\tau_1(j')} \right)$$

$$b'_{i,j} \;:\; \sum_{j' \in \chi_{1,j}, k' \in [n_2]} A_{j',k'} \mathbf{1}^{d_1 \times d_2}_{(i,j',k'),\tau_1(j')}$$

$$b_{j,k} \;:\; - \sum_{k' \in \chi_{2,k}} \mathbf{1}^{d_1 \times d_2}_{(j,k),\tau_2(k')}, \quad s_{l'} \;:\; \overline{\mathbf{1}}^{d_2}_{l'}, \quad v_k \;:\; - w_k \left(\sum_{k' \in \chi_{2,k}} \overline{\mathbf{1}}^{d_2}_{\tau_2(k')} \right)$$

$$r_{j,l} \;:\; \sum_{k' \in [n_2]} w_{k'} \left(\mathbf{1}^{d_1}_{(j,k')} - \sum_{i' \in [n_k], j' \in \chi_{1,j} \cap \overline{\Upsilon}} \frac{u_{i',l}}{G_l(\rho(j'))} \mathbf{1}^{d_1}_{(i',j',k')} \right)$$

for all $i \in [n_k], j \in [\hat{n}_1], k \in [\hat{n}_2], l \in [m], l' \in [\max(m'_1, m'_2)]$, where the row indices (j,k) and (i,j,k) are mapped injectively in the interval $[(n_k+2)n_1n_2]$. We have $d_1 = (n_k+2)n_1n_2$ and $d_2 = \max(m'_1, m'_2)$. It follows quickly that the polynomials evaluate to $\mathbf{0}$ (see Appendix C.1).

5.3 The co-selective property

For the co-selective property, we generalize the co-selective proof by Agrawal and Chase [4]. In this proof, the coefficients of the polynomials $G_l(x_{\text{att}})$ are embedded in the variables $b'_{i,j}$. We make the following substitutions:

$$b'_{i,j} \;:\; \sum_{l \in [m]} u_{i,l} \mathbf{1}^{d_1 \times d_2}_{(j,l),(1,j,l)}, \quad b_{j,k} \;:\; \sum_{l \in [m]} \mathbf{1}^{d_1 \times d_2}_{(j,l),(2,k)}, \quad v_k \;:\; \overline{\mathbf{1}}^{d_2}_{(2,k)},$$

$$r_{j,l} \;:\; \mathbf{1}^{d_1}_{j,l}, \quad s_{l'} \;:\; \sum_{k \in \tau_2^{-1}(l')} w_k \overline{\mathbf{1}}^{d_2}_{2,\hat{\tau}_2(k)} - \sum_{j \in \tau_1^{-1}(l') \cap \overline{\Upsilon}, l \in [m]} \frac{\mathbf{A}_j \mathbf{w}^\intercal}{G_l(\rho(j))} \overline{\mathbf{1}}^{d_2}_{(1,\hat{\tau}_1(j),l)}$$

for all $i \in [n_k], j \in [\hat{n}_1], k \in [\hat{n}_2], l \in [m], l' \in [\max(m'_1, m'_2)]$, where the row indices (j,l) are mapped injectively in the interval $[d_1]$, and the column indices $(1,j,l)$ and $(2,k)$ in the interval $[d_2]$, where $d_1 = \hat{n}_1 m$ and $d_2 = \hat{n}_1 m + \hat{n}_2$. It follows quickly that the polynomials go to $\mathbf{0}$ (see Appendix C.2).

6 Performance analysis

We analyze the performance of TinyABE for two configurations relevant to IoT settings. To illustrate the efficiency trade-offs and the advantages of TinyABE more clearly, we compare the efficiency of the two configurations with two large-universe CP-ABE schemes: one with linear-size ciphertexts and one with constant-size ciphertexts. In particular, we compare RW13 [45], which is an unbounded CP-ABE scheme with linear-size ciphertexts, and the version of AC16 [1] with unbounded attribute sets and constant-size ciphertexts: Att19 [9]. To effectively compare the efficiency of all relevant schemes, we run benchmarks for various group operations in RELIC [7], and extrapolate the computational costs of the schemes by counting the number of operations required by the algorithms. In particular, we use the pairing-friendly elliptic-curve group BLS12-381 [14,18] for our analysis. For this curve, Scott has recently performed measurements on several IoT devices [48], which we will use in our analysis in Section 6.3. RELIC [7] supports efficient constant-time implementations for "regular" exponentiations and two special types of exponentiation: fixed-base and multi-base exponentiation. In a fixed-base exponentiation, the base g to be exponentiated in g^x is fixed after setup, and as such, a precomputation table can be made to

speed up the computation [20]. In a multi-base exponentiation, the product of multiple exponentiations, e.g., $g_1^{x_1} \dots g_n^{x_n}$, is computed [41]. We have run these benchmarks on a 1.6 GHz Intel i5-8250U processor[5] (see Appendix E).

We compare the schemes as follows. For a fair comparison, we place all of them in the selective-security and prime-order setting. We then convert the schemes to the asymmetric setting using the same optimization approaches [43]. For each scheme, we optimize the encryption efficiency[6], e.g., by placing each ciphertext component in $\mathbb{G}$. (In Appendix D, we give a full description of RW13. Att19 is a special case of TinyABE, where $\hat{n}_1$ and $\hat{n}_2$ are the upper bounds on the access policies used during encryption.) Afterwards, we convert the results to match the most efficient variant in the full-security setting by applying the generic transformation in the AC17 [3] framework, which incurs roughly twice the costs. To obtain the most efficient implementation of encryption and decryption, we assume that the access policies used during encryption are Boolean formulas. Any Boolean formula can be converted to an LSSS matrix with matrix entries $A_{j,k}$ in $\{-1, 0, 1\}$, and coefficients ε_j in $\{0, 1\}$ [38].

6.1 Computational costs of TinyABE

We list the computational costs of the key generation, encryption and decryption algorithms by listing the number of group operations required by these (see Appendix F for further details on our analysis):

- **Key generation:** $\hat{n}_1(m + \hat{n}_1\hat{n}_2 m + |\mathcal{S}| + n_k(\hat{n}_1 - 1)m)$ fixed-base exponentiations in $\mathbb{H}$;
- **Encryption:** one exponentiation in $\mathbb{G}_T$, m fixed-base exponentiations in $\mathbb{G}$, one $(n_k \min(\hat{n}_1, |\Upsilon|))$-multi-base exponentiation in $\mathbb{G}$, the minimum of the following two costs:
  - one $(\hat{n}_1\hat{n}_2)$-multi-base exponentiation in $\mathbb{G}$;
  - one m_2'-multi-base exponentiation in $\mathbb{G}$ and $n_1 n_2$ multiplications in $\mathbb{G}$;

  and the minimum of the following two costs:
  - one $\hat{n}_1$-multi-base-exponentiation in $\mathbb{G}$;
  - one m_1'-multi-base exponentiation and n_1 multiplications in $\mathbb{G}$.

  (Note that the ciphertext component C' can be computed in multiple ways, which we show in more detail in Appendix F.2. Specifically, the costs are upper bounded by a constant.)
- **Decryption:** one $(m_1' + 1)$-multi-pairing operation, and $|\Upsilon|\hat{n}_1 n_2 + n_k n_1$ exponentiations and $(|\Upsilon| - 2)n_k n_1 + 2|\Upsilon|$ multiplications in $\mathbb{H}$.

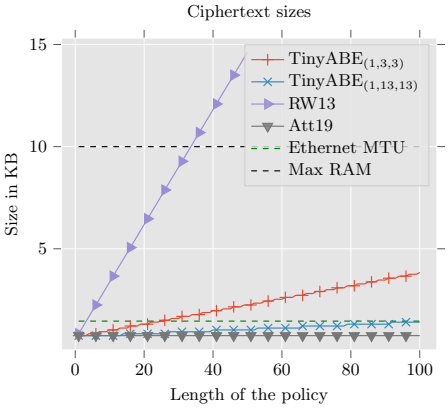
To convert these costs to the full-security setting (using the most efficient transformation in [3]), we multiply the costs in $\mathbb{G}$ and $\mathbb{H}$ by a factor 2.

[5] Our code is available as a Jupyter notebook at github.com/mtcvenema/tinyabe.

[6] In Appendix F, we explain how a more balanced encryption-decryption efficiency can be attained.

Scheme	Key sizes (in KB)		
	Size of the set		
	1	10	100
$\text{TinyABE}_{(1,3,3)}$	7.39	73.89	738.87
$\text{TinyABE}_{(1,13,13)}$	450.71	4507.09	$45.1{\cdot}10^3$
RW13	0.76	4.17	38.27
Att19	$191.3{\cdot}10^3$	$1.9\cdot10^6$	$19.1\cdot10^6$

(a) Secret keys



(b) Ciphertexts

Fig. 1: The key and ciphertext sizes of $\text{TinyABE}_{(n_k,\hat{n}_1,\hat{n}_2)}$, RW13 and Att19.

Two configurations of TinyABE. To investigate the feasibility of TinyABE in IoT settings, we analyze the efficiency of TinyABE for two configurations, i.e.,

1. where encryption is optimal: $(n_k, \hat{n}_1, \hat{n}_2) = (1, 3, 3)$;
2. where ciphertexts are small: $(n_k, \hat{n}_1, \hat{n}_2) = (1, 13, 13)$.

In particular, for the latter configuration, the ciphertexts are small enough such that they fit in one Ethernet packet for policy sizes $|\mathbb{A}| \leq 100$. In Appendix F.3, we give further details on how the parameters can be chosen.

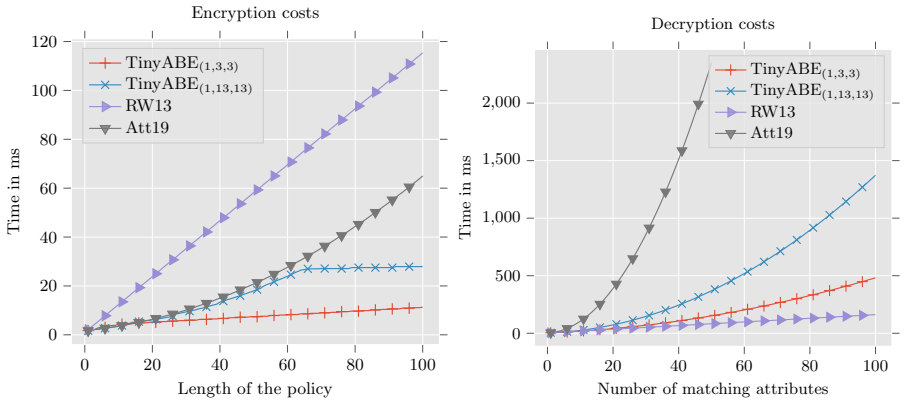
6.2 Comparison with RW13 and AC16/Att19

We compare the efficiency of TinyABE with RW13, a scheme with linear-size ciphertexts and Att19, the variant of AC16 with constant-size ciphertexts that is unbounded in sets $\mathcal{S}$. In particular, we focus on the ciphertext size and encryption efficiency, as ultimately, we want to optimize the scheme for low-quality networks and resource-constrained encryption devices. For all schemes, we require that they can support any $|\mathbb{A}|, |\mathcal{S}| \leq 100$, thus we set the upper bounds N_1, N_2 of Att19 on the number of rows and columns to $N_1 = N_2 = 100$. Because the key sizes and the key generation costs are linear and the differences between the schemes are large, we put those results in tables, and the rest in graphs.

Storage costs. As Figure 1 shows, our secret keys are generally much larger than RW13. Nevertheless, our ciphertexts are much smaller. Our scheme's configurations never exceed the maximum RAM size, and $\text{TinyABE}_{(1,13,13)}$ never exceeds the maximum transmission unit (MTU), which we show to be beneficial in Section 6.3. Compared to Att19, our keys are much smaller, while our ciphertexts are marginally larger. Importantly, our master public key is, at most, 19.13

Scheme	Key generation costs (in ms)		
	Size of the set		
	1	10	100
TinyABE$_{(1,3,3)}$	20.3	202.8	$2 \cdot 10^3$
TinyABE$_{(1,13,13)}$	$1.2 \cdot 10^3$	$12.4 \cdot 10^3$	$123.7 \cdot 10^3$ ($\approx$ 2 minutes)
RW13	2.1	11.4	105.1
Att19	$525.3 \cdot 10^3$ ($>$ 8 minutes)	$5.3 \cdot 10^6$ ($>$ 1 hour)	$52.5 \cdot 10^6$ ($>$ 14 hours)

(a) Key generation



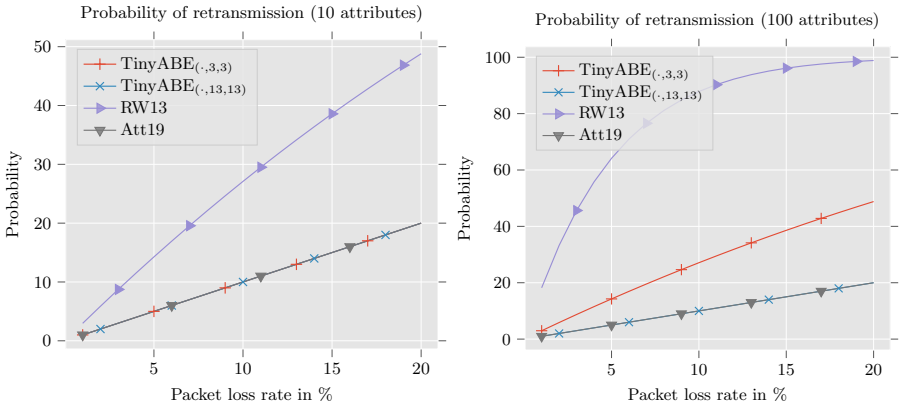
(b) Encryption (c) Decryption

Fig. 2: The computational costs (in milliseconds (ms)) of key generation, encryption and decryption with TinyABE$_{(n_k,\hat{n}_1,\hat{n}_2)}$, RW13 and Att19.

KB, in contrast to the 957 KB of Att19. In Section 6.3, we show that our scheme can thus be used in resource-constrained devices while Att19 cannot.

Computational costs. Similarly, Figure 2 shows that our key generation and decryption costs are higher than RW13, but our encryption is much more efficient. We show in Section 6.3 that this gain in encryption efficiency can make a difference between deployment or not, as it reduces the encryption timings on resource-constrained devices from minutes to mere seconds. (Also note that our key generation can be extended to an online/offline variant (see Appendix G). This allows the authority that generates keys to prepare many keys in advance (e.g., 0.7-44 MB per 100 attributes for $(n_k, \hat{n}_1, \hat{n}_2) \in \{(1,3,3),(1,13,13)\}$), which mitigates any potential issues caused by the large costs.) Moreover, all of our configurations outperform Att19, notably reducing the key generation costs to more feasible timings. While Att19 takes at least eight minutes to compute a key, our scheme never requires more than two minutes, even for large sets.

Comparison with other linear-sized schemes. The main reason why we compare our scheme with RW13 is because it is closely related to our scheme,



(a) 10 attributes (b) 100 attributes

Fig. 3: The probability that a partial ciphertext needs to be retransmitted for various packet loss rates.

and because it has linear-size ciphertexts and it is unbounded. In addition, to illustrate the advantages of TinyABE in IoT settings, we want to compare mainly its encryption (and the public key and ciphertext sizes) with a linear-sized scheme such as RW13. Because encryption with other popular large-universe schemes [16,2,6] is roughly as efficient as RW13 (see Appendix F.4), we expect that TinyABE compares similarly as favorably to those schemes as to RW13.

6.3 Advantages in low-quality networks and constrained devices

We showcase some practical advantages of TinyABE in IoT settings.

Packet loss in low-quality networks. One of the main features of TinyABE is that the ciphertexts can be configured to be small, which is beneficial in low-quality networks. In general, large ciphertexts may increase the risk that at least one of the packets is dropped during transmission, in which case the dropped packets need to be retransmitted [42], delaying the message's time of arrival. This may be problematic for resource-constrained devices, such as IoT devices, because their communication channels are often characterized by high packet loss rates (see RFC8576[7]). Packet sizes on the physical layer are even more limited, increasing the probability that packets are lost. Furthermore, these devices typically have much less memory at their disposal, meaning that they may not be able to store the entire ciphertext. As a result, these devices need to stream out partial ciphertexts, and either retain the partial ciphertext until the intended receiver confirms that it has arrived, or risk having to recompute it. To mitigate these issues, the size of the ciphertexts that are transmitted from the

[7] https://tools.ietf.org/html/rfc8576

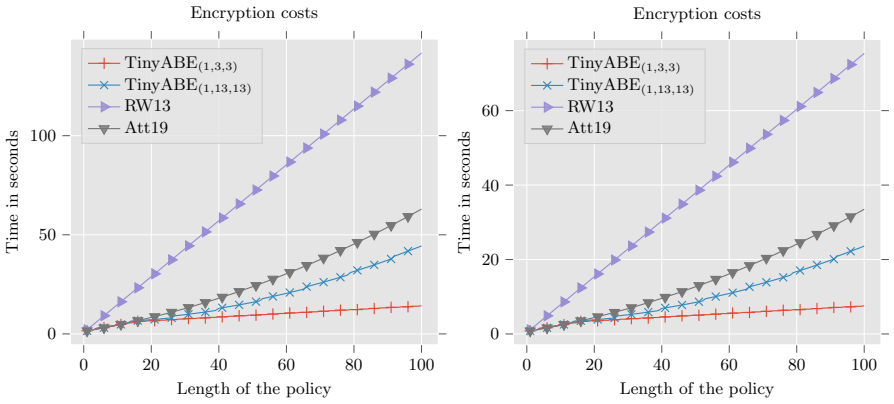
device should be minimized. TinyABE achieves this. In fact, even for $\hat{n}_1 = 3$, which optimizes the encryption efficiency, the probability that ciphertexts need to be retransmitted is significantly lower than for RW13.

Through an example, we briefly illustrate that the increase in size of the ciphertext may increase the probability that one of the packets drops. For this example, we use a similar approach as in [42]. We consider the maximum transmission unit (MTU) of an Ethernet connection, i.e., 1500 bytes, and consider varying packet loss rates between 1% and 20%, with steps of 1%. In general, if we consider the shortest round-trip time, then the expected additional time incurred by re-transmitting a packet is 6.193 ms. In comparison, we can encrypt a message under a policy of length 20 with our least efficient configuration of our scheme in that time (Figure 2), so this additional overhead is relevant in practice. To illustrate the effect of packet loss on the efficiency of the scheme, we analyze the probability that at least a part of the ciphertext is dropped in Figure 3. As the figures show, TinyABE never exceeds 50%, even for large policies and high packet loss rates. In contrast, for small policies, RW13 exceeds 50% at high rates (i.e., $> 20\%$) and for large policies at small rates (i.e., $> 3.5\%$). For large policies and high rates, it is almost certain that a partial RW13 ciphertext drops. Therefore, our scheme clearly provides an advantage in low-quality networks compared to schemes with linear-size ciphertexts such as RW13.

Resource-constrained devices: memory. In contrast to other expressive schemes such as RW13 and Att19, the ciphertexts and master public key of TinyABE easily fit in memory, even of resource-constrained devices. In RFC7228[8], three classes of constrained devices are listed: with < 10, ≈ 10 and 50 KB of RAM, and with < 100, ≈ 100 and 250 KB of flash memory (ROM), respectively. In practice, the master public key can be stored in flash memory, such that only the components that are needed during encryption are loaded in RAM. While the RW13 public key (of 0.89 KB) fits easily in flash memory, our public keys are fairly large, e.g., 2.25-19.13 KB for $(n_k, \hat{n}_1, \hat{n}_2) \in \{(1,3,3),(1,13,13)\}$. Although it leaves slightly less space for the code and other applications than RW13 would, it easily fits in devices with at least 100 KB of flash memory. This is not the case for the Att19 scheme, as the master public key of 957 KB is much larger than 100 KB for maximum policy length 30 or higher.

Additionally, while the ciphertexts of our scheme as well as the RW13 scheme would easily fit in 50 KB of RAM (even for large policies), it would be more problematic for devices with only 10 KB of RAM to fit RW13 ciphertexts. In fact, a ciphertext with policy length 33 already pushes the limit of 10 KB, and leaves no space for anything else, such as payload, additional overhead incurred by the computation, or optimization through precomputation. As such, trade-offs need to be made, e.g., by limiting the size of the access policy or by streaming out partially computed ciphertexts. However, because IoT devices often have a lossy communication channel, this increases the risk of recomputation of ciphertext components, increasing the encryption costs. Alternatively, the device can retain

[8] https://tools.ietf.org/html/rfc7228



(a) Arduino Nano 33 BLE Cortex-M4 (b) Fishino Piranha MIPS32 (@120MHz)

Fig. 4: Conservative estimates of the encryption costs on two IoT devices.

the partial ciphertexts, until the receiver has confirmed that the ciphertexts have arrived, which however also increases the encryption time. In contrast, TinyABE's ciphertexts easily fit in 10 KB of memory, even for large policies. For $\hat{n}_1 = 3$ and policy length $n_1 = 100$, the size of the ciphertext is 3.84 KB. For $\hat{n}_1 = 13$, the size of the ciphertext is 1.41 KB, which leaves an ample 8.59 KB of memory for e.g., the payload and optimization through precomputation. In this way, we may be able to gain some significant speedup [29]. Furthermore, because the ciphertext fits entirely in memory, the device can retain it, until it has been notified by the intended receiver that the ciphertext has arrived successfully. It therefore does not require the device to recompute any partial ciphertexts either, which may be the case for RW13.

Resource-constrained devices: speed. As shown, for some parameter choices, our scheme provides fast encryption, even for large inputs. For instance, for $(n_k, \hat{n}_1) = (1, 3)$, our encryption algorithm is several factors faster than RW13, which may be desirable for resource-constrained devices. Recently, Scott [48] tested the performance of some operations on the BLS12-381 curve on IoT devices. These results show that the devices with the slowest and fastest timings would approximately require 1.175 and 0.075 seconds, respectively, per exponentiation in $\mathbb{G}$. In Figure 4, we estimate the encryption costs for the two fastest devices measured in [48]. For the fastest device, our most efficient configuration requires only 7.6 seconds to encrypt with a policy of length 100, while RW13 requires one minute and 15 seconds.

7 Future work

Our work paves the way for further improvements in the design and implementation of ABE in IoT settings. While we have theoretically analyzed the feasibility

of implementing our schemes in practical settings such as low-quality networks and embedded devices, it would be useful to empirically test them. Notably, our conservative estimates for the IoT devices in Section 6.3 are based on the benchmarks in [48]. By implementing the scheme—possibly using optimizations (e.g., through precomputation [29]) that have not been used in [48] or using curves with more efficient arithmetic in $\mathbb{G}$ [24]—it may perform even better than our estimates suggest. Furthermore, our scheme supports monotone span programs only. It can be transformed into a scheme that also supports non-monotone span programs by applying transformations by Attrapadung [9] and Ambrona [5]. Finally, because our scheme is proven secure in the AC17 [3] framework, its security relies on a q-type assumption [17,19]. Even though existing attacks [23] have not been shown to break the specific q-type assumptions used in AC17, from a theoretical point of view, it might be preferable to rely on non-parametrized assumptions [40,13].

8 Conclusion

We proposed a new configurable unbounded large-universe CP-ABE scheme, mainly designed for settings with embedded devices or low-quality networks. We have proven the scheme secure in the AC17 framework, yielding efficient constructions that are provably fully secure. TinyABE can be configured such that encryption is very efficient, outperforming state-of-the-art CP-ABE schemes by several factors for large policies. Additionally, the ciphertexts are much shorter than those of schemes with linear-size ciphertexts, and are therefore more likely to fit in the constrained memories of embedded devices. Due to this shortness, ciphertexts are also much less likely to drop during transmission. While the ciphertexts are longer than those of schemes with constant-size ciphertexts, our public and secret keys are much shorter, and the computational costs are much lower. For these reasons, TinyABE is more practical for embedded devices and low-quality networks.

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. Ambrona, M.: Generic negation of pair encodings. In: PKC. pp. 120–146. Springer (2021)
6. 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)

7. 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
8. 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)
9. Attrapadung, N.: Unbounded dynamic predicate compositions in attribute-based encryption. In: EUROCRYPT. pp. 34–67. Springer (2019)
10. Attrapadung, N., Hanaoka, G., Matsumoto, T., Teruya, T., Yamada, S.: Attribute based encryption with direct efficiency tradeoff. In: ACNS. pp. 249–266. Springer (2016)
11. Attrapadung, N., Hanaoka, G., Yamada, S.: Conversions among several classes of predicate encryption and applications to ABE with various compactness tradeoffs. In: ASIACRYPT. pp. 575–601. Springer (2015)
12. Attrapadung, N., Libert, B., de Panafieu, E.: Expressive key-policy attribute-based encryption with constant-size ciphertexts. In: PKC. pp. 90–108. Springer (2011)
13. Attrapadung, N., Tomida, J.: Unbounded dynamic predicate compositions in ABE from standard assumptions. In: ASIACRYPT. pp. 405–436. Springer (2020)
14. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: SCN. pp. 257–267. Springer (2002)
15. Beimel, A.: Secure schemes for secret sharing and key distribution (1996)
16. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: S&P. pp. 321–334. IEEE (2007)
17. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT. pp. 440–456. Springer (2005)
18. Bowe, S.: Bls12-381: New zk-snark elliptic curve construction. https://blog.z.cash/new-snark-curve/
19. Boyen, X.: The uber-assumption family – a unified complexity framework for bilinear groups. In: Pairing. pp. 39–56. Springer (2008)
20. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation. In: EUROCRYPT. pp. 200–207. Springer (1992)
21. Chen, C., Chen, J., Lim, H.W., Zhang, Z., Feng, D., Ling, S., Wang, H.: Fully secure attribute-based systems with short ciphertexts/signatures and threshold access structures. In: CT-RSA. pp. 50–67. Springer (2013)
22. Chen, C., Zhang, Z., Feng, D.: Efficient ciphertext policy attribute-based encryption with constant-size ciphertext and constant computation-cost. In: ProvSec. pp. 84–101. Springer (2011)
23. Cheon, J.H.: Security analysis of the strong diffie-hellman problem. In: Vaudenay, S. (ed.) EUROCRYPT. LNCS, vol. 4004, pp. 1–11. Springer (2006)
24. Clarisse, R., Duquesne, S., Sanders, O.: Curves with fast computations in the first pairing group. In: CANS. pp. 280–298. Springer (2020)
25. Emura, K., Miyaji, A., Nomura, A., Omote, K., Soshi, M.: A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In: ISPEC. pp. 13–23. Springer (2009)
26. ETSI: ETSI TS 103 458 (V1.1.1) (2018)
27. ETSI: ETSI TS 103 532 (V1.1.1) (2018)
28. ETSI: Even more advanced cryptography – industry applications and use cases for advanced cryptography. ETSI Security Week 2020, slides available at https://docbox.etsi.org/Workshop/2020 (2020)
29. Fujii, H., Aranha, D.F.: Curve25519 for the cortex-m4 and beyond. In: LATINCRYPT. pp. 109–127. Springer (2017)

30. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: CCS. ACM (2006)
31. 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)
32. Herranz, J., Laguillaumie, F., Ràfols, C.: Constant size ciphertexts in threshold attribute-based encryption. In: PKC. pp. 19–34. Springer (2010)
33. Hohenberger, S., Waters, B.: Online/offline attribute-based encryption. In: PKC. pp. 293–310. Springer (2014)
34. Hülsing, A., Rijneveld, J., Schwabe, P.: Armed SPHINCS - computing a 41 KB signature in 16 KB of RAM. In: PKC. pp. 446–470. Springer (2016)
35. Kamara, S., Lauter, K.E.: Cryptographic cloud storage. In: FC. pp. 136–149. Springer (2010)
36. Kumar, S., Hu, Y., Andersen, M.P., Popa, R.A., Culler, D.E.: JEDI: many-to-many end-to-end encryption and key delegation for iot. In: 28th USENIX Security Symposium. pp. 1519–1536. USENIX Association (2019)
37. Lewko, A., Sahai, A., Waters, B.: Revocation systems with very small private keys. In: IEEE S & P. pp. 273–285 (2010)
38. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. Cryptology ePrint Archive, Report 2010/351 (2010)
39. Lewko, A.B., Waters, B.: Unbounded HIBE and attribute-based encryption. In: EUROCRYPT. pp. 547–567. Springer (2011)
40. Lin, H., Luo, J.: Succinct and adaptively secure ABE for ABP from k-lin. In: ASIACRYPT. pp. 437–466. Springer (2020)
41. Möller, B.: Algorithms for multi-exponentiation. In: SAC. pp. 165–180. Springer (2001)
42. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in TLS. In: PQCrypto. pp. 72–91. Springer (2020)
43. de la Piedra, A., Venema, M., Alpár, G.: ABE squared: Accurately benchmarking efficiency of attribute-based encryption. TCHES **2022**(2), 192—-239 (2022)
44. Pirretti, M., Traynor, P., McDaniel, P.D., Waters, B.: Secure attribute-based systems. J. Comput. Secur. **18**(5), 799–837 (2010)
45. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption. In: CCS. pp. 463–474. ACM (2013)
46. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT. pp. 457–473. Springer (2005)
47. 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)
48. Scott, M.: On the deployment of curve based cryptography for the internet of things. Cryptology ePrint Archive, Report 2020/514 (2020)
49. Takashima, K.: Expressive attribute-based encryption with constant-size ciphertexts from the decisional linear assumption. In: SCN. pp. 298–317. Springer (2014)
50. Tomida, J., Kawahara, Y., Nishimaki, R.: Fast, compact, and expressive attribute-based encryption. In: PKC. pp. 3–33. Springer (2020)
51. Venema, M., Alpár, G.: A bunch of broken schemes: A simple yet powerful linear approach to analyzing security of attribute-based encryption. In: CT-RSA. pp. 100–125. Springer (2021)
52. Venema, M., Alpár, G., Hoepman, J.H.: Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. Cryptology ePrint Archive, Report 2021/1172 (2021)

53. Waters, B.: Ciphertext-policy attribute-based encryption - an expressive, efficient, and provably secure realization. In: PKC. pp. 53–70. Springer (2011)

A Correctness of TinyABE

TinyABE (i.e., the scheme in Definition 7) is correct, i.e.,

$$C_2 \cdot C_3 = e(g,h)^{\sum_{j\in\Upsilon, j'\in[n_1], k\in[n_2]} \varepsilon_j s_{\tau_2(k)} (r_{\hat{\tau}_1(j),\mathrm{att}} A_{j',k} b_{j',k} - A_{j,k} v_{\hat{\tau}_2(k)})}$$
$$= e(g,h)^{\sum_{j\in\Upsilon, j'\in[n_1], k\in[n_2]} \varepsilon_j r_{\hat{\tau}_1(j),\iota(\rho(j))} s A_{j',k} b_{j',k}} \cdot e(g,h)^{-\alpha s}$$
$$C_6 = C \cdot C_2 \cdot C_3$$
$$= M \cdot e(g,h)^{\sum_{j\in\Upsilon, j'\in[n_1], k\in[n_2]} \varepsilon_j r_{\hat{\tau}_1(j),\iota(\rho(j))} s_{\tau_2(k)} A_{j',k} b_{j',k}}$$
$$C_4 \cdot C_5 = e(g,h)^{\sum_{i\in\overline{[n_k]}, j\in\Upsilon, j'\in[n_1]} \varepsilon_j r_{\hat{\tau}_1(j),\iota(\rho(j))} s_{\tau_1(j')} \rho(j')^i b'_{i,j'}}$$
$$C_1^{-1} = e(g,h)^{-\sum_{i\in\overline{[n_k]}, j\in\Upsilon, j'\in[n_1], k\in[n_2]} \varepsilon_j r_{\hat{\tau}_1(j),\iota(\rho(j))} s_{\tau_2(k)} A_{j',k} b_{j',k}}$$
$$\cdot e(g,h)^{-\sum_{i\in\overline{[n_k]}, j\in\Upsilon, j'\in[n_1], k\in[n_2]} \varepsilon_j r_{\hat{\tau}_1(j),\iota(\rho(j))} s_{\tau_1(j')} \rho(j')^i b'_{i,j'}}$$
$$C_7 = C_4 \cdot C_5 \cdot C_1^{-1}$$
$$= e(g,h)^{-\sum_{j\in\Upsilon, j'\in[n_1], k\in[n_2]} \varepsilon_j r_{\hat{\tau}_1(j),\iota(\rho(j))} s_{\tau_2(k)} A_{j',k} b_{j',k}}$$
$$C_6 \cdot C_7 = C \cdot C_2 \cdot C_3 \cdot C_4 \cdot C_5 / C_1 = M.$$

B The selective AC17 proof of AC16

We briefly review the selective AC17 [4] proof of the AC16 scheme [1]. We define for each $j \in [\hat{n}_1]$ the n_k-degree polynomial

$$\hat{G}_j(x_{\mathrm{att}}) = \sum_{i\in[n_k], k\in[\hat{n}_2]} (x_{\mathrm{att}}^i - \rho(j)^i) \mathbf{1}^{d_1\times d_2}_{(i,j,k),1} = \sum_{i=0}^{n_k} c'_{i,j} x_{\mathrm{att}}^i,$$

where the coefficients $c'_{0,j} = -\sum_{i\in[n_k], k\in[n_2]} \rho(j)^i \mathbf{1}^{d_1\times d_2}_{(i,j,k),1}$ for all $j \in [\hat{n}_1]$, and $c'_{i,j} = \sum_{k\in[n_2]} \mathbf{1}^{d_1\times d_2}_{(i,j,k),1}$ for $i \in [n_k]$, for which $\hat{G}_j(\rho(j)) = 0$. Then, the "public keys" are substituted as follows:

$$b_{j,k} \ : \ \mathbf{1}^{d_1\times d_2}_{(j,k),1}, \quad b'_{i,j} \ : \ \sum_{k\in[\hat{n}_2]} A_{j,k} \mathbf{1}^{d_1\times d_2}_{(i,j,k),1}$$
$$b'_{0,j} \ : \ -\sum_{k\in[\hat{n}_2]} A_{j,k} \left(\sum_{i\in[n_k]} \mathbf{1}^{d_1\times d_2}_{(i,j,k),1} + \mathbf{1}^{d_1\times d_2}_{(j,k),1} \right)$$

for all $i \in [n_k], j \in [\hat{n}_1], k \in [\hat{n}_2]$.

In the "key query phase", i.e., EncR, we divide the set of rows $[\hat{n}_1]$ in $\Upsilon = \{j \in [\hat{n}_1] \mid \rho(j) \in \mathcal{S}\}$ and $\overline{\Upsilon} = [\hat{n}_1] \setminus \Upsilon$. Because $\mathcal{S}$ does not satisfy $\mathbb{A}$, there exists $\mathbf{w} = (1, w_2, ..., w_{n_1}) \in \mathbb{Z}_p^{n_1}$ such that $\mathbf{A}_j \mathbf{w}^\intercal = 0$ for all $j \in \Upsilon$. Then, the randomness r_j contains $\mathbf{w}$ and, if $j \in \overline{\Upsilon}$, the "re-programming" of the polynomial $\hat{G}_j$ to the n_k-degree polynomial $G(x_{\mathrm{att}}) = \prod_{\mathrm{att}'\in\mathcal{S}} (x_{\mathrm{att}} - x_{\mathrm{att}'}) = \sum_{i=0}^{n_k} u_i x_{\mathrm{att}}^i$,

which has roots in the set $\mathcal{S}$, divided by $G(\rho(j))$ (which is not equal to 0 because $\rho(j) \notin \mathcal{S}$). This is done by substituting

$$r_j \ : \ \begin{cases} \sum_{k' \in [\hat{n}_2]} w_{k'} \left(\mathbf{1}^{d_1}_{(j,k')} - \sum_{i' \in [n_k]} \frac{u_{i'} \mathbf{1}^{d_1}_{(i',j,k')}}{G(\rho(j))} \right) & \text{if } \rho(j) \in \overline{\Upsilon} \\ \sum_{k' \in [\hat{n}_2]} w_{k'} \mathbf{1}^{d_1}_{(j,k')} & \text{otherwise} \end{cases}.$$

Intuitively, this ensures that $r_j b_{j,1} - \alpha$ (where α is substituted by $w_k \overline{\mathbf{1}}^{d_2}_1$) evaluates to $\mathbf{0}$ due to the part of r_j that embeds $\sum_{k' \in [\hat{n}_2]} w_{k'} \mathbf{1}^{d_1}_{(j,k')}$, and $r_j \sum_{i \in \overline{[n_k]}} b'_{i,j} x^i_{\text{att}}$ goes to 0, either because $\mathbf{A}_j \mathbf{w}^\intercal \overline{\mathbf{1}}^{d_2}_1 = \mathbf{0}$ (in the case that $j \in \Upsilon$) or because $\mathbf{A}_j \mathbf{w}^\intercal \overline{\mathbf{1}}^{d_2}_1$ is canceled by the other part.

In the "challenge phase", i.e., EncS, we substitute $s = \overline{\mathbf{1}}^{d_2}_1$, so the encoding associated with C'' of the AC16 ciphertext goes to $\mathbf{0}$, because $sA_{j,k} b_{j,k}$ cancels out the part in $s \sum_{i \in \overline{[n_k]}} b'_{i,j} \rho(j)^i$ associated with $\mathbf{1}^{d_1 \times d_2}_{(j,k),1}$, which then yields $\sum_{k \in [\hat{n}_2]} A_{j,k} \overline{\mathbf{1}}^{d_2}_1 \hat{G}_j(\rho(j)) = \mathbf{0}$, due to the definition of $\hat{G}_j$.

C More details on the security proof

We show that the polynomials in the selective and co-selective property proofs in Section 5 indeed evaluate to $\mathbf{0}$.

C.1 The selective property

The polynomials $k_{2,j,k,l}, k_{3,j,j',k,l}, k_{4,j,\text{att}}$ and $k_{5,i,j,j',l}$, and c' in Section 5.2 evaluate to $\mathbf{0}$. For $k_{3,j,j',k,l}$ and $k_{5,i,j,j',l}$, this follows trivially. Then,

$$k_{2,j,k,l} = r_{j,l} b_{j,k} - v_k = - \sum_{k' \in \chi_{2,k}} w_k \overline{\mathbf{1}}^{d_2}_{\tau_2(k')} + w_k \sum_{k' \in \chi_{2,k}} \overline{\mathbf{1}}^{d_2}_{\tau_2(k')} = \mathbf{0},$$

and

$$\begin{aligned} k_{4,j,\text{att}} &= r_{j,\iota(\text{att})} \sum_{i \in \overline{[n_k]}} x^i_{\text{att}} b'_{i,j} \\ &= r_{j,\iota(\text{att})} \sum_{j' \in \chi_{1,j}, k' \in [n_2]} A_{j',k'} \left(\hat{G}_{j',k'}(x_{\text{att}}) + \sum_{i' \in [n_k]} \mathbf{1}^{d_1 \times d_2}_{(j,k'),\tau_1(j')} \right) \\ &= \sum_{j' \in \chi_{1,j}, k' \in [n_2]} A_{j',k'} w_{k'} \left(\frac{G_{\iota(\text{att})}(x_{\text{att}}) - G_{\iota(\text{att})}(\rho(j'))}{G_{\iota(\text{att})}(\rho(j'))} + 1 \right) \overline{\mathbf{1}}^{d_2}_{\tau_1(j')} \\ &= \sum_{j' \in \chi_{1,j}, k' \in [n_2]} A_{j',k'} w_{k'} \left(-1 + 1 \right) \overline{\mathbf{1}}^{d_2}_{\tau_1(j')} = \mathbf{0}, \end{aligned}$$

and $c' = c_1 + c_2$, where

$$\begin{aligned} c_1 &= \sum_{j\in[n_1],k\in[n_2]} s_{\tau_2(k)} A_{j,k} b_{\hat{\tau}_1(j),\hat{\tau}_2(k)} \\ &= -\sum_{j\in[n_1],k\in[n_2],k'\in\chi_{2,\hat{\tau}_2(k)}} A_{j,k} \mathbf{1}^{d_1\times d_2}_{(\hat{\tau}_1(j),\hat{\tau}_2(k)),\tau_2(k')} \overline{\mathbf{1}}^{d_2}_{\tau_2(k)} \\ &= -\sum_{j\in[n_1],k\in[n_2],k'\in\chi_{2,\hat{\tau}_2(k)}} A_{j,k} \mathbf{1}^{d_1}_{(\hat{\tau}_1(j),\hat{\tau}_2(k))}, \end{aligned}$$

cancels out

$$\begin{aligned} c_2 &= \sum_{i\in\overline{[n_k]},j\in[n_2]} s_{\tau_1(j)} \rho(j)^i b'_{i,\hat{\tau}_1(j)} \\ &= \sum_{j\in[n_2],j'\in\chi_{1,\hat{\tau}_1(j)},k'\in[n_2]} A_{j',k'} \hat{G}_{j',k'}(\rho(j)) \overline{\mathbf{1}}^{d_2}_{\tau_1(j)} \\ &+ \sum_{j\in[n_2],j'\in\chi_{1,\hat{\tau}_1(j)},k'\in[n_2]} A_{j',k'} \mathbf{1}^{d_1\times d_2}_{(j',k'),\tau_1(j')} \overline{\mathbf{1}}^{d_2}_{\tau_1(j)} \\ &= \sum_{j\in[n_2],k'\in[n_2]} A_{j,k'} \hat{G}_{j,k'}(\rho(j)) \overline{\mathbf{1}}^{d_2}_{\tau_1(j)} - c_1 = -c_1. \end{aligned}$$

C.2 The co-selective property

If we substitute the values in the key and ciphertext polynomials in Section 5.3, we observe that they evaluate to $\mathbf{0}$. This follows trivially for $k_{3,j,j',k,l}$ and $k_{5,i,j,j',l}$, and for the others we have:

$$\begin{aligned} k_{2,j,k,l} &= r_{j,l} b_{j,k} - v_k \\ &= \mathbf{1}^{d_1}_{j,l} \sum_{l\in[m]} \mathbf{1}^{d_1\times d_2}_{(j,l),(2,k)} - \overline{\mathbf{1}}^{d_2}_{(2,k)} = \overline{\mathbf{1}}^{d_2}_{(2,k)} - \overline{\mathbf{1}}^{d_2}_{(2,k)} = \mathbf{0}, \end{aligned}$$

and

$$\begin{aligned} k_{4,j,\mathrm{att}} &= r_{j,\iota(\mathrm{att})} \sum_{i\in\overline{[n_k]}} x^i_{\mathrm{att}} b'_{i,j} = \mathbf{1}^{d_1}_{j,\iota(\mathrm{att})} \sum_{i\in\overline{[n_k]},l\in[m]} u_{i,l} x^i_{\mathrm{att}} \mathbf{1}^{d_1\times d_2}_{(j,l),(1,j,l)} \\ &= \sum_{i\in\overline{[n_k]}} u_{i,\iota(\mathrm{att})} x^i_{\mathrm{att}} \overline{\mathbf{1}}^{d_2}_{(1,j,\iota(\mathrm{att}))} = G_{\iota(\mathrm{att})}(x_{\mathrm{att}}) \overline{\mathbf{1}}^{d_2}_{(1,j,\iota(\mathrm{att}))} = \mathbf{0}, \end{aligned}$$

and we split $c' = c_1 + c_2$ into two parts again:

$$\begin{aligned} c_1 &= \sum_{j\in[n_1],k\in[n_2]} s_{\tau_2(k)} A_{j,k} b_{\hat{\tau}_1(j),\hat{\tau}_2(k)} \\ &= \sum_{j\in[n_1],k\in[n_2],k'\in\tau_2^{-1}(\tau_2(k)),l\in[m]} A_{j,k} w_{k'} \mathbf{1}^{d_1}_{(\hat{\tau}_1(j),l),(2,\hat{\tau}_2(k))} \overline{\mathbf{1}}^{d_1}_{2,\hat{\tau}_2(k')} \end{aligned}$$

$$= \sum_{j\in[n_1],k\in[n_2],l\in[m]} A_{j,k} w_k \mathbf{1}^{d_1}_{(\hat{\tau}_1(j),l)} = \sum_{j\in[n_1],l\in[m]} \mathbf{A}_j \mathbf{w}^\intercal \mathbf{1}^{d_1}_{(\hat{\tau}_1(j),l)},$$

which is canceled out by

$$c_2 = \sum_{i\in\overline{[n_k]},j\in[n_1]} s_{\tau_1(j)} \rho(j)^i b'_{i,\hat{\tau}_1(j)}$$

$$= - \sum_{\substack{i\in\overline{[n_k]},j\in[n_1],\\ j'\in\tau_1^{-1}(\tau_1(j))\cap\overline{\Upsilon},\\ l,l'\in[m]}} \frac{\mathbf{A}_{j'} \mathbf{w}^\intercal u_{i,l} \rho(j)^i}{G_{l'}(\rho(j'))} \mathbf{1}^{d_1\times d_2}_{(\hat{\tau}_1(j),l),(1,\hat{\tau}_1(j),l)} \overline{\mathbf{1}}^{d_2}_{(1,\hat{\tau}_1(j'),l')}$$

$$= - \sum_{j\in[n_1],l\in[m]} \frac{\mathbf{A}_{j'} \mathbf{w}^\intercal G_l(\rho(j))}{G_l(\rho(j))} \mathbf{1}^{d_1}_{(\hat{\tau}_1(j),l)} = -c_1.$$

D RW13 in the asymmetric setting

We give the definition of the RW13 [45] scheme. 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 $\mathbb{G}$ as possible.

Definition 9 (The RW13 scheme [45]). *The RW13 ciphertext-policy attribute-based encryption scheme with optimized encryption (as well as most balanced encryption/decryption) is defined as follows.*

- Setup(λ)*: On input the security parameter λ, the algorithm 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 $\mathcal{U} = \mathbb{Z}_p$. 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 the 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(MSK, $\mathcal{S}$)*: On input a set of attributes $\mathcal{S}$, the algorithm generates random integers $r, r_{\mathrm{att}} \in_R \mathbb{Z}_p$ for each* att $\in \mathcal{S}$*, lets $x_{\mathrm{att}} \in \mathbb{Z}_p$ denote the representation of* att *in $\mathbb{Z}_p$ and computes the secret key as*

$$\mathrm{SK}_\mathcal{S} = (K = h^{\alpha - rb}, K' = h^r,$$
$$\{K_{1,\mathrm{att}} = h^{r_{\mathrm{att}}(b_1 x_{\mathrm{att}} + b_0) + rb'}, K_{2,\mathrm{att}} = h^{r_{\mathrm{att}}}\}_{\mathrm{att}\in\mathcal{S}}).$$

- Encrypt(M, MPK, $\mathbb{A}$)*: 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 by computing the ciphertext as*

$$\mathrm{CT}_\mathbb{A} = (C = M \cdot A^s, C' = g^s, \{C_{1,j} = B^{\lambda_j} (B')^{s_j},$$

Table 2: The performance of various algorithms on the BLS12-381 elliptic curve, expressed in the number of 10^3 cycles.

Algorithm	In $\mathbb{G}$	In $\mathbb{H}$	In $\mathbb{G}_T$
Exponentiation	344	696	1163
Fixed-base exponentiation	183	416	-
Multi-base exponentiation ($n+1$ bases)	$344+101n$	$696+602n$	-
Hash	101	602	-

Pairing operation	
Single	2159
Product of $n+1$	$2159+633n$

$$C_{2,j} = \left(B_1^{\rho(j)} B_0\right)^{s_j}, C_{3,j} = g^{s_j}\}_{j\in[1,n_1]}),$$

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 2). Then, the plaintext M is retrieved by computing*

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

E Benchmarks in RELIC

Our benchmarks are summarized in Table 2.

F More details on the performance analysis

F.1 Optimizing encryption versus decryption

The chosen "optimization approach" affects the distribution of the key and ciphertext components over $\mathbb{G}$ and $\mathbb{H}$. Generally, operations in $\mathbb{G}$ are much more efficient than in $\mathbb{H}$. Thus, if we want to optimize the encryption costs, we place as many ciphertext components in $\mathbb{G}$ as possible. Consequently, the key components that are paired with these ciphertext components are all in $\mathbb{H}$. Not only does this affect the key generation efficiency, but it also affects the decryption efficiency, and most obviously in the computation of C_5, which can be computed most efficiently as

$$\prod_{l_1\in[m_1']} e\left(\prod_{\substack{i\in[\overline{n_k}],\\ j'\in\tau_1^{-1}(l_1)}}\left(\prod_{j\in\Upsilon\setminus\{j'\}} K_{5,i,\hat{\tau}_1(j),\hat{\tau}_1(j'),\iota(\rho(j))}^{\varepsilon_j}\right)^{\rho(j')^i}, C_{l_1}\right)$$

which costs at most m'_1 pairing operations, $n_k n_1$ exponentiations and $(|\Upsilon| - 2)n_k n_1$ multiplications in the group in which the associated key components live. If these key components live in $\mathbb{H}$ (like in Definition 7), a large part of the decryption costs are several times as expensive as when these components had lived in $\mathbb{G}$. On the other hand, if we put C_{l_1} in $\mathbb{H}$, this would impact only one exponentiation per $\hat{n}_1$ attributes during encryption. Similarly, computing C_3 in the decryption algorithm requires at most $|\Upsilon|\hat{n}_1 n_2$ multiplications in the group in which C_3 lives, which blows up quadratically for large access policies.

F.2 Optimizing the order of computations

The order in which the computations of the algorithms are executed influences the efficiency. We can reduce the number of pairing operations required to decrypt by analyzing the pairing operations that share an argument [44]. Similarly, we can change the order of exponentiations, which is most notable in the ciphertext component C'. We can compute C' in two ways. Depending on the choices of $\hat{n}_1, \hat{n}_2, n_k$ and the number of ciphertext partitions, either way may be more efficient. First, we split C' in two parts:

$$C' = \underbrace{\prod_{j\in[n_1],k\in[n_2]} B_{\hat{\tau}_1(j),\hat{\tau}_2(k)}^{s_{\tau_2(k)}A_{j,k}}}_{C'_1} \cdot \underbrace{\prod_{i\in\overline{[n_k]},j\in[n_1]} (B'_{i,\hat{\tau}_1(j)})^{s_{\tau_1(j)}\rho(j)^i}}_{C'_2}.$$

Then, C'_1 can be computed as

$$\prod_{l_2\in[m'_2]} \left(\prod_{j\in[n_1],k\in\tau_2^{-1}(l_2)} B_{\hat{\tau}_1(j),\hat{\tau}_2(k)}^{A_{j,k}} \right)^{s_{l_2}},$$

which costs m'_2 exponentiations, and at most $n_1 n_2$ multiplications, or as

$$\prod_{j\in[\hat{n}_1],k\in[\hat{n}_2]} B_{j,k}^{\sum_{j'\in\hat{\tau}_1^{-1}(j),k'\in\hat{\tau}_2^{-1}(k)} s_{\tau_2(k')}A_{j',k'}},$$

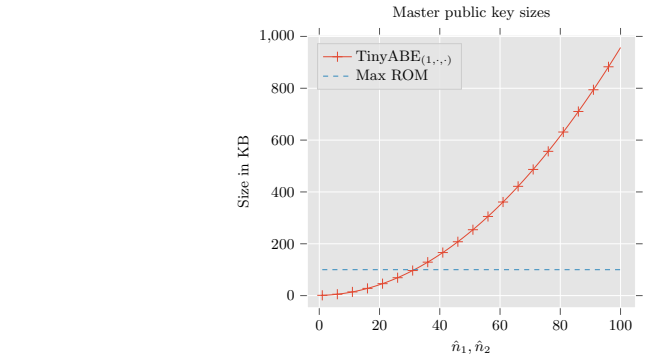
which costs $\hat{n}_1\hat{n}_2$ exponentiations.

Further, the computation of C'_2 requires $n_k \min(\hat{n}_1, n_1)$ exponentiations for the $i > 0$ part, and for the $i = 0$ part, we either compute

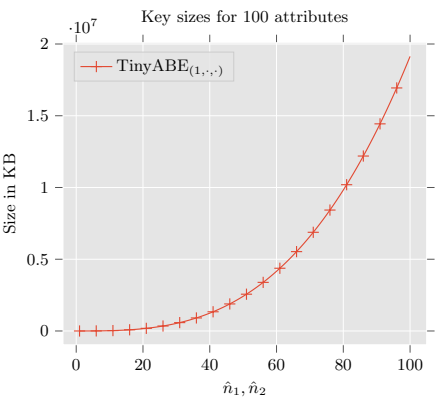
$$\prod_{j\in[\hat{n}_1]} (B'_{0,j})^{\sum_{j'\in\hat{\tau}^{-1}(j)} s_{\tau_1(j')}} \text{ or } \prod_{l_1\in[m'_1]} \left(\prod_{j\in\tau_1^{-1}(l_1)} B'_{0,\hat{\tau}_1(j)} \right)^{s_{l_1}},$$

which costs either $\hat{n}_1$ exponentiations, or m'_1 exponentiations and at most n_1 multiplications.

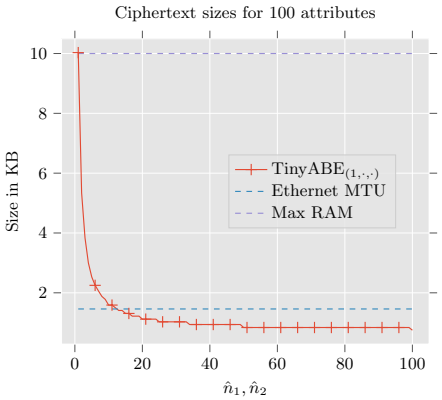
Interestingly, this analysis shows that the cost of computing C' can be upper-bounded in parameters that are fixed after the setup. That is, the computation of C' requires at most $\hat{n}_1(n_k + 1 + \hat{n}_2)$ exponentiations, regardless of the size of the access policy. This is the main reason why our encryption is so efficient when these parameters are small.



(a) Master public keys



(b) Secret keys



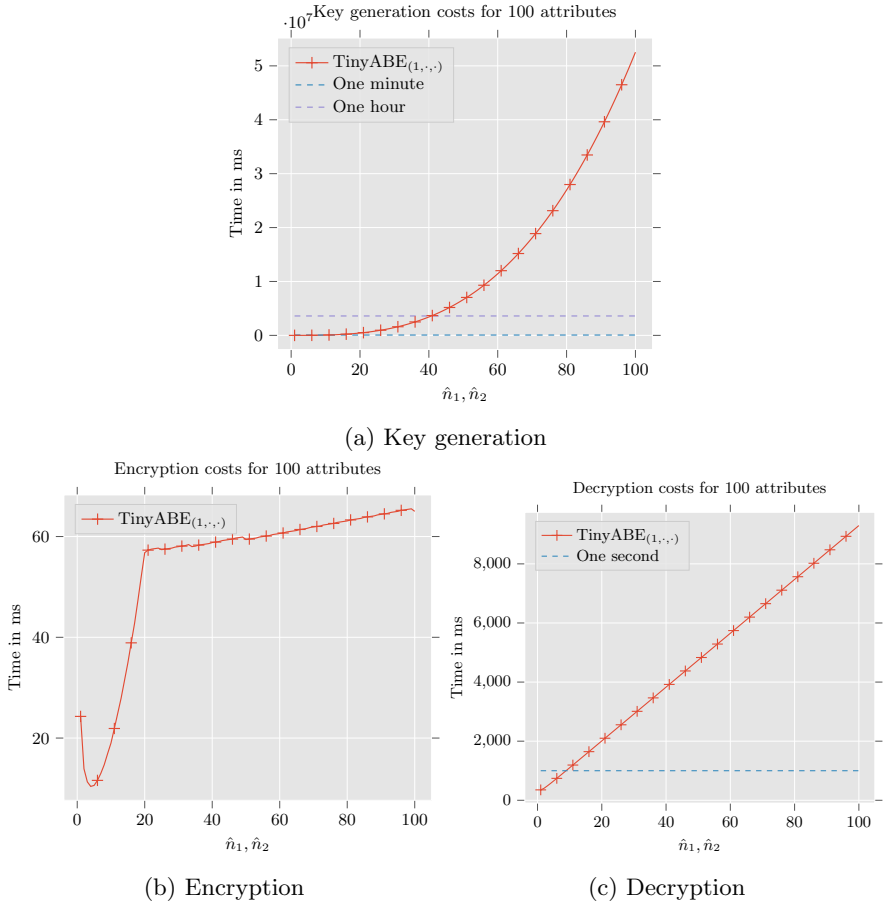
(c) Ciphertexts

Fig. 5: The sizes of the keys and ciphertexts for various choices of parameters $\hat{n}_1 = \hat{n}_2$, of the variants with optimized encryption (OE) and balanced encryption-decryption (BD) efficiency.

F.3 Selecting suitable parameters

To select suitable parameters, we analyze the computational costs of our scheme for various parameter choices.

Storage efficiency. We analyze the sizes of the master public keys, secret keys and ciphertexts for various choices of $\hat{n}_1$ in our scheme. This illustrates the increase of the key sizes versus the decrease in the ciphertext size. As Figure 5 shows, the sizes of the keys grow quadratically. In contrast, the effect on the ciphertext sizes of our scheme is very limited for large $\hat{n}_1$: even for small choices of $\hat{n}_1$, the ciphertext sizes are sufficiently small to e.g., fit in RAM of constrained devices or in one or two Ethernet packets during transmission.



(a) Key generation

(b) Encryption

(c) Decryption

Fig. 6: The computational costs of key generation, encryption and decryption for various choices of parameters $\hat{n}_1 = \hat{n}_2$.

Computational efficiency. Similarly, the computational efficiency of our scheme depends heavily on the choice of $\hat{n}_1$. To illustrate this, we analyze the computational costs of the key generation, encryption and decryption algorithms for various choices of $\hat{n}_1$ in our scheme. As Figure 6 shows, the computational costs of encryption and decryption are more or less optimal for $\hat{n}_1 \in [3, 10]$. Furthermore, for larger choices of $\hat{n}_1$, the key generation costs are extremely high. For $\hat{n}_1 > 30$, the key generation algorithm takes at least ten seconds per ten attributes. Interestingly, the encryption costs of our scheme are optimal for $\hat{n}_1 \approx 3$ (due to the upper bound on the computational costs of C'). In addition, the computational costs are still quite low for $\hat{n}_1 \approx 13$, while the ciphertexts fit into one Ethernet packet for up to 100 attributes.

The parameter sets. Based on our analysis, we select two sets of parameters $(n_k, \hat{n}_1, \hat{n}_2)$ for our comparison with RW13, for which

1. encryption is the most efficient: $(n_k, \hat{n}_1, \hat{n}_2) = (1, 3, 3)$;
2. ciphertexts are small, and encryption is efficient: $(n_k, \hat{n}_1, \hat{n}_2) = (1, 13, 13)$.

On the key generation costs. In general, encryption is the most efficient when $n_k = 1$. This comes however at a cost: the key generation is then the least efficient. To illustrate the effect of picking n_k higher, we also analyze the costs for variable n_k, and $(\hat{n}_1, \hat{n}_2) = (13, 13)$ in Appendix F.5. As Figures 8 and 9 show, the key sizes and key generation efficiency are the most optimal when $|\mathcal{S}| = n_k$. However, the effect is rather minimal: the sizes and computational costs are roughly halved. In turn, the sizes of the master public keys increase, as well as the computational costs of encryption and decryption. It is more effective to use an online/offline variant of the algorithm [33], which allows the key generation authority to precompute large batches of key material (Appendix G).

F.4 On the efficiency of CP-ABE with linear-size ciphertexts

As mentioned in Section 6.2, RW13 is the large-universe CP-ABE scheme with the most efficient algorithm. If we compare FAME [2] and the selective variants of BSW07 [16] and ABGW17 [6] transformed to the full security setting [3], then we observe that they incur the following encryption costs:

- **BSW07:** 1 exponentiation in $\mathbb{G}_T$, $2n_1$ regular exponentiations and 2 fixed-base exponentiations in $\mathbb{G}$, $2n_1$ hashes in $\mathbb{G}$, and $2n_1$ fixed-base exponentiations in $\mathbb{H}$;
- **FAME:** 2 exponentiations in $\mathbb{G}_T$, 3 exponentiations in $\mathbb{H}$, at least $3n_1(2+n_2)$ hashes in $\mathbb{G}$ and $3n_1$ two-base exponentiations in $\mathbb{G}$;
- **ABGW17:** 1 exponentiation in $\mathbb{G}_T$, $4n_1$ two-base exponentiations in $\mathbb{G}$ and $2n_1$ fixed-base exponentiations in $\mathbb{G}$.

Note that the ABGW17 costs are the same as the RW13 costs. The costs of the schemes are depicted in Figure 7. It shows that BSW07 is only marginally faster than RW13, and in particular, that our scheme outperforms all linear-sized schemes for both configurations.

F.5 On the effect of n_k on the efficiency

In Figures 8 and 9, the storage and computational costs of our scheme are depicted for various choices of n_k. This illustrates the decreased costs of key generation for higher choices of n_k, versus the increased costs of encryption and decryption.

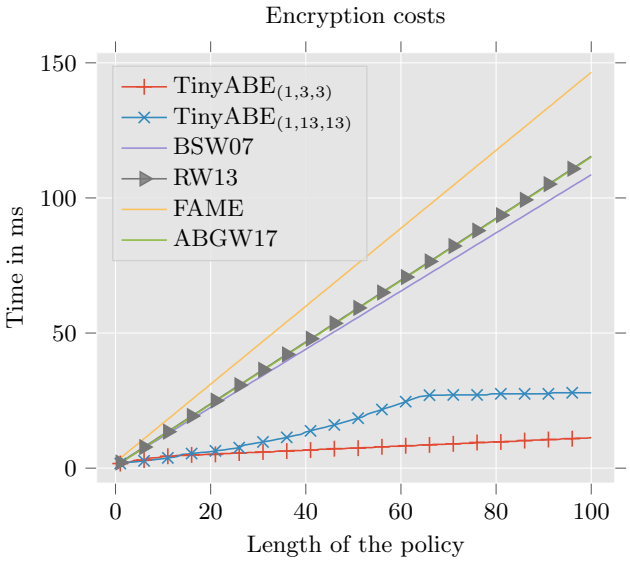


Fig. 7: The encryption costs of various CP-ABE schemes.

G Online/offline key generation

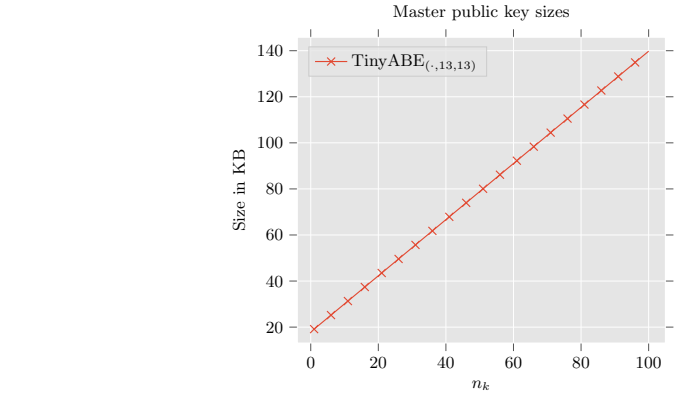
The biggest performance bottleneck of our scheme is the inefficiency of the key generation algorithm. To mitigate this issue, we can split the key generation in an online and offline phase, such as the HW14 [33] scheme does with the RW13 [45] scheme. In this way, the key generation authority can precompute many preliminary secret keys so that during an access request, it requires very little computational power. After the user has received her keys, she can compute the secret keys as in the original construction (Definition 7), and thus perform decryption as she usually would.

– **Offline key generation:** This algorithm generates an "intermediate secret key" for one partition by generating random integers $r_j, \hat{v}_k \in_R \mathbb{Z}_p$ for each $j \in [\hat{n}_1], k \in [2, \hat{n}_2]$ and random integers $z_i \in_R \mathbb{Z}_p$ for all $i \in [n_k]$, by setting $\hat{v}_1 = \alpha$, and by computing the intermediate secret key as

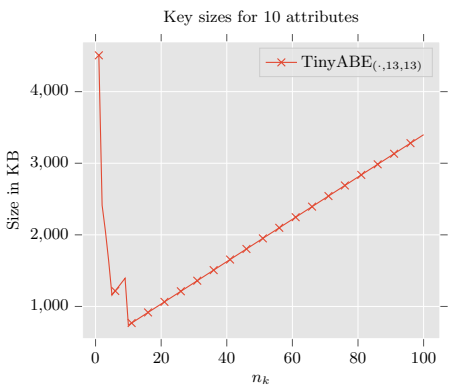
$$\hat{\mathrm{SK}} = (\{K_{1,j} = g^{r_j}, \hat{K}_{2,j,k} = g^{r_j b_{j,k} - \hat{v}_k}, K_{3,j,j',k} = g^{r_j b_{j',k}}, \hat{K}'_{4,j,i'} = g^{r_j z_{i'}},$$
$$K_{5,i,j,j'} = g^{r_j b_{i,j'}}, r_j, \hat{v}_k, z_{i'}\}_{i \in \overline{[n_k]}, i' \in [n_k], j,j' \in [\hat{n}_1], j \neq j', k \in [\hat{n}_2]}).$$

– **Online key generation:** On input a set of attributes $\mathcal{S}$, this algorithm computes $m = \left\lceil \frac{|\mathcal{S}|}{n_k} \right\rceil$, defining $\iota \colon \mathcal{S} \to [m]$ such that $|\iota^{-1}(i)| \leq n_k$ for each $i \in [m]$, and selecting m intermediate secret keys

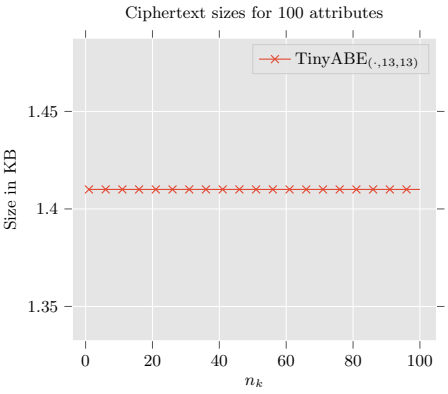
$$\hat{\mathrm{SK}}_l = (\{K_{1,j,l}, \hat{K}_{2,j,k,l}, K_{3,j,j',k,l}, \hat{K}'_{4,j,i',l}, K_{5,i,j,j',l},$$



(a) Master public keys



(b) Secret keys



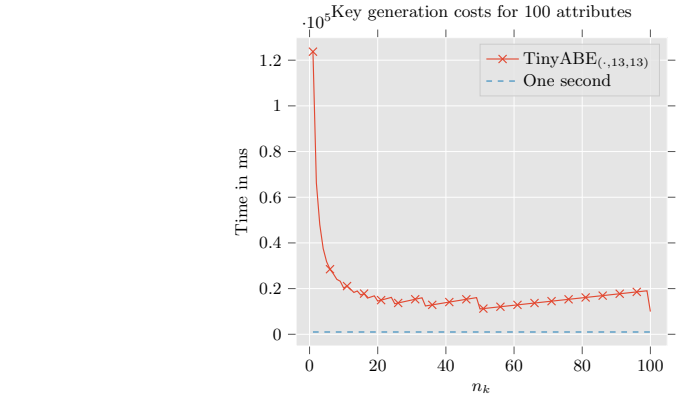
(c) Ciphertexts

Fig. 8: The sizes of the keys and ciphertexts for various choices of n_k.

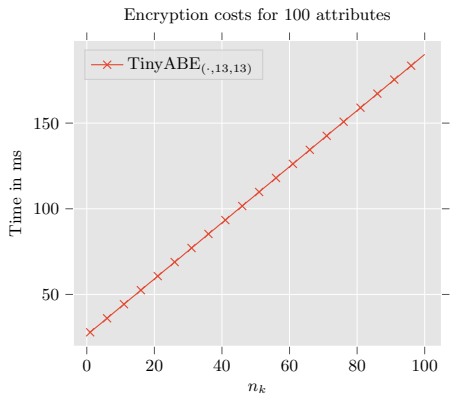
$$r_{j,l}, \hat{v}_{k,l}, z_{i',l}\}_{i\in\overline{[n_k]},i'\in[n_k],j,j'\in[\hat{n}_1],j\neq j',k\in[\hat{n}_2]}).$$

Then, it also injectively maps all the attributes in the same partition $l \in [m]$ to the "intermediate attributes" $[n_k]$ by defining a mapping $\gamma\colon \mathcal{S} \to [n_k]$ such that γ is injective on subdomain $\iota^{-1}(l)$ for each $l \in [m]$, and it chooses random $v_k \in_R \mathbb{Z}_p$ for each $k \in [2, n_2]$, and then computes

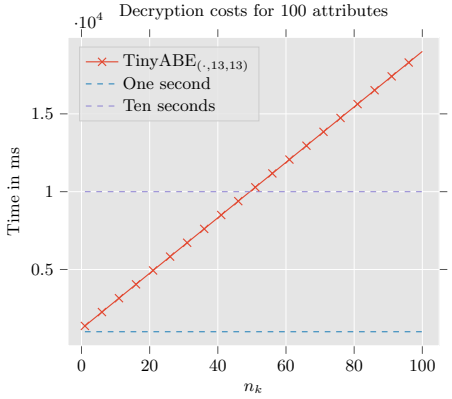
$$\hat{K}_{6,j,\mathrm{att}} = r_{j,\iota(\mathrm{att})} \left(\sum_{i\in\overline{[n_k]}} x^i_{\mathrm{att}} b'_{i,j} - z_{\gamma(\mathrm{att}),\iota(\mathrm{att})} \right)$$
$$\hat{K}_{7,k,l} = \hat{v}_{k,l} - v_k,$$
$$\hat{K}_{4,j,\mathrm{att}} = \hat{K}'_{4,j,\gamma(\mathrm{att}),\iota(\mathrm{att})}$$



(a) Key generation



(b) Encryption



(c) Decryption

Fig. 9: The computational costs of key generation, encryption and decryption for various choices of n_k.

for all $j \in [n_1], k \in [2, n_2], l \in [m], \mathrm{att} \in \mathcal{S}$. It returns the secret key as

$$\hat{\mathrm{SK}}_{\mathcal{S}} = (\{K_{1,j,l}, \hat{K}_{2,j,k,l}, K_{3,j,j',k,l}, \hat{K}_{4,j,\mathrm{att}}, K_{5,i,j,j',l}, \hat{K}_{6,j,\mathrm{att}}, \hat{K}_{7,k,l}\}_{\substack{i \in \overline{[n_k]}, j,j' \in [\hat{n}_1], j \neq j', \\ k \in [\hat{n}_2], l \in [m], \mathrm{att} \in \mathcal{S}}}).$$

– **Final step key generation:** After receiving the intermediate secret key $\hat{\mathrm{SK}}_{\mathcal{S}}$, the user can compute her secret key by first computing

$$K_{2,j,k,l} = \hat{K}_{2,j,k,l} \cdot g^{\hat{K}_{7,k,l}}, \qquad K_{4,j,\mathrm{att}} = \hat{K}_{4,j,\mathrm{att}} \cdot g^{\hat{K}_{6,j,\mathrm{att}}},$$

and then retrieving the secret key as in Definition 7 as

$$\mathrm{SK}_{\mathcal{S}} = (\{K_{1,j,l}, K_{2,j,k,l}, K_{3,j,j',k,l}, K_{4,j,\mathrm{att}}, K_{5,i,j,j',l}\}_{i \in \overline{[n_k]}, j,j' \in [\hat{n}_1], j \neq j', k \in [\hat{n}_2], l \in [m], \mathrm{att} \in \mathcal{S}}).$$

– **Correctness:** The final step of the key generation indeed yields the secret keys as defined in Definition 7:

$$K_{2,j,k,l} = \hat{K}_{2,j,k,l} \cdot g^{\hat{K}_{7,k,l}} = g^{r_{j,l} b_{j,k} - \hat{v}_{k,l}} \cdot g^{\hat{v}_{k,l} - v_k} = g^{r_{j,l} b_{j,k} - v_k}$$

$$K_{4,j,\mathrm{att}} = \hat{K}_{4,j,\mathrm{att}} \cdot g^{\hat{K}_{6,j,\mathrm{att}}}$$

$$= g^{r_{j,\iota(\mathrm{att})} z_{\gamma(\mathrm{att}),\iota(\mathrm{att})}} \cdot g^{r_{j,\iota(\mathrm{att})} \left(\sum_{i \in \overline{[n_k]}} x^i_{\mathrm{att}} b'_{i,j} - z_{\gamma(\mathrm{att}),\iota(\mathrm{att})} \right)}$$

$$= g^{r_{j,\iota(\mathrm{att})} \sum_{i \in \overline{[n_k]}} x^i_{\mathrm{att}} b'_{i,j}}.$$

– **Security:** This online/offline key generation yields a secure scheme. That is, if we can break the scheme with the online/offline key generation, then we can also break the scheme with a regular key generation. Suppose $\mathrm{SK}_{\mathcal{S}} =$

$$(\{K_{1,j,l}, K_{2,j,k,l}, K_{3,j,j',k,l}, K_{4,j,\mathrm{att}}, K_{5,i,j,j',l}\}_{\substack{i \in \overline{[n_k]}, j,j' \in [\hat{n}_1], j \neq j', \\ k \in [\hat{n}_2], l \in [m], \mathrm{att} \in \mathcal{S}}}).$$

is a secret key given in the security game of the regular scheme. Then we can generate a secret key in the security game of the online/offline extension of the scheme by computing $\hat{K}_{6,j,\mathrm{att}}, \hat{K}_{7,k,l} \in_R \mathbb{Z}_p$ for each $j \in [n_1], k \in [2, n_2], l \in [m], \mathrm{att} \in \mathcal{S}$ and setting $\hat{K}_{2,j,k,l} = K_{2,j,k,l} \cdot g^{-\hat{K}_{7,k,l}}$ and $\hat{K}_{4,j,\mathrm{att}} = K_{4,j,\mathrm{att}} \cdot g^{-\hat{K}_{6,j,\mathrm{att}}}$. Note that $\hat{K}_{6,j,\mathrm{att}}$ and $\hat{K}_{7,k,l}$ are correctly distributed due to the randomness of $z_{i',l}$ and $\hat{v}_{k,l}$. The secret key returned to the attacker that can break the online/offline extension of the scheme is

$$\hat{\mathrm{SK}}_{\mathcal{S}} = (\{K_{1,j,l}, \hat{K}_{2,j,k,l}, K_{3,j,j',k,l}, \hat{K}_{4,j,\mathrm{att}}, K_{5,i,j,j',l},$$
$$\hat{K}_{6,j,\mathrm{att}}, \hat{K}_{7,k,l}\}_{\substack{i \in \overline{[n_k]}, j,j' \in [\hat{n}_1], j \neq j', \\ k \in [\hat{n}_2], l \in [m], \mathrm{att} \in \mathcal{S}}}).$$

Table of Contents