Too Many Options: A Survey of ABE Libraries for Developers

Aintzane Mosteiro-Sancheza,b, Marc Barceloa, Jasone Astorgab, Aitor Urbietaa

aIkerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), P.J.M. Arizmendiarreria, 2, Arrasate-Mondragon, 20500, Spain
bUniversity of the Basque Country UPV/EHU, Alameda Urquijo s/n, Bilbao, 48013, Spain

Abstract

Attribute-based encryption (ABE) comprises a set of one-to-many encryption schemes that allow the encryption and decryption of data by associating it with access policies and attributes. Therefore, it is an asymmetric encryption scheme, and its computational requirements limit its deployment in IoT devices. There are different types of ABE and many schemes within each type. However, there is no consensus on the default library for ABE, and those that exist implement different schemes. Developers, therefore, face the challenge of balancing efficiency and security by choosing the suitable library for their projects. This paper studies eleven ABE libraries, analyzing their main features, the mathematical libraries used, and the ABE schemes they provide. The paper also presents an experimental analysis of the four libraries which are still maintained and identifies some of the insecure ABE schemes they implement. In this experimental analysis, we implement the schemes offered by these libraries, measuring their execution times on architectures with different capabilities, i.e., ARMv6 and ARMv8. The experiments provide developers with the necessary information to choose the most suitable library for their projects, according to objective and well-defined criteria.

Keywords: ABE, IoT, CP-ABE, IIoT, KP-ABE, dCP-ABE

1. Introduction

Attribute-Based Encryption (ABE) is a one-many encryption algorithm. This feature implies that the same piece of encrypted data can be shared with multiple users. Usually, the encryption of a Plaintext (PT) creates a personalized Ciphertext (CT) which only its unique intended user can decrypt. ABE breaks this limitation, improving encryption efficiency for one-to-many communication models. This feature is particularly relevant when the same information must be confidentially shared with multiple users, e.g., brokered communications in industrial environments.

ABE achieves the above feature by providing encryption schemes that bind information encryption and decryption to attributes (e.g., \( \hat{A} = [\text{att}_1, \text{att}_2, \text{att}_4, \text{att}_5] \)) and an Access Policy (AP) (e.g., \( \text{AP} = (\text{att}_1 \text{ AND att}_2) \text{ OR att}_3 \)). Since APs are defined according to attributes, access to information is only granted if the policy is fulfilled. As a result, ABE provides one-to-many encryption, offers implicit authorization to data, and decouples encryption and decryption from users’ identities.

Depending on how that fulfillment takes place, two variants of ABE have been defined: Key-Policy Attribute-Based Encryption (KP-ABE) and Ciphertext-Policy Attribute-Based Encryption (CP-ABE). KP-ABE uses APs to create a user’s Secret Key (\( SK \)). Meanwhile, CTs are generated according to attributes. If the attributes of the CT fulfill the AP of the user, the user gets access to information. CP-ABE does the opposite of KP-ABE: users receive SKs generated according to attributes, and information is protected according to AP. In both cases, attributes must satisfy the AP to decrypt the information.

Although the concept of fulfilling AP in order to access information is similar in both ABE schemes, it can be seen how CP-ABE allows the Data Owner (DO) to retain control over who can access the information. In distributed environments, it is more secure that whoever generates the information is the one who decides the access policy to it. Therefore CP-ABE is more suitable for these environments. Besides, defining users according to attributes makes the system more scalable since it is more manageable to update users’ privileges by adding or deleting attributes rather than redefining their AP by keeping a list of every resource they need to access.

There are numerous KP-ABE and CP-ABE schemes, and not all of them are implemented in every library. Therefore, knowing which libraries are more efficient or secure is a challenge. Choosing the proper ABE library lies in a trade-off between efficiency and security, which can be a challenge: Which ABE schemes do they offer? Which programming languages are available? Which mathematical libraries do they use? In order to provide answers to these questions, this article surveys existing cryptographic libraries and provides a fair comparison according to their security features and performance. The goal is to assist developers in choosing the most suitable library for their projects. This article is an extended version of a preliminary work presented in [I]. The contributions of this article can be summarized as:
• An experimental evaluation of ABE libraries, which is a practical tool to help developers choose the most suitable one for their projects.
• The methodology itself, which can be further used to evaluate new ABE schemes, increasing the scope of the assessment.

2. Background & Related Work

The first ABE algorithm was presented under the name of Fuzzy Identity-Based Encryption [2], giving more granularity to Identity-Based Encryption (IBE) schemes [3]. IBE schemes encrypt and decrypt information according to the receivers’ identities. The scheme presented in [2] provided more flexibility to encryption by using attributes instead of identities. Fuzzy Identity-Based Encryption eventually led to the term ABE and to encryption by using attributes instead of identities. Fuzzy Identity-Based Encryption [2], giving more granularity to ABE in 2019, called Charm [9], and CP-ABE [5], and even a decentralized version called Decentralized CP-ABE (dCP-ABE) [6].

However, for a cryptographic scheme to develop beyond academia, practical implementations should be available in the form of libraries. Developers use cryptography as a tool, so part of the popularity of certain schemes is based on the ease of obtaining a library. When Bethencourt et al. presented CP-ABE [5], they included the cpabe toolkit [7]. This library was, until 2011, the only library offering ABE schemes. However, in 2011 libfenc [8] appeared. This project reused part of the cpabe toolkit code but extended the functions and schemes of the library. These relationships are shown in Figure 1 by solid line arrows.

The same year libfenc appeared, the first version of Charm [9] was released. This library focuses on elliptic curve cryptography. Several libfenc developers have also participated in the development and maintenance of Charm, which, as of May 2022, is still being maintained and updated. On the other hand, it is also interesting to mention that some of the cpabe toolkit and Charm developers worked together and released a new library for ABE in 2019, called OpenABE [10]. This relationship of developers is shown in Figure 1 by dashed lines next to the name of the libraries.

In general, there are more ABE schemes in academia than libraries capable of implementing them because not every proposal includes a library. One of the exceptions is the case of BDABE [11], developed by Fraunhofer, which was accompanied by Rabe [12]. Moreover, Rabe implements several other ABE schemes, not only BDABE, and it is still maintained. Despite these advances, there is still a need for a production-grade library. Therefore, part of the results of the FENTEC project, which aimed to develop functional encryption systems, resulted in two ABE libraries, GoFE [13] and CiFEr [14], between 2018 and 2019. Both implement the same ABE schemes, but GoFE is implemented in Go and CiFEr in C.

To deploy ABE encryption schemes properly, their efficiency and complexity cannot have a negative impact on the system. Authors in [15] study the mathematical complexity of various CP-ABE schemes and propose different strategies to optimize them. They conclude that there is no generic solution capable of reducing the complexity of ABE schemes in every step. Instead, users must prioritize reducing the complexity of encryption, decryption, or key generation. Depending on the system requirements, authors suggest several mathematical improvements on the different ABE schemes.

Although mathematical efficiency is crucial for ABE schemes, their practical implementation is not always optimal. Dependencies on third parties, implementation errors, or programming language affect execution times. Furthermore, mathematical dependencies also affect the security of the implementations. Therefore, formal analyses of cryptographic schemes are insufficient, and a more practical approach is required.

A library’s execution time and security are crucial features to select it. Time can be measured by implementing the libraries, but security evaluations are more challenging. For this article, part of the analysis is based on the results of [16]. In the cited article, the researchers analyze several ABE schemes, study their vulnerabilities, propose new attacks, and provide an extensive list of vulnerable schemes.

3. Methodology

We provide two evaluations for ABE libraries: qualitative and experimental. Every library is qualitatively evaluated, but some are dismissed after the assessment and are not experimentally analyzed. This section defines which features are considered and describes the evaluation process (Figure 2).

• Maintained: We consider a library to be maintained when its main fork has had some update or upgrade in the last fourteen months as of May 2022.
• Recent Activity: We consider that there is recent activity when the main fork of the library has been maintained or if any branch has had activity in the last year.
• Broken Schemes: Some ABE schemes have been successfully attacked. The authors of [16] compile a list of 24 vulnerable ABE schemes. Interested readers are encouraged to consult Tables 1, 2, and 5 of the mentioned article for a more detailed analysis.
• Math Library: ABE schemes are based on elliptic curve cryptography, but do not require specific curves; instead, developers are the ones to choose them. Thus, the chosen mathematical library is crucial for ABE security and efficiency. Some mathematical libraries support outdated curves and should only be used in academic or experimental environments.
• Design Purpose: Not every library is intended or appropriate for production. Understanding the use of each library is critical for deploying a secure system. Libraries
4. Qualitative Evaluation

4.1. Generic Features

This section uses the criteria defined in the previous section to evaluate a total of 11 libraries. The information gathered about this libraries is presented in Table 1, which is analyzed according to the process shown in Figure 2. Information on whether there is an active community using the libraries has been included in the table. This is not such a crucial criterion as to merit inclusion in the selection process in Figure 2, but it may be of interest to developers. Note that an active community can provide technical support, even if the library is no longer maintained.

Table 1 shows that only three libraries continue to be maintained by the original developers in the main fork: Rabe, Charm and GoFE. Of the remainder, it should be noted that most had no activity in the main fork in almost two years. The exceptions to this are OpenABE and CiFEr. OpenABE was last updated in the main fork on January 2021 and CiFEr on February 2021. However, both of them have had recent activity in new forks. In the case of OpenABE fork 2, it solves installation issues and updates dependencies like OpenSSL. In the case of the new CiFEr fork 3, it has received minor updates. It should also be noted that CiFEr is the twin of GoFE but implemented in C instead of Golang. In other words, although the main branch of CiFEr has no activity, the project is still going on. Therefore, this article considers that all five libraries, i.e., OpenABE, Rabe, Charm, GoFE, and CiFEr, have had recent activity.

Regarding community activity, Rabe and Charm are the most active, although in different ways. Rabe developers respond and resolve open issues in the repository and update the library with user-requested features 4. Charm is the most widely used framework for cryptographic prototyping, and as such, it

---

Table 1: Library analysis details

| Library        | Start | Maintained? | Recent Activity | Production | Design Goal | Old Math Library | Broken Schemes | Academic |
|----------------|-------|-------------|-----------------|------------|-------------|------------------|----------------|----------|
| cpabe toolkit  |       |             |                 |            |             |                  |                |          |
| libfenc        |       |             |                 |            |             |                  |                |          |
| Charm          |       |             |                 |            |             |                  |                |          |
| OpenABE        |       |             |                 |            |             |                  |                |          |
| cpabe          |       |             |                 |            |             |                  |                |          |
| JCPABE         |       |             |                 |            |             |                  |                |          |
| DET-ABE        |       |             |                 |            |             |                  |                |          |
| JPBC-FAME      |       |             |                 |            |             |                  |                |          |
| CiFEr          |       |             |                 |            |             |                  |                |          |
| GoFE           |       |             |                 |            |             |                  |                |          |
| Rabe           |       |             |                 |            |             |                  |                |          |

Figure 2: Library analysis process

not designed for production environments should not be considered in such scenarios.

All the libraries are studied according to these five features. Then, following the process shown in Figure 2, we choose the libraries to be experimentally analyzed. We consider that libraries implementing outdated curves are unsuitable for the production environment, regardless of their design goal.

---

2 https://github.com/StefanoBerlato/openabe
3 https://github.com/swanhong/CiFEr/tree/master
4 https://github.com/Fraunhofer-AISEC/rabe/issues/9

has an extensive community of users. An active community favors problem resolution, which makes it an interesting feature to be considered. However, it is not so critical that its absence implies discarding the library.

4.2. Security Features

In this section, we analyze those features directly affecting the libraries’ security: the chosen mathematical libraries, the implemented ABE schemes, and the AES schemes underneath.

Regarding the mathematical libraries used, as Table 1 showed, some libraries use PBC, which supports obsolete elliptic curves that currently are not considered secure enough for production. Therefore, we consider libraries using PBC only suitable for research. Java libraries use jPBC, the Java implementation of the original PBC, and therefore implements the same elliptic curves as PBC. Libraries like Charm use PBC but can be compiled with Mircl or Relic. Relic, PBC, and Mircl are pairing-based cryptographic libraries. However, Relic offers more efficient pairing constructions and faster implementations than PBC, and Mircl is newer and available in seven programming languages. Mircl is also the mathematical library used by CiFEr, albeit with a crucial nuance compared to Charm: Charm uses the maintained branch of Mircl. In contrast, CiFEr uses Mircl-AMCL, a non-maintained branch of Mircl. Moreover, CiFEr implements a reduced version of Mircl-AMCL, using the BN-254 curve for 64 bits. As a result, CiFEr can only be implemented in a 64-bit architecture, limiting its deployability. In contrast, Charm imposes no restrictions on the curve to use from Mircl-Core. Developers should take this into account and modify the files as necessary. With the architecture limitation of CiFEr, OpenABE becomes the only library written in C++ with no architecture limitation. A C++ library is especially relevant for Internet of Things (IoT) devices with minimal computational power. This, as well as some embedded devices, benefit from the use of C and C++.

Regarding “Design Goal,” we refer to research or production quality. We consider cryptographic libraries using obsolete curves suitable for research and academic purposes but not production. Furthermore, some developers flag their libraries as unsuitable for production, either because they have not been adequately tested or are still in an early stage of development. Thus, OpenABE, Rabe, and Charm are the libraries currently suitable for production while GoFE should still be limited to academic and research environments. However, as pointed out earlier, Charm should be compiled with Mircl or Relic, not PBC. With the four main libraries identified, we examine their implementation in Table 2. To enhance results’ readability, these four libraries have also been highlighted in bold in Table 2.

In terms of supported schemes, Charm and Rabe support YCT14, a KP-ABE scheme proposed in 2016, which was broken in 2019. It is also one of the broken schemes compiled in 2016. In addition, Charm also implements some vulnerable dCP-ABE schemes: YJ14 and DAC-MACs, both broken in 2016. Broken schemes are insecure and should never be used. However, both Charm and Rabe support other ABE schemes, so there is no need to discard them. Instead, developers should use these libraries cautiously and warrant that vulnerable schemes are not implemented. Another scheme that should be carefully considered is BDABE. This is the only ABE scheme designed to work with Blockchain. Thus, as the authors of the scheme explain, developers should consider that the scheme’s efficiency will be affected by the deployed Blockchain solution.

As mentioned earlier, ABE schemes are computationally heavy and are often used in hybrid mode. Hybrid mode consists of using a symmetric scheme (usually AES) to encrypt
Table 2: Comparison of libraries. Part II

| Name          | Docs. | Language | KP-ABE | CP-ABE | dCP-ABE |
|---------------|-------|----------|--------|--------|--------|
| OpenABE [10]  |       |          | □      | □      | □      |
| Rabe [12]     |       |          | □      | □      | □      |
| Charm [9]     |       | □        | □      | □      | □      |
| GoFE [13]     |       | □        | □      | □      | □      |

1. Yes  2. No

1. C++  2. Rust  3. Go  4. Python

1. GPSW06 [4]  2. W11 [24]  3. LSW10 [21]  4. YCT14 [22]  5. BSW07 [5]  6. W11 [24]  7. YCT14 [22]  8. FAME [23]  9. LSW10 [21]  10. OpenABE [10]  11. Rabe [12]  12. Charm [9]  13. GoFE [13]  14. TimePRE [27]  15. RW15 [31]  16. DAC-MACS [25]  17. YJ14 [30]  18. YAHK14 [25]  19. CGW15 [26]  20. GPSW06 [4]  21. LSW10 [21]  22. YCT14 [22]  23. FAME [23]  24. W11 [24]  25. YAHK14 [25]  26. CGW15 [26]  27. TimePRE [27]  28. LW11 [23]  29. FAME [23]  30. YJ14 [30]  31. RW15 [31]  32. YCT14 [22]  33. GPSW06 [4]  34. LSW10 [21]  35. YCT14 [22]  36. FAME [23]  37. W11 [24]  38. YAHK14 [25]  39. CGW15 [26]  40. TimePRE [27]  41. RW15 [31]  42. DAC-MACS [25]  43. YJ14 [30]  44. FAME [23]  45. Rabe [12]  46. Charm [9]  47. GoFE [13]  48. OpenABE [10]  49. Rabe [12]  50. Charm [9]  51. GoFE [13]  52. OpenABE [10]  53. Rabe [12]  54. Charm [9]  55. GoFE [13]  56. OpenABE [10]  57. Rabe [12]  58. Charm [9]  59. GoFE [13]  60. OpenABE [10]  61. Rabe [12]  62. Charm [9]  63. GoFE [13]  64. OpenABE [10]  65. Rabe [12]  66. Charm [9]  67. GoFE [13]  68. OpenABE [10]  69. Rabe [12]  70. Charm [9]  71. GoFE [13]  72. OpenABE [10]  73. Rabe [12]  74. Charm [9]  75. GoFE [13]  76. OpenABE [10]  77. Rabe [12]  78. Charm [9]  79. GoFE [13]  80. OpenABE [10]  81. Rabe [12]  82. Charm [9]  83. GoFE [13]  84. OpenABE [10]  85. Rabe [12]  86. Charm [9]  87. GoFE [13]  88. OpenABE [10]  89. Rabe [12]  90. Charm [9]  91. GoFE [13]  92. OpenABE [10]  93. Rabe [12]  94. Charm [9]  95. GoFE [13]  96. OpenABE [10]  97. Rabe [12]  98. Charm [9]  99. GoFE [13]  100. OpenABE [10]  101. Rabe [12]  102. Charm [9]  103. GoFE [13]  104. OpenABE [10]  105. Rabe [12]  106. Charm [9]  107. GoFE [13]  108. OpenABE [10]  109. Rabe [12]  110. Charm [9]  111. GoFE [13]  112. OpenABE [10]  113. Rabe [12]  114. Charm [9]  115. GoFE [13]

AES-CBC and AES-GCM are secure symmetric ciphers, but AES-GCM provides authenticated encryption. Using AES-GCM provides confidentiality and guarantees the integrity of the CT. This protection does not stop attackers from violating the integrity of the CT but allows the receiver to detect tampering. Distributed environments benefit from data integrity protection and should favor the use of AES-GCM. As can be seen from the libraries presented in Table 3, the only ones implementing AES-GCM are OpenABE and Rabe. It should be noted that Rabe has implemented AES-GCM in rabe-0.3.4. Previous versions used the low-level AES block cipher function. Those versions should only be applied in academic and research environments and avoided for production.

5. Quantitative Evaluation Definition

Following the qualitative evaluation, this section experimentally analyzes the four libraries identified as the most relevant from a developer’s point of view. As mentioned above, one of the challenges developers face when implementing ABE schemes is the lack of knowledge about the performance capabilities of the libraries. Therefore, this section studies and compares how libraries behave on devices with different computational capabilities. This behavior is quantified by measuring the time that each scheme of each library requires to perform the basic operations in ABE: encryption, decryption, key generation, and, in dCP-ABE, authority setup.

5.1. Experiment Definition

Our experiment relies on the different types of ABE schemes. The classification of CP-ABE, KP-ABE, and dCP-ABE is shown in Table 2. dCP-ABE schemes have a similar behavior to the conventional CP-ABE schemes, but the key generation is distributed among several key-generating authorities.

The performance comparison of different ABE schemes illustrates how practical aspects of the implementation, like the mathematical library or the implementation language, can affect ABE schemes. Although this performance variation is predictable, it should be quantified. Therefore, the purpose is to provide developers with practical information about the implementation of different schemes and libraries to help them in their selection. Developers should consider that in the case of BDABE, the experiments only measure the time required for the scheme to perform cryptographic operations. All the time related to Blockchain operations is not considered here since it is highly variable and out of the scope of this paper.

5.2. Testbed Setup

The testbed consists of two Raspberry Pi (RPI), i.e., a Raspberry Pi Zero (RPi0) and a Raspberry Pi 4 (RPi4). The RPi0 runs Raspbian Stretch, has 512MB of RAM, 1GHz, a single-core ARMv6, and Wi-Fi. We consider it a good representation of an IoT device. The RPi4 has an ARMv8 processor, 8GB of RAM, and runs 32-bit Ubuntu Server TLS. This second device is considered representative of IoT devices with high processing capabilities. This setup provides a representative insight into the execution time of the libraries on IoT devices, as well as libraries’ multi-architecture capabilities.

3https://docs.rs/rabe/0.3.1/rabe/index.html
As explained in the discussion of AES schemes in Section 4.2, ABE schemes are paired with symmetric ciphers. Therefore, in this section’s experiments, AES encrypts a 43-byte PT, and ABE schemes encrypt the 256-bit AES key. The reason behind the small-sized PT is that this paper evaluates the computational efficiency of different ABE implementations, so a small PT has been chosen to make the computational burden of ABE more significant than that of AES. This provides a representative view of the efficiency of each library. For the mentioned time measurements, experiments have been carried out by benchmarking, which depends on the library and its implementation language:

- OpenABE: The library itself offers its proprietary benchmark.
- Rabe: Criterion for Rust.
- Charm: Timeit for Python.
- GoFE: Golang benchmarking.

It is worth mentioning that Rabe runs on Rust nightly, so the native Rust benchmark can be used. However, this is an unstable feature that always returns the time value in nanoseconds. In complex schemes, the execution time is longer than the maximum value that can be returned by the u32 type variable used by the native benchmark. Therefore, an overflow is obtained which does not capture the actual duration of the function. Criterion, which is more stable and accurate, has been used instead.

6. Results

To properly discuss the results, we present Table 4, which summarizes the properties of each scheme. Most ABE schemes have a linear time growth in encryption and decryption. However, some offer unique features such as constant times for certain operations.

| Scheme Type | KeyGen | Enc. | Dec. |
|-------------|--------|------|------|
| 1. KP-ABE   | Linear | Linear | Linear |
| 2. CP-ABE   | Constant | Constant | Constant |
| 3. dCP-ABE  |                     |               |

Table 4: Scheme features

6.1. CP-ABE

For this experiment, we analyze the time evolution of the different operations (i.e., keygen, encryption, and decryption) for a different number of attributes. This way, we quantify the variance of the time requirements according to the attributes contained in the AP or SK.

Figure 3 shows the time in seconds required to generate secret keys for a different number of attributes. The required time grows linear with the number of attributes used for the SK generation. GoFE-FAME (G_FAME in the Figure) is the slowest implementation for key generation. It takes 3.36s in the RPI4 and 18s in the RPI0. Meanwhile, Rabe-FAME (R_FAME) takes 0.71s for the worst case in the RPI4 and 3s in the RPI0.

The difference in times shown in Figure 3 is related to GoFE using crypt-go, Golang’s cryptographic libraries, which are less efficient than consolidated libraries like OpenSSL. There are forks of Go that add additional security features, but experiments for this paper have been carried out with the official Go distribution. Meanwhile, Charm and OpenABE use well-established and consolidated math libraries like PBC, Mircl, or Relic. These libraries have had numerous releases and patches that have added functionality and increased efficiency. Since programming languages are designed for different purposes, their optimization level impacts performance. However, the impact of mathematical dependencies is also noticeable: Figure 3 shows that a library written in an interpreted language (i.e., Python) is faster than one written in a compiled language (e.g., Go or Rust). The growth rate of GoFE makes it challenging to visualize the rest of the results. Therefore, we provide Figure 4 for a better view.

6 https://github.com/cloudflare/go
Figure 4 shows that the fastest scheme for key generation is OpenABE-Waters11 (O\_W11), which takes 83ms for twenty attributes in the RPI4 and 355ms in the RPI0. The next best scheme is Charm-Waters11 (C\_W11), with a notable difference from O\_W11. In fact, O\_W11 is approximately 74% faster than CW11 in both RPI4 and RPI0.

Another interesting finding is that Charm-BSW07 (C\_BSW07) takes a similar time as C\_W11 in both devices. In the RPI4, C\_BSW07 takes 327ms for the worst case, barely 13ms longer than C\_W11. In the case of RPI0, C\_BSW07 takes 1.60s and C\_W11 1.43s. Thus, the difference between C\_BSW07 and C\_W11 is 4.375% in RPI4 and 11.22% in RPI0.

Once we establish G\_FAME as the slowest scheme, it is interesting to see which one is the second slowest scheme. In RPI4, the second slowest scheme is R\_FAME at 0.70988s. Meanwhile, in the RPI0, the second slowest scheme is C\_FAME at 3.69s. This is because RPI0 favors compiled languages like Rust and it is more affected by interpreted languages like Python.

Key generation is critical, and its running time should be measured. However, this operation is performed at a much lower frequency than encryption and decryption. To study the libraries’ encryption times, we present Figure 5. It shows that G\_FAME grows exponentially, taking 40 seconds for the worst case in the RPI4 and 3 minutes for the RPI0. Meanwhile, the rest of the library-scheme combinations take less than a second in the RPI4 and less than four seconds in RPI0. Furthermore, FAME encryption time grows linearly with the complexity of the AP, which G\_FAME fulfills in neither device. Because of the long time taken by GoF\_E to encrypt information, we present Figure 6 to be able to visualize the rest of the results.

We can see in Figure 6 that R\_FAME is the slowest encryption scheme for every case in the RPI4. At 15 attributes, it holds a notable difference from the next-slowest one, C\_FAME. For this case, R\_FAME takes 690.36ms and C\_FAME 400ms, making R\_FAME 72.5% slower. However, this changes for more than 15 attributes, and C\_W11 becomes the second slowest.
scheme. At 20 attributes, C\textsubscript{W11} takes 513.42ms and R\textsubscript{FAME} 878.26, making this last scheme 70% slower. Developers should keep the dependence on the AP complexity in mind when implementing CP-ABE schemes. The fastest scheme is O\textsubscript{W11}, which takes 250 ms for 20 attributes, making it 33% faster than the second fastest scheme (C\textsubscript{BSW07}) for the same case.

Figure 6\textsubscript{b} presents the results for the RPI0. It shows that C\textsubscript{FAME} and R\textsubscript{FAME} have a slight difference of 0.26% in the case of 20 attributes: 3.76s for C\textsubscript{FAME} and 3.77s for R\textsubscript{FAME}. Overall, the fastest scheme is O\textsubscript{W11}, which takes 1s for the case of 20 attributes. Due to being implemented in C++, it is highly efficient and more suitable for the RPI0 than the python version of the same scheme (i.e., C\textsubscript{W11}). Regarding the second fastest scheme, C\textsubscript{BSW07} takes 1.98s for the worst case. This makes the difference between C\textsubscript{BSW07} and O\textsubscript{W11} noteworthy since it makes O\textsubscript{W11} 49% faster.

Although encryption times grow linearly with the number of attributes contained in the CT, the pattern does not always hold for decryption. As was introduced in Table\textsubscript{4} some schemes (e.g., FAME) have the distinctive feature of constant decryption time, independent of the number of attributes in the CT. This is seen in Figure\textsubscript{7}, where G\textsubscript{FAME} decryption takes 800ms for every case in a RPI4 and approximately 4s in a RPI0. However, the Python implementation (C\textsubscript{FAME}) and Rust implementation (R\textsubscript{FAME}) take 160ms and 130ms, approximately 80%-84% less time than G\textsubscript{FAME}.

![Figure 7: CP-ABE decryption times in seconds.](image1.png)

Figure 7: CP-ABE decryption times in seconds.

One of the differences between the RPI and RPI4 is that in RPI0, C\textsubscript{FAME} and R\textsubscript{FAME} no longer have similar times. In fact, Rust’s efficiency over Python is apparent in these results, with R\textsubscript{FAME} taking 607.69ms and C\textsubscript{FAME} taking approximately 950ms, 56% more time for the same scheme.

Interestingly, G\textsubscript{FAME} is not always the slowest scheme. In the case of RPI4, the slowest scheme for more than 12 attributes is R\textsubscript{BSW07}, taking up to 1.34s to decrypt a CT of 20 attributes. In the case of RPI0, the same happens for more than 16 attributes, with R\textsubscript{BSW07} taking 4.47s for a CT of 20 attributes. Finally, the fastest option in both devices is C\textsubscript{BSW07}.

6.2. KP-ABE

This section conducts an analysis of libraries that implement KP-ABE schemes, analogous to that conducted for CP-ABE in Section 6.1. To this end, we present Figure\textsubscript{8}, which shows the time required to create users’ secret keys. KP-ABE associates user keys with APs. Thus, this experiment analyzes the time required to generate the keys as a function of the number of attributes contained in the policy.

![Figure 8: KP-ABE Key Generation times in seconds.](image2.png)

Figure 8 shows that the FAME implementation of Rabe, named (R\textsubscript{FAME}) in the figure, has an exponential time evolution for user key generation. In fact, for the case of 20 attributes, the time required goes up to 12s for RPI4 and up to 56s for RPI0. The difference with the second slowest scheme is big enough that we provide Figure\textsubscript{9} to visualize it.

Figure\textsubscript{9} shows that R\textsubscript{YCT14} is the fastest scheme-library combination. It takes 12ms to generate a key with 20 attributes for a RPI4 and 76.4ms for a RPI0. The difference with the second slowest scheme is big enough that we provide Figure\textsubscript{9} to visualize it.

Figure\textsubscript{9} shows that R\textsubscript{YCT14} is the fastest scheme-library combination. It takes 12ms to generate a key with 20 attributes for a RPI4 and 76.4ms for a RPI0. In RPI4, R\textsubscript{YCT14} shows a significant difference with the second fastest scheme, O\textsubscript{GPSW07}. For the same case of 20 attributes, O\textsubscript{GPSW07} requires 243s, a difference of 181%. Meanwhile, in the RPI0, O\textsubscript{GPSW07} takes 848ms, a difference of 166% from R\textsubscript{YCT14}.

![Figure 9: KP-ABE Key Generation times in seconds.](image3.png)
Regarding the second slowest scheme, C_LSW10, it is interesting to see that in the RPI4, it requires 572ms. This implies 71% more time than the same scheme in Rust (R_LSW10), 334ms. However, as Figure 9b presents, in RPI0, the difference between both schemes goes up to 185%: 4s for the worst case in C_LSW10 and 1.4s in the R_LSW10. The experiment, thus, clearly shows how devices with reduced computing capabilities favor libraries written in Rust and C++ over those implemented in Python.

When it comes to encryption, the results can be seen in Figure 10. It is clear from the figure that the fastest scheme for encryption is O_GPSW06, taking 84ms for 20 attributes in a RPI4 and 292ms in a RPI0. Despite its slow key generation times, C_LSW10 is a close second for the most efficient encryption scheme. In a RPI4 it takes 104ms for the worst case: barely 23% more than O_GPSW06. In a RPI0, however, C_LSW10 requires 646ms for the worst case: 121% more than O_GPSW06. In contrast, the slowest scheme in both devices is G_GPSW06.

Finally, decryption times for KP-ABE are depicted in Figure 11. It can be seen that KP-ABE FAME has constant decryption time, as with its CP-ABE version. However, the only library implementing it is Rabe (R_FAME).

As a result of the FAME’s constant decryption times, the fastest decryption scheme depends on the number of attributes of the CT. Therefore, O_GPSW06 is the fastest scheme for less than 11 attributes in RPI4 and less than 15 attributes in RPI0. RPI4 takes 136 ms to decrypt a CT with 10 attributes using O_GPSW06; and for the same scheme, but with 15 attributes, RPI0 takes 670 ms. Meanwhile, for more than 11 attributes, R_FAME becomes the fastest option in RPI4, taking 150ms; and for more than 15 in RPI0, taking 700ms. Meanwhile, the slowest scheme is G_GPSW06.
6.3. Decentralized CP-ABE

Decentralized CP-ABE has an added operation to those shown for KP-ABE or CP-ABE: authority setup. Although every scheme requires a setup, decentralized schemes require the initialization of each authority. Some decentralized schemes allow users to setup authorities at any point in the system’s lifetime. Thus, it is interesting to quantify how much time these setups can take. The times can be visualized in Figure 12.

![Figure 12: dCP-ABE Authority Setup times.](image)

As Figure 12 presents, authorities setup times vary widely between schemes. The figure shows the time it takes to configure five authorities. Specifically, it shows the time evolution depending on how many attributes each manages. C_RW15 is the fastest one in both devices, taking 30ms for the five authorities to be set up in a RPI4, with each of them controlling four attributes each. The RPI0 requires 132ms for the same operation. The closest scheme in execution performance to C_RW15 is C_LW11. However, an analysis of the results shows that C_LW11 takes 246.84ms in RPI4, presenting a 157% difference compared to C_RW15. The same ratio is maintained at RPI0, where C_LW11 takes 1.12s. Therefore, even the second fastest scheme is no match for C_RW15. It is noteworthy that RW15 is the evolution of LW11, and thus stands to reason that RW15 is more efficient in some of its operations. Meanwhile, G_LW11 is the slowest, maintaining GoFE as the slowest library for many schemes and operations.

After the authorities are setup, they can start generating users’ secret keys. For this experiment, we work with five authorities, each of which controls a variable amount of attributes, from 2 to 4. The result of key generation times can be visualized in Figure 13.

![Figure 13: dCP-ABE Key Generation times.](image)

As shown in Figure 13, the fastest dCP-ABE scheme for key generation in the RPI0 is R_LW11, which takes 557ms for the case of requesting 4 attributes from each AA. In contrast, on RPI4, the fastest scheme for 3 attributes or more is R_BDABE. For the case of asking 4 attributes to each AA, R_BDABE takes 33% less than R_LW11 (85.25ms in R_BDABE VS 127ms in R_LW11). In the case of 3 attributes or less, the fastest scheme is R_LW11. However, the advantage presented by R_BDABE is so marginal that when the processing capabilities are reduced (case of RPI0), this advantage disappears. Moreover, as introduced during the qualitative evaluation in Section 4, to the result of R_BDABE, we have to add the time needed to interact with the Blockchain, which can be long and unpredictable. Thus, even for RPI4, R_LW11 can be considered the fastest key generation scheme, taking 127ms for the worst case.

Finally, the slowest scheme is C_RW15 in both cases. However, the difference between RPI0 and RPI4 is noteworthy. While G_LW11 is the slowest scheme in RPI4 with a noticeable difference from C_RW15, this difference is much smaller in RPI0. Moreover, in the case of having to request 4 attributes, G_LW11 on RPI0 takes 3.32s while C_RW15 takes 3.53s. The little difference between the two schemes is related to the implementation of GoFE. In all the results, it has been observed how this library, when implemented in RPI0, achieves much worse results than in RPI4. However, key generation is an operation that takes place only when users require a key, and its impact is limited. Meanwhile, encryption and decryption take place more often. Encryption results are presented in Figure 14.

Figure 14 shows that the slower encryption scheme is G_LW11, taking up to 5s for a policy of 20 attributes (distributed homo-
geneously among five authorities) in a RPI4. For the same operation, the RPI0 requires 37.5s. In fact, LW11 is generally one of the slowest schemes since it is also the slowest dCP-ABE scheme forRabe (R_LW11), which takes 1.3 seconds for the worst case in the RPI4 and 8s for a RPI0. To better see the results of the rest of the schemes, we provide Figure 15.

Figure 15 allows us to better see the encryption results of the fastest schemes. It shows that the fastest one is R_MKE08, taking 131ms for encryption in RPI4. This same operation in the RPI0 takes 439ms. Although R_BDABE is almost as fast as R_MKE08, one should consider the time delays related to the Blockchain operations.

Finally, after encrypting information, we depict decryption time results in Figure 16. We see that results are somewhat analogous to those of encryption. Decryption is faster than encryption, but the schemes’ performance is equivalent. Once again, the results show that G_LW11 is the slowest scheme.

Finally, R_BDABE and R_MKW08 are the fastest schemes for decryption. However, as previously stated, BDABE will have an added delay related to the Blockchain when implemented. Thus, the fastest scheme is R_MKE08, which takes 105ms on average for decryption in a RPI4 and 510ms for RPI0.

7. Discussion

As a result of our experimental evaluations, we provide Table 5 which summarizes the most efficient schemes for each case. The performance of BDABE relies on the type of Blockchain used, and developers must be aware of this when evaluating the results in this section.

Table 5 presents the results according to the type of ABE (CP-ABE, KP-ABE, and dCP-ABE) and the implementation language. Sometimes developers may require a specific ABE type but without a specific scheme. Table 5 shows which scheme from which library is the most efficient. Similarly, we also
present results tied to a specific library, as developers may be constrained to specific languages.

Regarding the scheme type, for developers working with CP-ABE, the fastest scheme for key generation is O_W11. One of the main advantages of this scheme is that it is also the fastest for encryption. However, developers should consider that CP-ABE schemes usually have slower decryption time than encryption, and O_W11 is no exception. In fact, if developers need a fast decryption CP-ABE scheme, they should choose C_BSW07. This scheme has a good balance between the three operations, since it is the second fastest for key generation and encryption in both devices.

When dealing with KP-ABE, the fastest scheme for key generation is R_YCT14. Key generation only takes place when system users request them, but fast generation is useful in systems where new users are continually added. Meanwhile, the fastest scheme for encryption is O_GPSW06. One of the advantages of this scheme is that it is also the second fastest for key generation. In addition, O_GPSW06 is also the fastest scheme for APs with less than 10 attributes in RPI4 and less than 15 attributes in RPI0. However, if the system has APs with more attributes, R_FAME becomes a better choice. R_FAME has constant decryption times, so its decryption time is independent of the complexity of the APs.

Regarding dCP-ABE, developers now have to consider the time required to set up authorities. Some schemes only allow developers to do this during the system setup, but others allow AAS to be set up throughout the system’s lifetime. In this regard, the fastest scheme is C_RW15. However, it is also one of the slowest schemes for key generation, encryption, and decryption. For faster key generation, the best option is R_RW11. However, this scheme is also the second slowest for encryption and the slowest for decryption. Thus, the fast key generation is hardly compensated by the rest of the delays since encryption and decryption are more common tasks. However, as mentioned earlier, systems with many new users may benefit from this. Finally, for fast encryption and decryption, the best option is R_MKE08. It is noteworthy that this scheme is also the second fastest one for key generation.

Table 5 also summarizes the fastest schemes for each library. Not all libraries implement the same schemes, so the options for some may be reduced. For example, OpenABE only provides one CP-ABE and one KP-ABE scheme. However, we can see that W11 is the fastest scheme for setupkey generation in both OpenABE and Charm, and GPSW06 is the fastest for encryption in GoFE and OpenABE. In the case of texitRabe, we can see how MKE08 is one of the most balanced schemes: it is the fastest scheme provided by texitRabe for authority generation, encryption, and decryption.

Finally, other results that might be useful for developers and are not contemplated in Table 5 may be:

- BDABE is the only Blockchain-based scheme, and texitRabe is the only library implementing it.

- OpenABE was not originally designed for ARM architectures. However, it can be adapted to ARMv7 and ARMv8 architectures\(^7\) and with further modifications\(^8\) to ARMv6.

- Some users may require schemes with attribute revocation. This requirement is out of the scope of the survey presented in this paper. However, the schemes that claim to provide this feature are LSW10, TimePRE, DAC-MACS, YJ14, and BDABE.

- YCT14, YJ14, and DAC-MACS are broken, and therefore should be avoided.

8. Conclusions

This paper provides a qualitative and quantitative comparison of 11 ABE libraries. We first highlight those implementing

---

\(^7\)https://github.com/IBM/openabe
\(^8\)https://github.com/relic-toolkit/relic/issues/211
vulnerable ABE schemes and identify the mathematical library underneath. This, among other criteria, has discouraged using several schemes.

We also provide a qualitative evaluation of each scheme provided by the libraries and their distinctive features. Since all schemes are implemented in hybrid mode, we also identify and analyze the AES mode used by each library.

We select the four best candidates and quantitatively assess their efficiency based on the qualitative evaluation. We measure efficiency in terms of how much time each scheme takes to perform basic operations like key generation, encryption, and decryption.

As a result of our experimental evaluations, we provide a table in which we indicate which schemes are faster based on their approach, programming language, and the device running them.

Developers may require efficiency at different points in the process: key generation, authority setup, encryption, or decryption. There is no one-size-fits-all scheme, because no one scheme can be efficient in all aspects of the process. Thus, developers should analyze which step will require lower computational time. Our discussion provides developers with the tools to support them in selecting the most suitable library and scheme.

Acknowledgment

The European commission financially supported this work through Horizon Europe program under the IDUNN project (grant agreement N° 101021911). It was also partially supported by the Ayudas Cervera para Centros Tecnológicos grant of the Spanish Centre for the Development of Industrial Technology (CDTI) under the project EGIDA (CER-20191012), and by the Basque Country Government under the ELKARTEK program, project REMEDY - Real tME control and embedded security (KK-2021/00091).

References

[1] A. Mosteiro-Sanchez, M. Barcelo, J. Astorga, A. Uribeta, All Cryptoli- ibreries Are Beautiful, But Some Are More Beautiful Than Others: A Survey of CP-ABE Libraries, in: URSI 2022, Málaga, Spain, 2022, pp. 1–4.
[2] A. Sahai, B. Waters, Fuzzy Identity-Based Encryption, in: R. Cramer (Ed.), Adv. Cryptol. – EUROCRYPT 2005, Springer Berlin Heidelberg, Aarhus, Denmark, 2005, pp. 457–473. doi:10.1007/11426639_27
[3] A. Shamir, Identity-Based Cryptosystems and Signature Schemes, in: G. R. Blakley, D. Chaum (Eds.), Advances in Cryptology, Springer Berlin Heidelberg, Berlin, Heidelberg, 1985, pp. 47–53.
[4] V. Goyal, O. Pandey, A. Sahai, B. Waters, Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS ’06, Association for Computing Machinery, New York, NY, USA, 2006, p. 89–98. doi:10.1145/1180405.1180418
[5] J. Bethencourt, A. Sahai, B. Waters, Cipher-text-policy attribute-based encryption, in: 2007 IEEE Symposium on Security and Privacy (SP ’07), 2007, pp. 321–334. doi:10.1109/SP.2007.11
[6] S. Müller, S. Katzenbeisser, C. Eckert, Distributed attribute-based encryption, in: P. J. Lee, J. H. Cheon (Eds.), Information Security and Cryptol- ogy – ICISC 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 20–36. doi:10.1007/978-3-642-00730-9_2
[7] J. Bethencourt, A. Sahai, B. Waters, cpabe-toolkit (2011). URL: https://accc.cs.utexas.edu/cpabe/
[8] M. D. Green, J. A. Akinyele, M. A. Rushanan, libfenc (2011). URL: https://code.google.com/archive/p/libfenc/
[9] J. Security, C. Lab, Charm (2021). URL: https://github.com/JRILISI/charm
[10] B. Waters, M. Green, S. H. Waters, J. A. Akinyele, A. M. Dunn, M. Rushanan, OpenABE (2011). URL: https://github.com/zeuto/OpenABE
[11] G. Bramm., G. Gall., J. Schütte., Bdabe - blockchain-based distributed attribute based encryption, in: Proceedings of the 15th International Joint Conference on e-Business and Telecommunications - SEC- RYTP. INSTICC, SciTePress, 2018, pp. 99–110. doi:10.5220/0006852602650276
[12] J. Schütte., G. Bramm., D. Bücheler, Rabe (2021). URL: https://github.com/praunhofer-AISEC/Rabe
[13] M. Bizjak, J. Hartman, T. Marc, M. Stopar, CoPe (2021). URL: https://github.com/fentec-project/cope
[14] M. Bizjak, J. Hartman, T. Marc, M. Stopar, CiFEr (2021). URL: https://github.com/fentec-project/CiFEr
[15] A. de la Piedra, M. Venema, G. Alipr, ABE Squared: Accurately Benchmarking Efficiency of Attribute-Based Encryption Cryptology ePrint Arch., Report 2022/038 (2022). URL: https://ia.cr/2022/038
[16] M. Venema, G. Alipr, A Bunch of Broken Schemes: A Simple yet Pow- erful Linear Approach to Analyzing Security of Attribute-Based Encryp- tion, in: CT-RSA 2021, Springer, Virtual Event, 2021, pp. 100–125. doi:10.1007/978-3-030-75539-3_5
[17] C. Chow, JBPC-FAME (2020). URL: https://github.com/chrizchow/JBPC-FAME
[18] J. Wang, J. Perrochet, M. Grossi, S. Weiland, cp-abe (2018). URL: https://github.com/junwei-vang/cpabe/
[19] M. Morales-Sandoval, A. Díaz-Perez, Det-abe: a java api for data confidentiality and fine-grained access control from attribute based encryption, in: WISTP, Springer, 2015, pp. 104–119. doi:10.1007/978-3-319-24018-3_7
[20] TU-Berlin-SNET, JCPABE (2016). URL: https://github.com/TU-Berlin-SNET/JCPABE
[21] A. Lewko, A. Sahai, B. Waters, Revocation systems with very small priv- ies, in: 10th IEEE Symposium on Security and Privacy, 2010, pp. 273–285. doi:10.1109/SP.2010.23
[22] X. Yao, Z. Chen, Y. Tian, A lightweight attribute-based encryption scheme for the internet of things, Future Generation Computer Systems 49 (2015) 104–112. doi:10.1016/j.future.2014.10.010
[23] S. Agrawal, M. Chase, Fame: Fast attribute-based message encryption, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS ’17, Association for Computing Machinery, 2017, p. 665–682. doi:10.1145/3133956.3135014
[24] B. Waters, Cipher-text-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization, in: D. Catalano, N. Fazio, R. Gennaro, A. Nicolosi (Eds.), 14th Int. Conf. Pract. Theory Public Key Cryptogr., Springer Berlin Heidelberg, Taormina, Italy, 2011, pp. 53–70. doi:10.1007/978-3-642-19379-8_4
[25] S. Yamada, N. Attrapadung, G. Hanaoka, A framework and compact constructions for non-monotonic attribute-based encryption, in: H. Krawczyk (Ed.), Public-Key Cryptography – PKC 2014, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 275–292. doi:10.1007/978-3-642-54631-0_16
[26] J. Chen, R. Gay, H. Wee, Improved dual system abe in prime-order groups and compact constructions for non-monotonic attribute-based encryption, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS ’17, Association for Computing Machinery, New York, NY, USA, 2018, p. 89–98. doi:10.1145/3133956.3135014
[27] Q. Liu, G. Wang, J. Wu, Time-based proxy re-encryption scheme for secure data sharing in a cloud environment Information Sciences 258 (2014) 355–370. doi:https://doi.org/10.1016/j.ins.2012.09.034
[28] A. Lewko, B. Waters, Decentralizing Attribute-Based Encryption, in: K. G. Paterson (Ed.), Adv. Cryptol. – EUROCRYPT 2011, Springer
[29] K. Yang, X. Jia, DAC-MACS: Effective Data Access Control for Multi-Authority Cloud Storage Systems, Springer New York, New York, NY, 2014, pp. 59–83. \textit{doi:10.1007/978-1-4614-7873-7_4}

[30] K. Yang, X. Jia. Expressive, efficient, and revocable data access control for multi-authority cloud storage, IEEE Transactions on Parallel and Distributed Systems 25 (7) (2014) 1735–1744. \textit{doi:10.1109/TPDS.2013.253}

[31] Y. Rouselakis, B. Waters, Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption, in: R. Böhme, T. Okamoto (Eds.), Financ. Cryptogr. Data Secur., Springer Berlin Heidelberg, San Juan, Puerto Rico, 2015, pp. 315–332. \textit{doi:10.1007/978-3-662-47854-7_19}

[32] S.-Y. Tan, K.-W. Yeow, S. O. Hwang, Enhancement of a lightweight attribute-based encryption scheme for the internet of things, IEEE Internet of Things Journal 6 (4) (2019) 6384–6395. \textit{doi:10.1109/JIOT.2019.2906631}