Multi-Locking and Perfect Argument Order: Two Major Improvements of Attribute-Based Encryption

Cyrius Nugier, Remi Adelin, Vincent Migliore, Eric Alata

October 17, 2019

Abstract. Attribute Based Encryption, proposed by Sahai and Waters in 2007, is a set of promising cryptographic schemes that enable various fine grained access control on encrypted data. With a unique encryption key, a user is able to encrypt data for a very specific group of recipient that matches a set of attributes contained inside their decryption key. In current scenario where personal devices share an increasing volume of private data on the web, such encryption algorithms are more than ever a strong alternative to standard encryption algorithms.

In this paper, we propose two major improvements of ABE namely the Perfect Argument Order Optimization and the Multi-Locking. Multi-Locking ABE is an extension of ABE that enables to share access control policy on an arbitrary number of entities. We also make a step further for the speed-up of ABE by providing the "Perfect Argument Order Optimization", which is a generalization of the "Fixed Argument Optimization" of Scott et al. to a much wider range of ABE constructions (and in particular to our Multi-Locking ABE). Based on those two improvements we propose a construction of the first privacy-preserving Cloud service based on ABE, allowing ephemeral accesses to the data. The Multi-Locking ABE and the Perfect Argument Order Optimization have been successfully integrated to the OpenABE library, providing a speed-up for a variety of ABE constructions.

Keywords: Attribute-Based Encryption · Optimization · Privacy

1 Introduction

Usage of computers has recently evolved with a great increase of connectivity features. From social networks to connected objects such as Internet of Things (IoT), a massive amount of private data is daily exchanged through internet. Consequently, people should be more and more careful about their privacy.

Traditional encryption algorithms are usually well suited for one-to-one communications. To satisfy recent needs, they have been adapted to communications with one emitter and a fixed and well defined group of recipients, with limited possibilities to manage a fine-grained access control policy. For example, symmetric and asymmetric encryptions such as AES or RSA are based on a "all or nothing" strategy, which requires as much key pairs as the number of targeted groups of recipients.

Thankfully, Public Key Encryption was subjected to successive evolutions to provide more flexibility. The first evolution was the Identity-Based Encryption [Sha85], where the encryption key is function of publicly known information about the recipient and can be computed on-the-fly. Management of keys was greatly improved, however these constructions did not allow groups of recipients. A second construction presented in [SA05] proposed an encryption for a group of people based on a "fuzzy" identity. More precisely, people with identity "close" to the identity selected during encryption were able to decrypt. Then in 2006, Han et al. proposed in [GPSW06a] the first encryption algorithm that allows a fine-grained access control on encrypted data, called Attribute-Based Encryption (ABE).

ABE is based on a combination of attributes and access structures that can respectively be seen as arguments and functions. These function evaluate to 1 if decryption is possible and to 0 otherwise. When attributes are integrated to the encrypted data and access structure is attached to recipients, we construct a Key-Policy ABE scheme. Inversely, if access structure is integrated to encrypted data and attributes are attached to recipients, we construct a Ciphertext-Policy ABE scheme. In any case, decryption restitutes the correct message only when attributes match the access structure.

Expressive and efficient realizations exist for both these types of schemes (knowingly [Wat11] for CP-ABE and [GPSW06a] for KP-ABE). Most common ABE schemes are divided into four algorithms:

Setup Executed by a trusted authority. Takes as input the security parameter and the universe of attributes and outputs the public key and master key.

KeyGen Executed by a trusted authority. Given the master key and a set of attributes (CP-ABE) / access tree (KP-ABE), generates the corresponding secret key. In order to be secure against collusions, each key is randomized using a secret element, even with the same set of attributes.

Encrypt Takes as input a message, the public key and the wanted access tree (CP-ABE) / attributes (KP-ABE) and outputs the ciphertext. In order to be secure against replay, a randomly generated element makes each ciphertext of the same message different.

Decrypt Takes a ciphertext and a secret key, and provides the right plain message if and only if the attributes validates the access tree.

A wide range of variations are constructed from this basis, providing compromise between computation times of Setup, Encrypt, KeyGen, Decrypt, and the length of Master Key, Public Key, Secret Keys and Ciphertexts. Additionally, in order to solve specific situations, each scheme exhibits some interesting properties like being multi-authority, decentralized, hierarchical, having non-monotonic access structures, allowing user revocation, attribute revocation, having a hidden policy, collusion resistance, being quantum computer resistant...

At this point, when opting for an ABE scheme for a cryptosystem, one would check out the most recent survey of ABE schemes (such as [PSA18], where the previous properties are defined), and find the construction that matches the most with desired properties. This “top-down” approach can sometimes leaves some specifications unsatisfied.

A “bottom-up” approach would be, a priori, much more desirable, taking as inputs specifications to generate a freshly designed ABE scheme. However, a such process may be particularly complex to design and implement. Moreover, it is not practical since devices capabilities, infrastructures and scenario constraints continuously evolve.

OUR CONTRIBUTIONS: To address this issue, we propose in this paper the “Multi-Locking” ABE construction, a framework that allows efficient composition of several ABE schemes during transfer of data, making construction of advanced infrastructures particularly simple. Additionally, the framework allows designers to implement and combine the following features:

1. the possibility to impose a specific route for the data.

2. the possibility to delegate access control to trusted nodes.

To demonstrate of the framework capacities, we combined both features to construct the first privacy-preserving Cloud service with ephemeral data access, presented in section 6.

Since now nearly all ABE schemes can be integrated inside the framework, the second major contribution of the paper is an extension of the Fixed Argument Optimization presented in [Sco11], leading to a straight 30% boost of decryption pairing calculation time to nearly 100% of existing ABE schemes. We call this optimization "Perfect Argument Order Optimization".

Both Multi-Locking and Perfect Argument Order Optimization have been successfully integrated into the OpenABE library.

The remainder of the paper is organized as follows:

Section 2 provides fundamental background of ABE;

Section 3 (*Contribution*) presents the Perfect Argument Order Optimization;

Section 4 (*Contribution*) explains of Multi-Locking framework construction;

Section 5 (*Contribution*) reviews existing ABE schemes and proposes some recommendations to make them compatible with Perfect Ordering and Multi-Locking;

Section 6 (*Contribution*) defines the construction of the ephemeral Cloud;

Section 7 (*Contribution*) details the integration of the Perfect Argument Order Optimization and Multi-Locking framework into OpenABE and RELIC libraries and benchmark results;

Section 8 draws some conclusions and perspectives.

2 Background and definitions

This Section provides usual definitions and results of standard ABE schemes. We will also make a focus on the work [ALdP11] since it will be reused to construct our privacy-preserving Cloud service with ephemeral data access (see Section 6 for details).

2.1 Functional Encryption: Syntax and Security Definition

2.1.1 Syntax

Let $R : \Sigma k \times \Sigma e \to \{0, 1\}$ be a boolean function where Σk and Σe denote "keyindex" and "ciphertextindex" spaces. A functional encryption (FE) scheme for the relation R consists of algorithms: $Setup, KeyGen, Encrypt, Decrypt$.

Setup$(\lambda, des) \to (mpk, msk)$**:** The setup algorithm takes as input a security parameter λ and a scheme description des and outputs a master public key mpk and a master secret key msk.

KeyGen$(msk, X) \to sk_X$**:** The key generation algorithm takes in the master secret key msk and a key index $X \in \Sigma k$. It outputs a private key sk_X.

Encrypt$(mpk, M, Y) \to C$**:** This algorithm takes as input a public key mpk, the message M, and a ciphertext index $Y \in \Sigma e$. It outputs a ciphertext C.

Decrypt$(mpk, sk_X, X, C, Y) \to M$ ***or*** $\perp$**:** The decryption algorithm takes in the public parameters mpk, a private key sk_X for the key index X and a ciphertext C for the ciphertext index Y. It outputs the message M or a symbol $\perp$ indicating that the ciphertext is not in a valid form.

Correctness mandates that, $\forall\lambda$, $\forall(mpk, msk)$ produced by $Setup(\lambda, des)$, $\forall X \in \Sigma k$, all keys sk_X returned by $KeyGen(msk, X)$ and all Y$\in \Sigma e$,

- If R(X, Y) = 1, then $Decrypt(mpk, Encrypt(mpk, M, Y), sk_X) = M$.
- If R(X, Y) = 0, then $Decrypt(mpk, Encrypt(mpk, M, Y), sk_X) = \perp$.

2.1.2 Security notions

We now give the standard security definition for FE schemes.

Definition 1. FE scheme for relation R is fully secure if no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ has non-negligible advantage in this game:

Setup. The challenger runs $(mpk, msk) \leftarrow$ Setup(λ,des) and gives mpk to $\mathcal{A}$.

Phase 1. On polynomially-many occasions, $\mathcal{A}$ chooses a key index X and gets: $sk_X = KeyGen(msk, X)$. Such queries can be adaptive in that each one may depend on the information gathered so far.

Challenge. $\mathcal{A}$ chooses messages M_0, M_1 and a ciphertext index Y^* such that $R(X, Y^*) = 0$ for all key indexes X that have been queried at Step 2. Then, the challenger flips a fair binary coin $d \in \{0, 1\}$, generates a ciphertext $C^* = Encrypt(mpk, M_d, Y^*)$, and hands it to the adversary.

Phase 2. $\mathcal{A}$ is allowed to make more key generation queries for any key index X such that $R(X, Y^*) = 0$.

Guess. $\mathcal{A}$ outputs a bit $d' \in \{0, 1\}$ and wins if $d' = d$. The advantage of the adversary $\mathcal{A}$ is measured by $Adv(\lambda) := |Pr[d' = d] - \frac{1}{2}|$.

2.2 Key-Policy Attribute-Based Encryption

In a Key-Policy Attribute-Based Encryption scheme, ciphertexts are associated with a set of attributes S and private keys correspond to access structures $\mathbb{A}$.

Definition 2 (Access Structures)**.** Consider a set of parties P such as $P = \{P_1, P_2, \cdots, P_n\}$. A collection $\mathbb{A} \subset 2^P$ is said to be monotone if, for all B, C, if $B \in \mathbb{A}$ and $B \subset C$, then $C \in \mathbb{A}$. An access structure (resp., monotonic access structure) is a collection (resp., monotone collection) $\mathbb{A} \subset 2^P \backslash \{\emptyset\}$ .The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.

When motonic access structures are represented as access trees, they contain only $\wedge$ and $\vee$, while non-monotonic ones can also contain $\neg$

Definition 3 (Linear Secret Sharing Scheme)**.** Let P be a set of parties. Let L be a $l \times k$ matrix. Let $\pi : \{1, \cdots, l\} \rightarrow P$ be a function that maps a row to a party for labeling. A secret sharing scheme Π for access structure $\mathbb{A}$ over a set of parties P is a linear secret-sharing scheme (LSSS) in $\mathbb{Z}_p$ and is represented by (L, π) if it consists of two efficient algorithms:

$Share(L, \pi)$: takes as input $s \in \mathbb{Z}_p$ which is to be shared. It chooses $\beta_2, \cdots, \beta_k \in_r \mathbb{Z}_p$ and let $\beta = (s, \beta_2, ..., \beta_k)$. It outputs $L \cdot \beta$ as the vector of l shares. The share $\lambda_i := \langle L_i, \beta\rangle$ belongs to party $\pi(i)$, where L_i is the i^{th} row of L.

$Recon(L, \pi)$: takes as input an access set $S \in \mathbb{A}$. Let $I = \{i|\pi(i) \in S\}$. It outputs a set of constants $\{(i, \omega_i)\}, i \in I$ such that $\sum_{i\in I} \omega_i \cdot \lambda_i = s$.

A clear example of construction algorithm for LSSS matrices is explained and detailed in [LCW10]. Decryption is possible when the attribute set S is authorized in the access structure $\mathbb{A}$ (i.e., $S \in \mathbb{A}$).

We formally define it as an instance of FE as follows:

Definition 4 (KP-ABE)**.** Let U be an attribute space. Let n $\in \mathbb{N}$ be a bound on the number of attributes per ciphertext. A Key-Policy Attribute-Based Encryption (KP-ABE) for a collection $\mathcal{AS}$ of access structures over U is a functional encryption for $R^{KP} : \mathcal{AS} \times \binom{U}{<n} \to \{0,1\}$ defined by $R^{KP}(\mathbb{A}, S) = 1$ iff $S \in \mathbb{A}$ (for $S \subseteq U$such that $|S| < n$, and $\mathbb{A} \in \mathcal{AS}$). Furthermore, the description *des* consists of the attribute universe U, $\Sigma_k^{KP} = AS, and\ \Sigma_e^{KP} = \binom{U}{<n}$.

2.3 Constant-Sized Ciphertext KP-ABE

In [ALdP11], authors provide the first method to turn a linear IBBE scheme with specifically constant-sized ciphertexts to an ABE scheme with the same constant-size property. However, authors do not explicit any formal instantiation of the scheme, so we propose the following construction:

We note $e(\cdot,\cdot)$ a pairing $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ whose properties are reminded in section 3.1.

Let n be the maximal amount of attributes that can be used to encrypt a message.

Setup$(\boldsymbol{\lambda}, \boldsymbol{U}) \to (\boldsymbol{MK}, \boldsymbol{PK})$: It chooses bilinear groups $\mathbb{G}$, $\mathbb{G}_T$ of prime order p $> 2^\lambda$, $g \in \mathbb{G}$

Then it randomly chooses α and $X = (x_0, \cdots, x_n) \in \mathbb{Z}_p^n$.

It sets H $= g^X$. and outputs the master key $MK = X$ and the public key $PK = (g, e(g,g)^\alpha, H)$.

One $\mathbb{Z}_p$ value *att* is publicly linked to each attribute in U.

KeyGen$(\boldsymbol{MK}, (\boldsymbol{L}, \boldsymbol{\pi})) \to \boldsymbol{SK_{id}}$: Compute $Share(L, \pi)$ of 1 where L is the given LSSS matrix of l columns and m rows. This is done by choosing $\beta_i\ \forall i \in [2:l]$ and defining column vector $\beta = (1, \beta_2 \cdots \beta_l)$. Then we define λ_i as $\langle L, \beta \rangle$ for all $i \in [1:m]$.

Then, $\forall i \in [1:m]$, choose $r_i \in \mathbb{Z}_p$ and define $SK_i = (D_{1i}, D_{2i}, D_{3i})$, where:

$$\begin{cases} D_{1i} = g^{\alpha\lambda_i} * H_0^{\lambda_i + r_i}, \\ D_{2i} = g^{\lambda_i + r_i}, \\ D_{3i} \text{ is the list } (D_{3ij}) \forall j \in [2:n] \text{ where } D_{3ij} = (H_1^{-att^{j-1}} * H_j)^{\lambda_i + r_i}. \end{cases}$$

Finally we output the list of all SK_i as the secret key SK.

Encrypt$(\boldsymbol{PK}, \boldsymbol{M}, \boldsymbol{S}) \to \boldsymbol{C}$: S is the receiver set for the message M. We define $Y = (y_1, \cdots, y_n)$ the coefficients vector of

$$Ps[Z] = \sum_{i=1}^{n} y_i * Z^{i-1} = \prod_{att_i \in S} (Z - att_i).$$

We then pick s in $\mathbb{Z}_p$ and compute C the ciphertext as follows:
$C = (C_0, C_1, C_2) = (M.e(g,g)^{\alpha s}, g^s, (H_0 \prod_{i=1}^n H_i^{y_i})^s)$.

Decrypt$(\boldsymbol{MK}, \boldsymbol{SK}, (\boldsymbol{L}, \boldsymbol{\pi}), \boldsymbol{C}, \boldsymbol{S}) \to \boldsymbol{M}$ or $\perp$: First, output $\perp$ if S is an un-authorized set for L.

Otherwise, take an authorized set w, let $I = \{i|\pi(i) \in w\}$ and calculate the recomposition constants w_i using $recon((L, \pi), w)$. Parse C as (C_0, C_1, C_2) and SK as $\{D_{1i}, D_{2i}, D_{3i}\}$ for all $i \in [1 : m]$ and keep the ones where i is in I.

To retrieve the plaintext message, compute

$$M' = Co \times \prod_i \Big(\frac{e(C_2, D_{2i})}{e(C_1, D_{1i} \prod_j D_{3ij}^{y_i})}\Big)^{w_i}$$

Correctness:

$$M' = Co \times \prod_i \Big(\frac{e(g^{s(x_0+x_1y_1+\cdots+x_ny_n)}, g^{\lambda_i+r_i})}{e(g^s, g^{\alpha\lambda_i+x_0(\lambda_i+r_i)} \cdot g^{(\lambda_i+r_i)\cdot\sum y_j(x_1*att^{j-1}+x_j)})}\Big)^{w_i}$$

$$= Co \times \prod_i \Big(\frac{e(g,g)^{s(\lambda_i+r_i)(x_0+x_1y_1+\cdots+x_ny_n)}}{e(g,g)^{s\alpha\lambda_i+s(\lambda_i+r_i)(x_0+\sum y_jx_1*att^{j-1}+x_jy_j)}}\Big)^{w_i}$$

$$= Co \times \prod_i \Big(\frac{e(g,g)^{s(\lambda_i+r_i)(x_0+x_1y_1+\cdots+x_ny_n)}}{e(g,g)^{s\alpha\lambda_i+s(\lambda_i+r_i)(x_0+x_1y_1+\cdots+x_ny_n)}}\Big)^{w_i}$$

$$= M * e(g,g)^{s\alpha} \prod_i e(g,g)^{-s\alpha*\lambda_i*w_i}$$

$$= M \text{ , since } \sum_i \lambda_i * w_i = 1.$$

Some remarks about this scheme: First, we easily verify constant-size property of the ciphertext since it is constructed with one element of $\mathbb{G}_T$ and two elements of $\mathbb{G}$. Second, this property is counterbalanced by the increase of SK length, which will be growing in magnitude $\mathcal{O}(n \times m)$. This must be taken into account when implementing this scheme.

Important note: The message confidentiality comes from its multiplication by $e(g,g)^{\alpha.s}$. The idea behind Attribute-Based Encryption is to send fragments of s that allow only a specific group of users to reconstruct s from their secret key. This the fundamental consideration that will be exploited in section 4 to provide Multi-Locking.

3 Perfect Argument Order for faster decryption

This Section presents the Perfect Argument Order Optimization that extends the idea of using precomputation to speed-up decryption computation time. Starting from the Fixed Argument Optimization presented in [CS10] (optimization that improves decryption but for a limited number of pairings), we propose a method called Switch Argument Method that allows to extend it to all pairings in existing ABE constructions. We also validated the optimization by a practical implementation, where we observed a 30% speed-up in pairing calculation during decryption (see Section 7 for details).

3.1 Asymmetric pairings and Perfectly Ordered schemes

The curve selection and the pairing computation have a considerable impact on the computation times and key length. Consequently, they drive design considerations of any cryptographic system based on pairings.

Fortunately, a well detailed mathematical analysis has been made in [Lyn07] chapter 4, providing a strong basis to construct efficient primitives. This work has been a solid foundation of efficient ABE scheme construction in [Sco11], which also exhibits some guidelines that should be followed to construct efficient KP-ABE scheme.

Among them, the Fixed Argument Optimization is one of the most impacting ones. Theoretically, the optimization should lead to 30% speed-up in pairing calculation during decryption as found in [CS10]. Unfortunately, this optimization is not applicable to all pairings of a scheme without a special attention to design.

3.1.1 Asymmetric pairings and selection of the optimal curve

We briefly provide mathematical background of asymmetric pairings and the method to select optimal curve.

Definition 5 (Pairing)**.** Let $\mathbb{G}_1, \mathbb{G}_2$ be two additive cyclic groups of prime order q, and $\mathbb{G}_T$ another cyclic group of order q. A pairing is a map: $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that have the following properties:

- Bilinearity: $\forall a, b \in \mathbb{F}_q^*,\ \forall P \in \mathbb{G}_1,\ \forall Q \in \mathbb{G}_2 : e(P^a, Q^b) = e(P, Q)^{ab}$.
- Non-degeneration: e $\neq$ 1.

Remark: for practical use of pairings in cryptography, it is fundamental that they should be computed efficiently and difficult to invert for security.

Definition 6 (Pairing Types)**.** We call:

- Symmetric and Type I a pairing where $\mathbb{G}_1 = \mathbb{G}_2$.
- Asymmetric and Type II a pairing where $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$.
- Asymmetric and Type III a pairing where $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$.

From definition 6, we can already note that from a cryptographic standpoint, Type II asymmetric pairings are mostly equivalent to symmetric pairings due to the existence of the homomorphism. Thus, in the following, we will call an asymmetric pairing a Type III pairing only.

When selecting the optimal pairing, the decisive parameter is the embedding degree k, which is the degree of the extension $\mathbb{K}$ of input group $\mathbb{G}_1$. The length of such elements follows the formula: $k \times \log_2(|\mathbb{G}_1|) = \log_2(|\mathbb{K}|)$.

In general, smaller embedding degrees allows for smaller output sizes and more efficient pairing computation. Unfortunately, attacks exist for small embedding degrees, referenced in [AS15].

Considering complexities of best actual attacks, we know that in order to achieve at least a 80-bit level of security, one should ensure that $\log_2(|\mathbb{G}_1|) \geq 160$ and $\log_2(|\mathbb{K}|) \geq 1024$.

For our implementation, we will consider a 128-bit security. With this setup, the best case scenario is achieved for a 256-bit input group embedded in a field of the same size (resist against Pollard's Rho attack) and an embedding degree of 12, leading to the output group being embedded into a 3072-bit field (safe against index calculus attack).

In practice, Type I pairings are implemented using super-singular elliptic curves and are limited to embedding degrees of 2, 4 and 6 respectively [Lyn07]. For asymmetrical pairings, there is no limitation on the embedded degree, thus it can be soundly selected in order to have the lowest secure input and output field sizes, maximizing efficiency.

We extracted the best curves from the work in [Lyn07], targeting an asymmetric pairing with the shortest possible ciphertexts and fastest computation times while keeping 128-bit security.

3.1.2 Selection of the pairing algorithm and Fixed Argument Optimization

Similarly to curve selection, pairing algorithm selection is also well documented. For Type I pairings, the Tate pairing is the most common approach, being computed through Miller's Algorithm [Mil86]. For better efficiency on small characteristic fields, one should definitely look at the modified Tate Pairing η_T as described for example in [BBD$^+$08].

For Type III pairings, some variants like the ate paring or the R-ate pairing allow loop reductions. But the main benefit of using Type III pairings lies in the possibilities of pre-computation. The first improvement was proposed by Scott in work [SCA06], performing pre-computations on one argument of the pairings and its associated slopes. Enhancement were proposed afterwards by Costello and Stebila in [CS10] that defined "Multi-pairings" and the so called "Fixed Argument Optimization" presented by Scott in [Sco11]. Important note: all other things held constant, we will always call the precomputable argument the *left-side argument* and the other one the *right-side argument* and display them accordingly to uniformize across different pairings[1].

Scott also proposed in [Sco11] a CP-ABE scheme that takes advantage of the Fixed Argument Optimization:

Setup: Select random points $P \in \mathbb{G}_2$ and $Q \in \mathbb{G}_1$. If the pairing friendly group is of size r, then pick random group elements α and $\delta \in \mathbb{Z}_r$. Set $P_d = \delta P$, $Q_d = \delta Q$, $Q_\alpha = \alpha Q$ and $v = e(P,Q)^\alpha$. For each of the U attributes in the system, generate a random $H_i \in \mathbb{G}_2$. The public parameters are $\{P, Q, v, P_d, Q_d, H_1 \cdots H_U\}$. The master key is Q_α.

Encrypt: The inputs are a message M, and a $m \times n$ LSSS matrix S. Generate a random vector $\bar{u} = (s, y_2, ...y_n) \in \mathbb{Z}_r$. Then calculate the m vector $\bar{\lambda} = S \cdot \bar{u}$ and generate another random m vector $\bar{x} \in \mathbb{Z}_r$. Calculate the ciphertext as $C_t = Mv^s$, $C_d = sP$, and for i equal 1 to m calculate $C_i = \lambda_i P_d - x_i H_{f(i)}$ and $D_i = x_i Q$. Note that the same attribute may be associated with different indices i.

KeyGen: The inputs are the master key and a set of l attributes A assigned to an individual. Pick a random group element $t \in \mathbb{Z}_r$ and create a private key as $K = Q\alpha + tQ_d$, $L = tQ$ and $K_i = tH_i$ for each possessed attribute $i \in A$.

Decrypt: First reduce the matrix S by removing rows associated with attributes that are not in A and remove redundant all-zero columns from the matrix. Next calculate the vector $\bar{\omega}$ which in the first row of S^{-1}. For a reasonable number of attributes, the ω_i will be very small integers. (The shared secret $s = \bar{\omega} \cdot \bar{\lambda}$.) Set all $C_j \leftarrow \omega_j C_j$ and $D_j \leftarrow \omega_j D_j$. Where the same attribute is associated with more than one row of the S matrix, combine the associated C_j and D_j values by simply adding

[1] In the literature Tate parings are often written with their precombutable argument on the left side, but it is the opposite for ate pairing

them. (We exploit bilinearity as $e(K_i, D_j) \cdot e(K_i, D_k) = e(K_i, D_j + D_k)$, and rewrite $D_i = D_j + D_k$). Finally recover the message as

$$M = C_t \cdot e(C_d, -K) \cdot e\big(\sum_{i \in A} C_i, L\big) \cdot \prod_{i \in A} e(K_i, D_i).$$

During decryption, only pairings where the left argument is known beforehand (i.e. are part of the keys) can benefit from Fixed Argument Optimization. Since all K_i are part of the decryption key, pairings $e(K_i, D_i)$ can benefit of pre-computation with the Fixed Argument Optimization. However, pairings $e(C_d, -K)$ and $e\big(\sum_{i \in A} C_i, L\big)$ do not since C_d and $\sum_{i \in A} C_i$ are part of the ciphertext.

3.2 Notion of Perfect Argument Order and Switch Argument Method

3.2.1 Perfect Argument Order

First of all, we introduce the notion of Perfect Argument Order.

Definition 7 (Perfect Argument Order)**.** An ABE scheme using Type III pairings has a **Perfect Argument Order**, and is called **Perfectly Ordered** if for all pairings, the left-side arguments are obtained before or along the right-side ones. That implies that if the left-side argument depends of the ciphertext, the right-side argument also does.

More roughly, it means all pairing computations done in the Decrypt algorithm that could benefit from the Fixed Argument Optimization actually benefit from it.

For example, Scott's scheme has not a Perfect Argument Order since two pairings have their left-side argument dependant of the ciphertext, while right-side ones are from the secret key. In the following, we propose the **Switch Argument Method** that switches arguments of any pairing in the decryption, making their arguments Perfectly Ordered.

3.2.2 Switch Argument Method

The fundamental idea behind the Switch Argument Method is that for all pairings during decryption where the left-side argument is dependant of the ciphertext, we can make an "argument swap" by using at the same time pairing bilinearity property and the fact that any element can be expressed as a power of the group generator.
More precisely, we rely on the following property:

$$e(aP, bQ) = e(bP, aQ) \tag{1}$$

By doing so, we can effectively "swap" arguments of any pairing by swapping multiplicities a and b. We can note that, at some point, the method requires controlling multiplicities. In general, since the correctness of the scheme does not require such control, designers often pick random points with unknown multiplicity order.

Consequently, our method requires to pick randomly generators P and Q only, and then express following random points as multiple of P for key elements and Q for ciphertext elements. In practice, this implies to pick random integer $a \in \mathbb{Z}_{\text{l}}$ then computing aP for secret key elements or aQ for ciphertext elements.

This way, no more Ciphertext elements are the left-side arguments of the pairings. All of them benefit of the Fixed Argument Optimization, and the scheme has now a Perfect Argument Order.

3.3 Construction of a Perfectly Ordered CP-ABE

In the paper presenting the Fixed Argument Optimization [Sco11], Scott carefully selected group $\mathbb{G}_1$ and $\mathbb{G}_2$ to apply Fixed Argument Optimization to the majority of pairings. However, two last pairings could not be optimized and he believed that he applied the Fixed Argument Optimization to the maximum number of pairings of Waters' scheme [Wat11].

Thanks to our Switch Argument Method, we tackle this limitation and provide below a modified version of the scheme achieving Perfect Argument Order. We can note that, to the best of our knowledge, the method can be applied to all existing ABE schemes. More information is provided in Section 5.

Modification of Scott's Scheme: First of all, pairings that are "wrong-sided" in Scott

construction are those that depend on C_d, K, C_i and L. To apply our method, we first expressed them as multiple of the group generator, then swapped them using bilinearity. We provide below the explicit expression of each elements before application of the Switch Argument Method (left) and after (right):

$$(2)\begin{cases} C_d = sP \\ C_i = \lambda_i \delta P - x_i H_{f(i)} \\ L = tQ \\ K = \alpha Q + t\delta Q \end{cases} \implies (3)\begin{cases} C_d = sQ \\ C_i = \lambda_i \delta Q - x_i h_{f(i)} Q \\ L = tP \\ K = \alpha P + t\delta P \end{cases}$$

We can note that the remark we made in the description of our method about picking random points after P and Q concerns perfectly Waters scheme. Indeed, we turned point $H_{f(i)}$ as multiple of P such as $H_{f(i)} = h_{f(i)}P$, then swapped it using equation 1.

We can now explicit the Perfectly Ordered version of Waters' CP-ABE scheme.

Setup: Select random points $P \in \mathbb{G}_2$ and $Q \in \mathbb{G}_1$. If the pairing friendly group is of size p, then pick random group elements α and $\delta \in \mathbb{Z}_r$. Set $P_d = \delta P$, $Q_d = \delta Q$, $P_\alpha = \alpha P$ and $v = e(P, Q)^\alpha$. For each of the U attributes in the system, pick a random group element $x_i \in \mathbb{Z}_r$. Set $h_i = x_i P$ and $k_i = x_i Q$. The public parameters are $\{P, Q, v, Q_d, k_1, \cdots, k_U\}$. The master key is $\{P_\alpha, P_d, h_1, \cdots, h_U\}$.

Encrypt: The inputs are a message M, and a $m \times n$ LSSS matrix S. Generate a random vector $\bar{u} = (s, y_2, ...y_n) \in \mathbb{Z}_r$. Then calculate the m vector $\bar{\lambda} = S \cdot \bar{u}$ and generate another random m vector $\bar{r} \in \mathbb{Z}_r$. Calculate the ciphertext as $C = Mv^s$, $C' = sQ$, and for i equal 1 to m calculate $C_i = \lambda_i Q_d - r_i k_{f(i)}$ and $D_i = r_i Q$. Note that the same attribute may be associated with different indices i.

KeyGen: The inputs are the master key and a set of l attributes A assigned to an individual. Pick a random group element $t \in \mathbb{Z}_r$ and create a private key as $K = P\alpha + tP_d, L = tP$ and $K_i = th_i$ for each possessed attribute $i \in A$.

Decrypt: First reduce the matrix S by removing rows associated with attributes that are not in A and remove redundant all-zero columns from the matrix. Next calculate the vector $\bar{\omega}$ which in the first row of S^{-1}. For a reasonable number of attributes, the ω_i will be very small integers. (The shared secret $s = \bar{\omega} \cdot \bar{\lambda}$.) Set all $C_j \leftarrow \omega_j C_j$ and $D_j \leftarrow \omega_j D_j$. Where the same attribute is associated with more than one row of the S matrix, combine the associated C_j and D_j values by simply adding them. (We exploit bilinearity as $e(K_i, D_j) \cdot e(K_i, D_k) = e(K_i, D_j + D_k)$, and rewrite $D_i = D_j + D_k$). Finally recover the message as

$$M' = C \cdot e(-K, C') \cdot e\big(L, \sum_{i \in A} C_i\big) \cdot \prod_{i \in A} e(K_i, D_i).$$

Correctness:

$$M' = C \cdot e(-K, C') \cdot e\big(L, \sum_{i \in A} C_i\big) \cdot \prod_{i \in A} e(K_i, D_i)$$

$$= M \cdot e(P,Q)^{s\alpha} \cdot e\big(-(P\alpha + tP_d), sQ\big) \cdot e\big(tP, \sum_{i \in A} \omega_i \lambda_i Q_d - \omega_i r_i h_i\big) \cdot \prod_{i \in A} e(tk_i, r_i Q)$$

$$= M \cdot e(P,Q)^{s\alpha} \cdot e(P,Q)^{-s\alpha - st\delta} \cdot e\big(tP, \sum_{i \in A} \omega_i \lambda_i Q_d - \omega_i r_i x_i Q\big) \cdot \prod_{i \in A} e(tx_i P, \omega_i r_i Q)$$

$$= M \cdot e(P,Q)^{s\alpha} \cdot e(P,Q)^{-s\alpha - st\delta} \cdot e(P,Q)^{t\left(\sum_{i \in A} \omega_i \lambda_i \delta - \omega_i r_i x_i\right)} \cdot \prod_{i \in A} e(P,Q)^{t\omega_i r_i x_i}$$

$$= M \cdot e(P,Q)^{s\alpha} \cdot e(P,Q)^{-s\alpha - st\delta} \cdot e(P,Q)^{\sum_{i \in A} t\omega_i \lambda_i \delta}$$

$$= M.e(P,Q)^{s\alpha} \cdot e(P,Q)^{-s\alpha - st\delta} \cdot e(P,Q)^{st\delta}$$

$$= M$$

Now all pairings of the decryption have their left argument known before the right ones. All of them benefit of the Fixed Argument Optimization because the scheme is now Perfectly Ordered.

3.4 Important notes

Turning a given ABE scheme to its Perfectly Ordered version speeds-up decryption in any case, but has, at first sight, some impact on other parts of the algorithm as long as they require random curve points picked. Indeed, for random points where a control of the multiplicity is required, one need at some point to choose random $a \in \mathbb{Z}_p$ and make the full point multiplication with the generator. For random points without the requirement of multiplicity control, exponentiation time is reduced since the following "trick" (described in [Lyn07] ch 5.5) can be often used:

1. Pick random X axis value.
2. Use the curve equation to find a point with such X. If there is no such point, try again with X+1.
3. Multiply the point with the cofactor of the curve to ensure that the picked point is in the group of the right order.

Considering this, curves that are recommended in standards are often chosen to have a small cofactor, letting the point multiplication be fast (this shows that the choice of the best curve is really crucial for a scheme).

In most finite-universe ABE schemes, this is a very slight drawback because the attribute-linked random picks are performed during Setup, and Setup times are considered irrelevant since they are computed once. For Large Universe ABE schemes, where attributes are generated "on-the-fly", random picks are generally performed during Encryption, making a possible overhead during this phase.

Fortunately, to bypass this limitation, we found a smart method to move much of random picks to the Setup phase. This proposition allow us to make at the same time Encryption and Decryption faster. More information is provided in section 7.2.4.

4 Multi-Locking

In this section, we will describe the concept and construction of the "Multi-Locking" framework. In a few words, Multi-Locking allows efficient composition of several ABE schemes during transfer of data, making construction of advanced infrastructures particularly simple. Additionally, the framework allows designers to implement and combine the following features:

1. the possibility to impose a specific route for the data.
2. the possibility to delegate access control to trusted nodes.

In the following, we will describe how KP-ABE can benefit of Multi-Locking. Note that the method still stand for CP-ABE and more generally for various encryption algorithms.

4.1 Introductory example

Consider the following scenario: young Alice $\mathcal{A}$ wants to send messages to Bob $\mathcal{B}$, but her mother $\mathcal{M}$ wants to be sure she does not send messages after 10 pm and that Bob is really the intended recipient. They reach an agreement, but Alice insists that her mother should not be able to read her messages. The scheme they will use is the following: $\mathcal{A}$ locks the message with two separate locks, $L_{\mathcal{AB}}$ and $L_{\mathcal{AM}}$ and sends it. $\mathcal{M}$ then checks if it is sent before 10 pm and if it is the case, she will unlock $L_{\mathcal{AM}}$ with the corresponding key $K_{\mathcal{AM}}$, add a new lock $L_{\mathcal{MB}}$ and pass forward the new message. $\mathcal{B}$ then receives the message and is able to use his two keys $K_{\mathcal{MB}}$ and $K_{\mathcal{AB}}$ to get the plaintext sent from Alice.

4.2 Definition of a Multi-Locking scheme

Now we should list all the properties we expect to hold when implementing a Multi-Locking scheme: let us name the original sender the Source, the intermediary users the Relays and the final reader the Target.

First, we present the two hypothesis we assume to hold for a Multi-Locking implementation:

Keys conservation: The Relays and the Target will not distribute their private keys.

Trustworthiness: The Relays will add or remove locks according to the access policy agreement.

Then, these are the properties a Multi-Locking scheme is able to provide:

Circuit: The scheme defines possible circuits by providing specific locking/unlocking capabilities to the Source, Relays and Target. No Relay is able to retrieve the plaintext, and the Target is not able to retrieve it before the end of the circuit among the relays.

Correctness: The Target retrieves the right plaintext message.

Privacy: Only the Target can decrypt the ciphertext.

Collusion Resistance: *this concerns ABE schemes* A given Relay or Target is not a specified user, but a set of users sharing attributes (CP-ABE) or authorizations (KP-ABE). A set of users will never be able to access data if none of its parts can access it.

Definition 8 (Cryptographic scheme)**.** Let S be a **cryptographic scheme**. It is the definition of:

- A space of encryption keys K, decryption keys K', a space of plaintexts M and a space of ciphertexts C
- An Encryption $E : K \times M \to C$
- A Decryption $D : K' \times C \to M$
- A morphism $\phi : K \to K'$, such that $D(\phi_i(k), E(k, \cdot)) = Id_M$

Definition 9 (Multi-Locking Family)**.** A **Multi-Locking Family** is a family of n cryptographic schemes $\{S_i \ , i \in I\} = \{(K_i, K'_i, M_i, C_i, E_i, D_i, \Phi_i)\}_{i \in I}$ such as:

1. $\forall i, j \in I, M_i \cong M_j \cong C_i \cong C_j$
2. $\forall i, j \in I, F \in \big\{\{E_i\} \cap \{D_i\}\big\}, k_F \in K_F, G \in \big\{\{E_j\} \cap \{D_j\}\big\}, k_G \in K_G, F(k_F, \cdot) \circ G(k_G, \cdot) = G(k_G, \cdot) \circ F(k_F, \cdot)$

Notation: We call a multi-locking family accordingly to the operation responsible of Encryption/Decryption commutativity *Op*. If useful we also specifying the number of primes i in the factorized cardinality of the plaintext and ciphertext spaces. Thus, we call a generic Multi-Locking Family by the name "$p_i - Op$-MLF".

4.3 Common Multi-Locking Families

4.3.1 Xor-MLF: case of One-Time Pad scheme

The One-Time Pad cryptographic scheme with fixed message length n defined below is part of a Multi-Locking Family:

- $K = M = C = \mathbb{Z}_{2^n}$
- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$
- $\phi(k) = k$

Property 2 from Multi-Locking Family definition can also be straightforwardly verified:

$$\forall (m, k_1, k_2) \in (\mathbb{Z}_{2^n})^3, \ (m \oplus k_1) \oplus k_2 = (m \oplus k_2) \oplus k_1 \tag{2}$$

Since Encryption/Decryption commutativity is derived from the commutativity of XOR, this construction belongs to Xor-MLF. In particular, a OTP multi-locking scheme is secure as long as every k_i is chosen at random uniformly since $k_1 \oplus k_2$ is indistinguishable from another key chosen at random uniformly.

4.3.2 p_2-mul-MLF: case of Textbook-RSA

Given $N = p \cdot q$, Textbook RSA forms a Multi-Locking Family because,

- $K = \mathbb{Z}_{(p-1)(q-1)}, M = C = \mathbb{Z}_N$
- $E(k, m) = m^k[N]$
- $D(k, c) = c^k[N]$
- $\phi(k) = k^{-1}[(p-1)(q-1)]$

$$\forall(m, k_1, k_2) \in \mathbb{Z}_N \times \mathbb{Z}^2_{\phi(N)},\ (m^{k_1}[N])^{k_2}[N] = (m^{k_2}[N])^{k_1}[N] \quad (3)$$

Important note: In contrast to previous construction, the use of RSA inside p_2-Exp-MLF is not secure: in order to form a Multi-Locking family, RSA schemes need to share a same N (if not, plaintext and ciphertext spaces are different, so encryption and decryption no longer commute). However, it is known for a long time that having multiple instances of RSA sharing the same modulus leads to major security issues [B+98, HL10, SIM83]. An construction in p-Exp-MLF would not be secure since modular inversion would be easy. However, it would be interesting to study the security of p_i-Exp-MLF with $i > 2$.

4.3.3 p-mul-MLF: case of ABE schemes

Most of ABE schemes (and more generally FE schemes) naturally belong to p-mul-MLF. For example, the wide range of schemes are constructed with $E(k, m) = m \cdot e(g_1, g_2)^{\alpha s}$ and $D(k, c) = c \cdot e(g_1, g_2)^{-\alpha s}$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ have a prime order.

These constructions are obviously Multi-Lockable since modular multiplications are commutative, thus they fall inside p-mul-MLF. For information, all ABE schemes detailed previously in this paper are part of this family.

However, we should make an important note on ABE schemes: Some of them are not part of p-mul-MLF but belong to another Multi-Locking Families, making them not Multi-Lockable with p-mul-MLF ones. A contrario, they can be composed with other cryptographic schemes that share the same family, making an open area to smart compositions.

4.3.4 Construction of Multi-Locking Scheme based on p-mul-MLF ABE schemes

To make the presentation of Multi-Locking scheme with ABE easier, we will explicit a reduced version called "tripartite Multi-Locking" composed of a unique Source, Relay and Target, corresponding to the former example of Alice, her mother and Bob.

To enable Multi-Locking, we should at some point let the ciphertext be partially decrypted by Relay and Target. As a recap, a ciphertext of ABE has the form $C = m \cdot e(P, Q)^{\alpha s}$, with s picked uniform at random during Encryption and removed during Decryption using information from ciphertext.

To ensure that ciphertext is only partially decryptable by Relay and Target, we must at some point generate a specific uniformly random value for each "Lock". This values will correspond to layers of security of the data. As in the example of Alice presented before, we will need three locking and unlocking mechanisms, namely $\mathcal{AB}$, $\mathcal{AM}$, and $\mathcal{MB}$.

Currently, to be safe against replay attacks, every Encrypt, Source selects a random number $s \in \mathbb{Z}_p$. This is equivalent security-wise to pick n random numbers $s_1 \cdots s_n \in \mathbb{Z}_p$ and set $s = \sum s_i$ (distribution for s is uniform if for all i, s_i is picked uniformly). This will allow us to add or remove "Locks" (S_i) while guarantee the privacy of the data if there is at least one lock at any given time.

During the Setup phase, we arbitrarily select a partition of the universe of attributes U in n (here 3) sub-universes. We note them $U_{\mathcal{AB}}, U_{\mathcal{AM}}$ and $U_{\mathcal{MB}}$. They addresses specific possibilities of access control. Each subsequent users can be further narrowed down by users that have locking capabilities.

In the Encrypt phase, for all sub-universes of U in which the Source have control possibilities (namely $U_{\mathcal{AB}}$ and $U_{\mathcal{AM}}$), it will pick uniformly at random a corresponding $s_{\mathcal{AB}}$ (resp. $s_{\mathcal{AM}}$). Then it will have to encrypt as normal successively in every sub-universe, each time taking the message-dependant part of the ciphertext as the input of the next encryption. Every other ciphertext element required by the specific scheme for a sub-universe is simply appended to the previous ciphertext.

For example in CP-ABE it will share s_i using the LSSS matrix L_i of the sub-universe U_i and so, three matrices will be joined to the ciphertext, the width of each one being the number of leaves of the corresponding sub-universe access tree. Note that we could use a single matrix, but it will be filled with mostly zeros, we will prefer multiple smaller ones for efficiency.

During the KeyGen phase, we can deliver wisely the keys to force the circuit. This implies having keys in only some universes. For example Bob should not have keys in $U_{\mathcal{AM}}$. Note that it does not change from our usual KeyGen for CP-ABE but for KP-ABE it forces us to calculate keys separately for the three sub-universes (one LSSS per sub-universe).

In practice $\mathcal{M}$ will run Decrypt with all elements generated by the encryption in $U_{\mathcal{AM}}$ to retrieve $C' = m \cdot e(P,Q)^{\alpha s_{\mathcal{AB}}}$. As planned, $\mathcal{M}$ cannot access to the plaintext before $\mathcal{B}$. Since the relay is suposed trustworthy, it will run Encrypt using C' as input, to get $c'' = m \cdot e(P,Q)^{\alpha s_{\mathcal{AB}}} \cdot e(P,Q)^{\alpha s_{\mathcal{MB}}}$ and append the other scheme-specific elements.

Ultimately, $\mathcal{B}$ can run Decrypt once with each set of keys (once in $U_{\mathcal{AB}}$ to remove $s_{\mathcal{AB}}$ and once in $U_{\mathcal{MB}}$ to remove $s_{\mathcal{MB}}$) in order to retrieve M.

Because of the form taken by the message-dependent part of the ciphertext $C = m \cdot e(P,Q)^{\alpha s}$ that grants inclusion into the p-Mul-MLF, we can see that all properties guaranteed by a Multi-Locking scheme are met. Collusion Resistance is granted by the collusion resistance protecting each secret S_i

4.4 Generalization

We will propose an extension to the idea of tripartite Multi-Locking. For this we will show how we can generate a Multi-Locking scheme matching a given circuit type we will consider general enough.

Definition 10 (Sequential Circuit)**.** We call the source $\mathcal{S}$, the target $\mathcal{T}$ and the different groups of relays $\mathcal{R}_1, \mathcal{R}_2 \cdots$. (If a same user have the required attributes or credentials to be in several groups we will consider two distinct groups).

We call a Sequential Circuit an expression linking different groups of relays with the following natural operators: OR ($\vee$), AND ($\wedge$), and THEN ($\rightarrow$), which beginning is $\mathcal{S} \rightarrow$ and which end is $\rightarrow \mathcal{T}$.

The example with Alice presented earlier corresponds to the Sequential Circuit $\mathcal{S} \rightarrow \mathcal{R}_1 \rightarrow \mathcal{T}$.

For instance, $\mathcal{S} \rightarrow (\mathcal{R}_1 \vee \mathcal{R}_2) \rightarrow (\mathcal{R}_3 \wedge \mathcal{R}_4) \rightarrow \mathcal{T}$ is a valid Sequential Circuit. This example would correspond to a message being checked by branch director or secretaries before being checked by IT and CCO's and then sent.

Property 1. Every $(\mathcal{A}) \wedge (\mathcal{B})$ is equivalent to $((\mathcal{A}) \rightarrow (\mathcal{B})) \vee ((\mathcal{B}) \rightarrow (\mathcal{A}))$.

Property 2. $\rightarrow$ distributes over $\vee$ i.e. every $(\mathcal{A}) \rightarrow ((\mathcal{B}) \vee (\mathcal{C}))$ is equivalent to $((\mathcal{A}) \rightarrow (\mathcal{B})) \vee ((\mathcal{A}) \rightarrow (\mathcal{C}))$ and $((\mathcal{B}) \vee (\mathcal{C})) \rightarrow (\mathcal{A})$ is equivalent to $((\mathcal{B}) \rightarrow (\mathcal{A})) \vee ((\mathcal{C}) \rightarrow (\mathcal{A}))$.

Proposition 1. *For each Sequential Circuit, at least one Multi-Locking scheme implementing it exists.*

Proof. We will prove the existence of it in two steps: first, we will show that every sequential circuit is equivalent to a sequential circuit of the form $\mathcal{S} \to (\mathcal{M}_1) \vee \cdots \vee (\mathcal{M}_n) \to \mathcal{T}$, where every $\mathcal{M}_i$ is of the form $\mathcal{R}_{i1} \to \mathcal{R}_{i2} \to \cdots \to \mathcal{R}_{i\omega}$. Second, we will show how to implement this exact type of circuit.

To transform any Sequential Circuit in the desired form, we fist remove all the $\wedge$ using property 1 and then distribute all $\to$ over the $\vee$ using property 2.

For example: $\mathcal{S} \to (\mathcal{R}_1 \vee \mathcal{R}_2) \to (\mathcal{R}_3 \wedge \mathcal{R}_4) \to \mathcal{T}$ is equivalent to

$$\begin{aligned} \mathcal{S} \to \ & ((\mathcal{R}_1 \to \mathcal{R}_3 \to \mathcal{R}_4) \\ & \vee (\mathcal{R}_1 \to \mathcal{R}_4 \to \mathcal{R}_3) \\ & \vee (\mathcal{R}_2 \to \mathcal{R}_3 \to \mathcal{R}_4) \\ & \vee (\mathcal{R}_2 \to \mathcal{R}_4 \to \mathcal{R}_3)) \to \mathcal{T} \end{aligned} \tag{4}$$

To implement the Multi-Locking schemes corresponding to all the Sequential Circuits of this kind, we can do the following: $\mathcal{S}$ will lock the message with the locks L_{ST} and L_{S1} and broadcast it. The first relays in all $(\mathcal{M}_i)$ have K_{S1}, remove the corresponding lock, and adds locks L_{i1T} and L_{i2}. Similarly, all others relays j in $(\mathcal{M}_i)$ except the last will use their key K_{ij} and add locks L_{ijT} and $L_{i\,j+1}$. The last relay uses $K_{i\omega}$ and adds lock $L_{i\omega T}$ only. Finally, $\mathcal{T}$ can use all K_{ijT} keys and K_{ST} to retrieve the message.

Since lock L_{ST} is always kept, we can verify easily that no one is able to retrieve the message before $\mathcal{T}$ does. Moreover, during all broadcasts, there is at least one lock where $\mathcal{T}$ does not have the corresponding key. Thus $\mathcal{T}$ cannot retrieve the message before the circuit is ended, ending the proof.

□

4.5 Benefits over super-encryption

So far, Multi-Locking does not lead to more expressivity than super-encryption. In fact, in the previous proof, the proposed implementation works very well with super-encryption. To visualize this, we will propose the Lock-key graphs. They are finite oriented binary connected graphs where all vertices are groups in the scheme (i.e. source, relays, target). The start of an edge shows who puts a lock and the end of this edge indicates who possesses the corresponding key. These graphs have no loops since we consider users intervening many times will count in as many of separated vertices. In the following, we highlighted L_{ST} for visibility.

Thereafter is presented the construction proposed in the above proof (1).

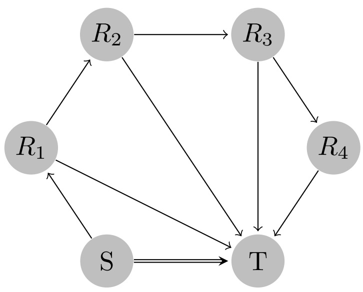

Figure 1: Lock-key graph of an scheme implementing of a Sequential Circuit composed only by five successive →.

In particular, any node at the end of an edge needs information broadcast by the node at the beginning of this edge. Thus, this construction achieves the circuit property of

Multi-Locking. Furthermore, Figure 1 represents only one $\mathcal{M}_i$. A full implementation for $\mathcal{S} \rightarrow (\mathcal{R}_1 \vee \mathcal{R}_2) \rightarrow (\mathcal{R}_3 \wedge \mathcal{R}_4) \rightarrow \mathcal{T}$ would be represented like in Figure 2.

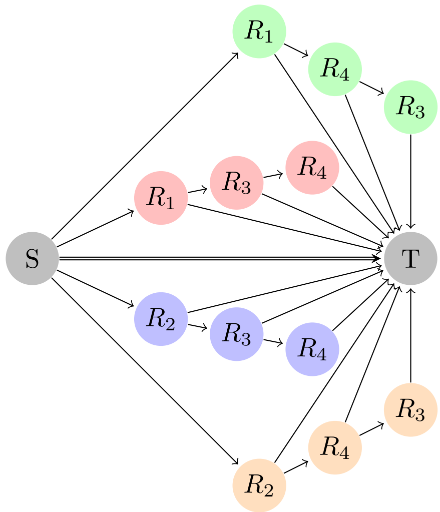

Figure 2: Lock-key graph of an implementation of $\mathcal{S} \rightarrow (\mathcal{R}_1 \vee \mathcal{R}_2) \rightarrow (\mathcal{R}_3 \wedge \mathcal{R}_4) \rightarrow \mathcal{T}$.

Note that all edges are not necessary for a working scheme. All the ones ending in T except S-T are optional. We can see that every Sequential Circuit implementation can be represented in at least one planar Lock-key graph. As a visualisation, Figure 2 shows us how to transform the developed form of our example Sequential Circuit (4) into a planar Lock-key graph.

In super-encryption, locks work as a stack. So all schemes can be represented with a Sequential Circuit[2] and so with a planar Lock-key graph. It motivates the following proposition:

Proposition 2. *A scheme implementing a Sequential Circuit following the requirements listed in section 4.2 admits a Planar Lock-key (weakly connected, oriented, finite, binary) graph.*

A Planar Lock-key graph with two or more vertices is the representation of a scheme implementing a Sequential Circuit (SC) if every walk starting in S ends in T and there is an edge from S (only vertex without antecedent) to T (only vertex without successor).

Proof. The first point was discussed earlier.

For the second point, the existence of the S-T edge guarantees *privacy*.

Besides this, every walk ends in T so there is at least one edge $\mathcal{R}_i$ pointing only to T (who has no successor). So we can define a sub-graph representing the SC $(\mathcal{R}_i \rightarrow T)$. We create as many SCs that there are points like this and list them $\mathcal{M}_i$.

From there, we will iterate the following until that all edges are added to a single SC:

There is at least one edge with a predecessor in our list, and all walks lead to T so there is at least one edge $\mathcal{R}_\alpha$ pointing only to nodes already in our list. For every one of them, and for every SC $\mathcal{M}_i$ starting with this node, we add $\mathcal{R}_\alpha \rightarrow \mathcal{M}_i$ to our list.

This ends when S, with no antecedent, is included to all elements of the list. This termination is guaranteed because the graph is finite and S is the only vertex with no predecessor.

We define $Circuit := \bigvee_i (\mathcal{M}_i)$, then factor all starting S and ending T so it is in a valid SC form. This ends the proof.

□

[2]Since decryption must be done in the reverse order of the locks

We can conclude that since all super-encryption schemes can be expressed by a Sequential Circuit, they all have a planar Lock-key scheme.

But in spite of super-encryption, Multi-Locking allows us to have non-planar Lock-key graphs like Figure 3.

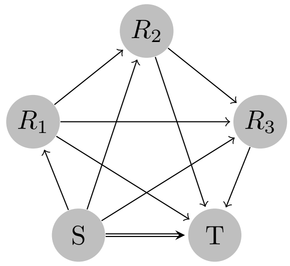

Figure 3: A Lock-key graph that is not planar.

To finish this section, we want to put forward a metaphor: super-locking is like putting a locked case inside a locked case inside a locked case... This makes all the cases untouchable except the outer one. Instead, Multi-Locking corresponds to adding and removing multiple locks on the same case. In this case, they are all simultaneously accessible.

4.6 Computational consequences

Multi-authority support: It happens that different trusted authorities can manage each of one these universes and the distribution of the corresponding keys. As long as all the authorities have the Master Secret Key for the scheme, this allows a per-universe Multi-authority construction. Sub-universes where the ABE scheme is Multi-authority allows for even finer distribution of the KeyGen responsibilities.

A second remark is that the construction proposed in the proof (using developed Sequential Circuit form) corresponds to the worst case scenario. There are almost always some ways to factorize the number of keys needed, but this is out of the scope of this paper.

Scheme Modifications: Given this construction, we can already study the impact on our KP-ABE scheme. First, in the setup phase the creation of the σ new universes of attributes is not affecting efficiency since it just requires to have the attribute names start with the name of the universe they belong to. Then, at usual, we have to link each attributes to a value. Note that the computation times are the same if there is no increase of the number of attributes. The only requirement is to have at least one attribute per sub-universe.

The partition of the universe of attributes allows a gain in bandwidth used for transmission of keys, and space needed for storage: not all users will want to use all of the attributes for encryption (they do not encrypt in every possible universe). The cherry-picking of universes according to the wanted attributes keeps minimal the storage and transmission of keys.

Finally, the number of decryptions made across the full scheme has increased sightly. In most schemes the number of pairings required during Decrypt can be decomposed in two parts: one part scaling with the number of lines in the LSSS, and one part being constant, depending of the scheme, but usually very low (1-3 for most schemes). For equivalent access structures, the linear number of pairings do not change between a plain scheme and a Multi-Locking one. However the constant number stacks with every decryption. Even if this is negligible in a scalable scheme, this supports even more the choice of Perfect Argument Order. This is will be even more true for the later relays of the scheme, whose number of decryptions has increased the most.

The last impact to study concerns the ciphertext. This impact depends strongly on the scheme. However, in order to give a substantial example, we describe the consequences on a Multi-locking ABE scheme based only on KP-ABE with constant ciphertext length[ALdP11]. For 80 security bits and an optimal embedding degree $k = 6$, the length of a emitted message is $1026 + (2 \times 170 \times \delta)$. δ is the number of current locks protecting the message, i.e. the number of edges going out of the vertex and all his predecessors, minus the number of incoming edges for this vertex and all his predecessors. In general for an embedding degree of k, the plaintext/ciphertext ratio is $\frac{k}{k+\delta}$.

ABE composition: The real payout of this construction is that multiple ABE instances can be used in this scheme. We can mix CP-ABE with KP-ABE and even with Fuzzy-IBE, the only constraint is that the schemes belong in the same Multi-Locking family.

This brings us to our final remark: it happens that the property of trustworthiness may be too strong in real case. In fact, without this property, a single user can not enforce the circuit by itself. In the worst case scenario, if the source adds a lock for every relay, it can at least guarantee that the message will go through all of them, or will not be deciphered. The fact that order does not matter (commutativity required for Multi-Locking) will give us more possible uses.

5 Discussion about the scope of the improvements

5.1 Compatibility of Perfect Argument Order with existing optimizations

In the litterature, two distinct types of optimizations exist: the first one concerns the optimisation of the pairing computation time, and the second one concerns modification of the scheme for efficiency.

The first category is mostly represented by the work of Costello and Stebila in [CS10]. Authors propose a way to accelerate the product of pairings with no common argument. Since this acceleration can be applied with or without the Fixed Argument Optimization, we consider using the Multi-Pairing Optimization natural for all implementations.

For optimization that falls into the second category, they basically require a modification of the scheme itself. Consequently, it requires at some point a comprise between Perfect Argument Order and existing optimizations. The most powerful one is the so called Decryption Optimization of Pirretti [PTMW10b]. To illustrate this optimization and its intricacy with Perfect Argument Order, lets first describe how related schemes are constructed.

Let us imagine a scheme where Decrypt contains $\prod_{i \in S} e(D_i, E)^{\omega_i}$, with D_i derived from SK and E from the ciphertext. These pairings can be accelerated by precomputation with the Fixed Argument Optimization. In particular the previous computation requires $|S|$ pairings, $|S|$ exponentiations in $\mathbb{G}_T$ and $|S| - 1$ multiplications in $\mathbb{G}_T$.

The Pirretti optimization is based on bilinearity property of pairings, dispatching the external product computation into the left argument:

$$\prod_{i \in S} e(D_i, E)^{\omega_i} = e(\prod_{i \in S} D_i^{\omega_i}, E) \tag{5}$$

This leads to two great optimizations. First, only 1 pairing computation is needed instead of $|S|$. Second, exponentiations and multiplications are now performed on much smaller groups since $\mathbb{G}_1$ is smaller than $\mathbb{G}_T$ by a factor defined by the embedding degree k.

Since the set S is specified in the ciphertext, the pairing touched by Decryption Optimisation now have both his arguments obtainable at the same time, so, following definition 7 this pairing is can no longer be subjected to Fixed Argument Optimization and the scheme remains Perfectly Ordered.

Note that since Decryption Optimization reduces the number of pairings (right side of eq.5) it is better to use it and renounce precomputation possibilities than to calculate the product as in the right side of eq. 5. As a consequence it can be useless to use the Swap Argument Method.

However, one should consider it in some specific cases. For example, a really small universe of attributes, implies a reduced number of possible values of $\prod_{i\in S} D_i^{\omega_i}$. In this case, one should calculate these points and use the Fixed Argument Optimization precomputation possibilities. That would require this point to be the left-side argument.

We see no generalizable transformation to apply to a scheme in order to make the Decryption optimization applicable, so we will consider this possibilities dependent of the scheme design.

Note that in Scott scheme presented earlier, Pirretti's Decryption Optimization was not applicable, making it a perfect candidate for Perfect Argument Order Optimization.

5.2 Compatibility of Perfect Argument Order and Multi-Locking with existing ABE schemes

In the following, we will provide a survey of the most relevant existing ABE schemes listed in [QLDJ14] and their compatibility with Perfect Argument Order and Multi-Locking.

Below notation used in table 1:

MLF: Scheme is part of a multi locking family. Additionally, schemes with a "•" falls into the p-Mul-MLF.

Perfectly Ordered: Scheme is based on a Type III pairing and perfectly ordered.

Category 1: Scheme is based on a Type I pairing, but the classical generalisation to a Type III pairing gives a Perfectly Ordered scheme.

Category 2: Scheme is based on a Type III pairing. Applying the Switch Argument Method provides Perfect Argument Order.

Category 3: Scheme is based on a Type I pairing. After generalizing it to a Type III pairing, applying the Switch Argument Method provides Perfect Argument Order.

Furthermore, to make meaningful nuances, we use the 3 different symbols for specific cases:

"•": indicates that the property is present for the scheme.

"⋄": indicates that the presence or not of the property cannot be directly given. Either a change could be made to the scheme in order to obtain the property, or some extra justification of the property presence or absence is needed. These cases are discussed in the list below.

"⋆": indicates that the scheme also benefits from Decryption Optimisation

Based on the table 1, we can make some interesting remarks:

- Almost all studied ABE schemes are part of a Multi-Locking Family, most of them in p-Mul-MLF. Exceptionally, some of them are in p_3-Mul-MLF and Xor-MLF, but can be composed with non-ABE schemes as stated in section 4.3.

Table 1: Survey of Multi-Locking and Perfect Argument Order in ABE schemes

Author	Contribution	MLF	P.O.	Cat. 1	Cat. 2	Cat. 3
[SA05]	Fuzzy IBE	•		•		
[GPSW06b]	Key policy	•		•		
[BSW07]	Ciphertext policy	•		•		
[PTMW10a]	Security	•		•		
[Cha07]	Multi-Auth.	•		⋆		
[BSSV09]	Decentralized	•				•
[OSW07]	Non-monotonic policy	•		⋆		
[MKE09]	Multi-Auth.	◇			•	
[MKE08]	Distributed Auth.	◇				•
[WLW10]	HABE	◇				•
[CC09]	Improved Multi-Auth.	•	•			
[LOS+10]	Decentralized Auth.	◇				•
[LCH+11]	W/O Random Oracle	◇				⋆
[LCLS10]	W/O Central Auth.	◇				•
[Hur13]	Revocation+Delegation	•				•
[HSMY12]	Privacy+Decentralized	•		•		
[WLWG11]	Revocation	◇				•
[LLLS10]	Revocation	•				•
[AHL+12]	Const Size Cipher.	•				•
[CWM+13]	Revocation					◇
[LHC+11]	Accountability	•	•			
[HJSNS08]	Anti-key-cloning	•				•
[HW13]	Fast Decryption	•				•

- We observe two groups: the simpler schemes that presents only one security property (in [QLDJ14]) are usually in category 1, and the more complex ones, intentionally combining more of these properties are usually in category 3.

  We believe that most category 1 schemes have a Perfect Argument Order just because of the aesthetics of having all ciphertext elements being the same argument for each pairing.

- Numerous schemes should have been much more efficient with slight modifications. In particular, those that does not generalize Type I pairings into Type III pairings even if it unlocks more inclusion degrees and thus efficiency in more security levels.

To clarify schemes marked with ◇, we provide additional information below:

Müller 2008-2009, Multi & Distributed Auth. In these two schemes, the part of the ciphertext that depends of the message is split in many pieces that each have the expected p-Mul-MLF form. We can make this scheme part of a Multi-Locking scheme implementation, but that would need every piece of the ciphertext to go through every subsequent computation, thus multiplying ciphertext length and computation time by the number of pieces. So in theory, these schemes are part of the p-Mul-MLF, but in practice it would be better not to include them in a scalable Multi-Locking scheme.

Lewko, 2011: Decentralized Auth. At first sight, the ciphertext seems to be in the same form than p-Mul-MLF schemes, but in fact the $\mathbb{G}$ group is not of prime order (on the contrary of all other schemes in the survey). Since the order of the $\mathbb{G}_T$ of all schemes have to be equal across this case of Multi-Locking Family, this scheme is not part of p-Mul-MLF, but p_3-Mul-MLF. Note that the author believes that it can be transformed into a prime order system.

Wang, 2011: HABE & Revocation. These schemes are not part of the p-Mul-MLF but instead part of the Xor-MLF since they rely on hiding the message with a $\oplus$. We believe that replacing the $\oplus$ with a modular multiplication (in Encrypt) and a modular multiplication by the inverse (in Decrypt) could make it part of the p-Mul-MLF.

Liu, 2011: W/O Random Oracle. This scheme, like Lewko's one, belongs to the p_3-Mul-MLF since the order is also multiplication of 3 primes, and the ciphertext have the right form.

Lin, 2008: W/O Central Auth. As it is the scheme requires appending a certain number l of zeroes at the end of the message to verify if a message if valid. If $l = 0$ then the scheme is part of p-Mul-MLF.

Cheng, 2013: Revocation. Relies on a generic CP-ABE to store the data. Whether this scheme is Perfectly Ordered or not depends on the choice of the underlying CP-ABE scheme.

6 Privacy-preserving Cloud service with ephemeral data access

If we make general considerations about connected objects, they are mostly designed to upload large amounts of data to Cloud servers. In return, device owners have access to services obtained by the merging of devices data.

Regarding data itself, if is usually stored permanently into servers with no particular control from device owners. Thus, if they want some kind of ephemeral access, making data available for a configurable time window, owners should rely only on the trustworthiness of the Cloud.

Moreover, even for a trusted Cloud, making conditional access control depending on a per-message configuration mixed with decryption time windows licensed to service providers is fundamentally complex and requires fine-grained authentication mechanisms.

To face this limitation, Multi-Locking ABE is an excellent candidate for a privacy-preserving Cloud service with ephemeral data access because it allows efficient share of access control policy between device and the Cloud.

Furthermore, we pushed Multi-Locking to the point that the Cloud's computations have been reduced to completely regular operations independent from device owners and service providers.

We detail the steps of the Multi-Locking scheme design and will then discuss the benefits:

- First, the user will encrypt the data, positioning all attributes he needs to enforce the accessibility he wants. If he wants to access the data later, he needs to check that he has the right accesses to decipher. If he has not, he also positions an "identity" attribute for which only him have secret keys. We will suppose that the Cloud will never have access rights in this "User-User" universe of attributes (Key conservation requirement of a secure Multi-Locking scheme, see section 4.2).

- Then, the user will encrypt again, but this time within a set of attributes that matches the authorizations of the Cloud (a "User-Cloud" universe). This Multi-Locking prevents anyone from accessing data that did not pass through the Cloud. The user can now broadcast the ciphertext.

- When receiving the ciphertext, the Cloud removes a lock using his "User-Cloud" keys and then stores the cipher data , marking it with a timestamp, and other metadata if it is useful.

- At the reception of a user's request to access some data, the Cloud first ciphers the data with a set of attributes in a "Cloud-User" universe, that is function of the metadata previously attached to the data. This function was given by the data owner beforehand. This needs to include a Δ_t attribute that represents the difference between the current time and the timestamp. This re-ciphered data can then be sent to the user requesting it.

- Finally, the end user will decipher the data if he has the authorizations to, by deciphering successively within the two universes of attributes (removing the two locks protecting the data).

The improvement brought by this construction comes from the possibility in some user's access tree to refine its authorizations with time conditions. Let us imagine a user Alice having as authorizations:

$$\big(\text{“}Videos\text{”} \wedge \text{“}\Delta_t < 1week\text{”}\big) \vee \big(\text{“}Patented\text{”} \wedge \text{“}\Delta_t > 10years\text{”}\big) \vee \big(\text{“}Alice'sData\text{”}\big)$$

The privacy gain can be considerable if the distribution of keys by the trusted authority is made carefully. Some Large Universe schemes authors already describethe possibility of having time-dependant elements in secret keys. In these schemes keys are said to be revocable but in reality they are just expirable. Updating access rights require constant renewing of secret keys.

In contrast, our construction does not require updating keys that passed their "lifespan". Instead, the key is kept the same, it is the data accessibility that is limited in time.

We can understand from looking at the previous example that a good ABE scheme for this construction would have a mechanism allowing Δ_t attributes to have an arbitrary precision (Year, milliseconds...) in order to be independent of a preset list of attributes given by the trusted authority like in [Sco11] or [ALdP11]. Thus, a Large Universe scheme is almost mandatory.

Moreover our supposition is that the Cloud is honest, but not the users. We cannot let the receiver users calculate the Δ_t value, so our only choice is to let the Cloud "Tag" the value of Δ_t to the data and thus forcing the underlying ABE scheme to be a Key-Policy one.

So, for a prototype implementation of our privacy-preserving cloud service with ephemeral data access (presented in the upcoming section 7), we chose a KP-ABE scheme that matches our needs for flexibility. We took as starting point the OpenABE KP-ABE scheme that we modified in order to enable Multi-Locking and ensure all possible optimizations are used. This constraints however touches only our "Cloud to User" communication.

On the other hand, this ABE scheme is not the best for the "User to Cloud" communication: considering the context of embedded devices and Internet of Things, ABE schemes that limit computation time and bandwidth are to consider. For example, Attrapadung's KP-ABE scheme with constant-sized Ciphertexts [ALdP11] (as described in section 2.3) is a interesting candidate. We can merge this scheme with the OpenABE KP-ABE scheme we modified because they are both part of p-Mul-MLF.

7 Implementation

In this section, we will present the two libraries we enhanced, then we will detail and justify the optimizations brought and discuss about our implementation results.

7.1 RELIC

RELIC [AG] is a low-level cryptographic library designed to realize efficient operations on Elliptic Curves. This library is written in C and assembly and is dual-licensed under Apache 2.0 and LGPL 2.1. A major advantage of this library is that it contains efficient state-of-the-art pairing algorithms which are useful to improve computation time of ABE schemes. As mentioned in section [CS10], we implement the Fixed Argument Optimization.

7.1.1 Optimal ate Pairing algorithm

The Optimal ate Pairing algorithm (shortened "oatep") [CS10] enables to realize pairing computations on Elliptic Curves with even embedding degrees. As it is the fastest pairing algorithm in RELIC library, we used it as the basis for the optimization. We now recall this algorithm:

Algorithm 1 Miller's affine double-and-add algorithm with denominator elimination

s **Input:** $R = (x_R, y_R), S = (x_S, y_S), m = (m_{l-1}...m_1, m_0)_2$.
Output: $f_{m,R}(S) \leftarrow f$
1: $T \leftarrow R, f \leftarrow 1$
2: **for** i from $l-2$ to 0 **do**
3: Compute $g(x, y) = y - y_T + \lambda(x_T - x)$, where λ is the gradient of the tangent line to T.
4: $T \leftarrow [2]T = [2](x_T, y_T)$.
5: $\boldsymbol{g \leftarrow g(x_S, y_S)}$.
6: $\boldsymbol{f \leftarrow f^2 \cdot g}$.
7: **if** $m_i \neq 0$ **then**
8: Compute $g(x, y) = y - y_T + \lambda(x_T - x)$, where λ is the gradient of the line joining T and R.
9: $T \leftarrow T + R$.
10: $\boldsymbol{g \leftarrow g(x_S, y_S)}$.
11: $\boldsymbol{f \leftarrow f \cdot g}$.
12: **end if**
13: **end for**
14: **return** f

As we can see, the second argument appears only after computation on the first argument (at line 5, 6, 10 and 11). The whole Fixed Argument Optimization is based on this particular repartition of the variables dependencies. Note that a good curve would have a low l for more efficiency.

7.1.2 Fixed Argument Optimization

The pairing optimization is realized using the optimization described in [CS10, SCA06]. The algorithm 2 computes the internal variables of the algorithm 1 depending only on the left-side argument. All the values of these variables are then stored for later reuse.

The second part of the split Miller's algorithm computes the pairing result using the precomputed internal variables obtained from algorithm 2 and the right-side argument. It is detailed in algorithm 3.

Algorithm 2 R-dependant precomputations

Input: $R = (x_R, y_R), m = (m_0, m_1, ..., m_{\#DBL-1}, m_{\#DBL})_2$.
Output: $G_{DBL} = \{(\lambda_1, c_1), (\lambda_2, c_2), ..., (\lambda_{\#DBL}, c_{\#DBL})\}$ and $G_{ADD} = \{(\lambda'_1, c'_1), (\lambda_2, c'_2), ..., (\lambda'_{\#DBL}, c'_{\#DBL})\}$

```
T ← R, G_DBL ← {∅}, G_ADD ← {∅}.
for i from 1 to #DBL do
    Compute λ_i and c_i, such that y + λ_i x + c_i is the line tangent to T.
    T ← [2]T.
    Append (λ_i, c_i) to G_DBL.
    if m_i ≠ 0 then
        Compute λ'_i and c'_i, such that y + λ'_i x + c'_i is the line joining T and R.
        T ← T + R.
        Append (λ'_i, c'_i) to G_ADD.
    end if
end for
return G_DBL, G_ADD
```

Algorithm 3 S-dependant computations

Input: $S = (x_S, y_S), m = (m_0, m_1, ..., m_{\#DBL-1}, m_{\#DBL})_2, G_{DBL}$ and G_{ADD} (from Algorithm 2).
Output: $f_{m,R}(S) \leftarrow f$.

```
f ← 1, count_ADD ← 1
for i from 1 to #DBL do
    Compute g ← (y_S + λ_i x_S + c_i)
    f ← f^2 · g
    if m_i ≠ 0 then
        Compute g ← (y_S + λ'_{count_ADD} x_S + c'_{count_ADD})
        count_ADD ← count_ADD + 1
        f ← f · g
    end if
end for
return f
```

7.2 OpenABE

OpenABE is a cryptographic library incorporating various Attribute-Based Encryption schemes. This library is developed by Zeutro in C++ for the core part and is available for use under the AGPL 3.0 license. In OpenABE, all low-level cryptographic operations are realized using a low-level cryptographic library. OpenABE supports two low-level cryptographic libraries, OpenSSL and RELIC. The library doing the elliptic curve operations is choosen at compilation time depending on the script options. Symmetric-key cryptographic operations are always carried out by OpenSSL. Since these two libraries support similar pairing algorithms, we choose to use RELIC, that provides the most efficient ones.

7.2.1 KP-ABE scheme

In order to validate our improvements and realize our cloud service, we focused our work on optimizing the KP-ABE scheme implemented in OpenABE. This scheme and all its properties are described in OpenABE's design document, we recall it here[3] :

Setup$(\tau, n) \rightarrow (MSK, PK)$: The setup algorithm takes as input the security parameter τ, creates the parameters for a bilinear group $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that p is a prime in $\Theta(2^\tau)$, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order p where g_1 generates $\mathbb{G}_1$, g_2 generates $\mathbb{G}_2$ and $e : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is an admissible bilinear map. Then it chooses a random exponent $y \in \mathbb{Z}_p$, and computes $Y = e(g_2, g_1)^y$.

In addition, we will use as a collision-resistant hash function $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^n$ that we model as a random oracle. The algorithm outputs the public parameters PK and the master secret key MSK as follows:

$$PK = \{g_1, g_2, e(g_2, g_1)^y\} \quad MSK = y$$

KeyGen$(PK, MSK, \mathcal{T}) \rightarrow SK$: Let $T : \{0,1\}^* \rightarrow \mathbb{G}_1$ be a function that we will model as a random oracle. The key generation algorithm outputs a private key which enables the user to decrypt a message encrypted under a set of attributes γ, if and only if $\mathcal{T}(\gamma) = 1$. The algorithm calculates the randomized shares of y according to the access structure $\mathcal{T}$. The following secret values are handed to the user for each leaf node x in the tree:

$$D_x = g_1^{\lambda_x} \cdot T(i)^{r_x} \text{ where } i = att(x) \quad d_x = g_2^{r_x}$$

Encrypt$(PK, \gamma; u) \rightarrow (Key, CT)$: The encryption algorithm takes as input the public parameters PK, a set of attributes γ, and an optional input seed $u \in \{0,1\}^k$ to a pseudo-random generator G. The algorithm first chooses a random $s \in \mathbb{Z}_p$ (if seed u is specified, then use G as source of randomness) and then returns the following:

$$Key \leftarrow H_1(e(g_2, g_1)^{ys}), \quad CT = (\gamma, E'' = g_2^s, \{E_i = T(i)^s\}_{i \in \gamma})$$

Decrypt$(CT, SK) \rightarrow Key$: The decryption algorithm takes as input the ciphertext CT and the user's private key SK which embeds the access structure $\mathcal{T}$. The algorithm first determines whether the access structure $\mathcal{T}$ satisfies the attributes γ on the ciphertext (e.g., if $\mathcal{T}(\gamma) = 1$). If so, then we proceed to recover the Lagrange coefficients ω for the minimum set of attributes S necessary to satisfy $\mathcal{T}$. Therefore,

[3] The ate pairing inputs groups are $(\mathbb{G}_2, \mathbb{G}_1)$ and not the usual ones $(\mathbb{G}_1, \mathbb{G}_2)$. We displayed the scheme in a way such that the precomputable element is always on the left side.

for each such attribute $i \in S$, the corresponding coefficient ω_i and the corresponding SK components in $\mathcal{T}$ (e.g., defined by $D_{\rho(i)}$ and $d_{\rho(i)}$), the algorithm first computes:

$$\prod_{i \in S} \left(\frac{e(E'', D_{\rho(i)})}{e(d_{\rho(i)}, E_i)} \right)^{\omega_i} =$$

$$\prod_{i \in S} \frac{e(g_2, g_1)^{\lambda_x \omega_i s} \cdot e(g_2, T(i))^{\omega_i r_x s}}{(g_2, T(i))^{\omega_i r_x s}} = \prod_{i \in S} e(g_2, g_1)^{\lambda_x \omega_i s} = e(g_2, g_1)^{ys}$$

The algorithm can then compute $Key = H_1(e(g_2, g_1)^{ys})$.

7.2.2 Multi-Locking support

The original form of the ciphertext in OpenABE's KP-ABE scheme is the following one:

$$Key \leftarrow H_1(e(g_2, g_1)^{ys}), \quad CT = (\gamma, E'' = g_2^s, \{E_i = T(i)^s\}_{i \in \gamma})$$

To protect against CCA-2 attack, ABE provides a Key Encapsulation Mechanism (KEM) for an AES key. The AES key encrypts and decrypts the data. This key is created by hashing the ABE lock.

Then, the aim of the ABE decryption is to reconstruct the lock and hash it to retrieve the symmetric key. This hashing prevents the scheme from being Multi-Lockable since the lock does not appear explicitly in the ciphertext, thus no re-encryption or partial decryption can be done.

We provide a construction that makes the encryption and decryption composition commutative, and thus makes the scheme multi-lockable. We will make the following Encrypt changes: first, pick a random $M \in \mathbb{G}_T$. Then, M is hashed to produce the symmetrical (AES-256 in OpenABE) encryption key: $Key \leftarrow H_1(M)$. Then, we multiply M with the lock $e(g_2, g_1)^{ys}$ and add it to the ciphertext, thus letting the scheme belong to p-mul-MLF:

$$Key \leftarrow H_1(M), \quad CT = (\gamma, E = M \cdot e(g_2, g_1)^{ys}, E'' = g_2^s, \{E_i = T(i)^s\}_{i \in \gamma})$$

The Decrypt as it is gives a reconstruction of $e(g_2, g_1)^{ys}$. We modify the output to be either $Key \leftarrow H_1(E \cdot e(g_2, g_1)^{-ys})$ in the case of the last decryption or $M' \leftarrow E \cdot e(g_2, g_1)^{-ys}$ for partial decryptions done during the multi-locking circuit.

In order to make Multi-Locking easy to implement, we added an option to KeyGen, Encrypt and Decrypt. This one allows the specification of a sub-universe by passing its name as an argument. By simply appending a universe specifier at the start of the name of an attribute, we allow an easy generation of keys where access rights are ensured to be sub-universe specific. This slight adaptation already mentionned in section 4.6 allows the Multi-Locking scheme to be intuitively created from its Lock-Key graph.

7.2.3 Perfect Argument Order

The original form of the Decrypt is the following one:

$$\prod_{i \in S} \left(\frac{e(E'', D_{\rho(i)})}{e(d_{\rho(i)}, E_i)} \right)^{\omega_i}$$

The pairing used in the scheme is Type III since the setup generate two different groups used as input groups.

In the numerator, we see that the left-side argument of the pairing does not depends of i, since $E'' = g_1^s$, so the pairing can be subjected to the Decryption Optimization. The right-side argument also depends on the ciphertext, we are in the case described in 5.1. Here, we should think that we should not Swich E'' and $D_{\rho(i)}$ since exponentiations are faster in $\mathbb{G}_1$ beacause group elements are two times smaller than $\mathbb{G}_2$ elements.

The OpenABE developers have already implemented the Decryption Optimization, so the Decrypt algorithm actually computes:

$$\frac{e(E'', \prod_{i\in S} D_{\rho(i)}^{\omega_i})}{\prod_{i\in S} e(d_{\rho(i)}, E_i)^{\omega_i}}$$

Then we still have to optimize the product of the remaining pairings which means computing it using the multi-pairing operation.

7.2.4 Optimizing Encrypt

In OpenABE, one multiplication per attribute is realized during Encrypt, and also one during the re-encryption that happens during Decrypt for CCA-2 security. This multiplication happens during the calculation of $T(i)$: we multiply a point of the curve issued from the hash of the attribute name by the cofactor of the curve. This operation is typical in Large Universe schemes. Studying the design of our Cloud service with ephemeral data access, we made some remarks:

- Some attributes are really common and come regularly attached to data (usual meta-data tags e.g “Video”, “GPS”, ...)
- A single user will have a preferred set of attributes to Encrypt with, as well as a set of attributes more often needed for decryption. The most common one in this category for example is the attribute corresponding to a user’s identity.
- Some attributes are hapax lagomena and appear only once. For example timestamps, if they are precise up to the second or millisecond, or other values with great (dates, GPS coordinates, ...).

Given this behaviour, that we expect applied to the vast majority of real-case scenarios, we can propose new trade-offs allowing to avoid the repeated calculation of $T(i)$, and providing better performances in both Encrypt and Decrypt than just accelerating pairings.

We propose for each of the three previous cases, respectively that:

- The Setup computes, once and for all, the point multiplications linked to the attributes that are most common for all users. The trusted authority can propose batches of precomputed attributes as optional parts of the public key. So all users can download specifically batches that concerns them.

  To implement this behaviour, we modify Setup: $Setup(\tau, n) \rightarrow (MSK, PK, V_{att})$ such that the trusted authority precomputes $T(atti)$ for all attributes $atti$ of a set δ of most common attributes into a vector:

  $$V_{att} \leftarrow \{T(x)\}_{x\in\delta}$$

- For further optimization, any user can also compute point multiplications linked to personal most-used attributes and store the results in his copy of V_{att} for later reuse. All following Encrypt and Decrypt using attributes in V_{att} will be sped-up.

To implement this change, we modify the Encrypt algorithm in order to enable it to use our stored attributes from V_{att}. $Encrypt(PK, \gamma, V_{att}; u) \rightarrow (Key, CT)$

To encrypt using a set of attributes γ, for each attribute $x \in \gamma$, the algorithm retrieves the point linked to an attribute if it is contained in the vector V_{att}, otherwise he computes $T(x)$.

Then the schemes continues as usual by an exponentiation with a randomly picked $s \in \mathbb{Z}_p$ and putting the result into the ciphertext:

$$Key \leftarrow H_1(e(g_2, g_1)^{ys}), \quad CT = (\gamma, E'' = g_2^s, \{E_i = E_{att_i}^s\}_{i \in \gamma})$$

- Hapax legomema (one time used) attributes cannot be precomputed, but they represent the smallest proportion of attributes. In our Cloud service, the only such attribute to calculate during Encrypt is the "Δ_t" attribute.

The stastically-driven factorization of calculation in these three levels of computation allows for a faster schemes with a compromise in memory space used of one elliptic point per attribute.

7.2.5 Optimizing Decrypt

During Decryption, in the denominator, to remove a lock or obtain the plaintext, we have to compute a number of exponentiations equal to the size of the minimum accepted set S of attributes:

$$\prod_{i \in S} D_{\rho(i)}^{\omega_i}$$

The possibility of swapping the arguments of the pairing can ensure us these exponentiations take place in the smallest possible group for efficiency. Since $D_{\rho(i)}$ belongs to $\mathbb{G}_1$, whose elements are twice shorter than elements of $\mathbb{G}_2$ we do not need the swap.

In the IoT context that motivates the construction of our privacy-preserving Cloud service with ephemeral data access, data will be sent to the cloud as multiple data streams coming from registered devices. Most commonly, the stream incomming from a specific device will almost always be ciphered using the same set of attributes.

Our last optimization, that we will call "Stream Optimization" consists to consider that the "stram" situation places us in the very small universe scenario described in 5.1. In this case, we can store during the first decryption of each stream the value of $\prod_{i \in S} D_{\rho(i)}^{\omega_i}$.

During decryption, we recover the Lagrange coefficients ω if the access structure $\mathcal{T}$ satisfies the attributes γ contained in the ciphertext. Those coefficients depend only of the access structure and the attributes. So once we succeeded to retreive the coefficients, for $x \in \gamma$, and computed the exponentiations, we store their product D into a vector that will be valid until a change of the access strucure:

$$V_D \leftarrow \{\prod_{i \in S_j} D_{\rho(i)}^{\omega_i}\}_j \text{ where j are the stream identifiers}$$

When the cloud receives a new ciphertext, if it belongs to an identified stream (already encountered set of attributes) the user can reuse the corresponding $\prod_{i \in S_j} D_{\rho(i)}^{\omega_i}$ previously calculated from the vector V_D.

Ultimately, because of this we now are in possession of pairing arguments before the reception of the ciphertext. On this behalf, this pairing can now be subjected to the Fixed Argument Optimization. If needed, the Switch Argument Method has to be used to allow the precomputation. If we decide to use it, we can change the storage V_D from storing points to storing precomputation data for later pairing calculation. This makes the data stream context, the one with the fastest possible decryption.

In recapitulation: in cases where the data comes from an identified stream, the computation time is reduced to the retrieving time of the precomputed data in V_D. This optimization was initially proposed for the cloud as the most obvious destination for streams of data ciphered with the same set of attributes.

However, another setting exists using this optimization: a user could want to download from the cloud all possible data ciphered with a given set of attributes (for example "Alice's data" and "videos"). This configuration creates the possibility for Alice to precompute $\prod_{i\in S_j} D_{\rho(i)}^{\omega_i}$ as a left-side pairing argument, in a advanced method to speed-up decryption of the stream coming from the cloud.

7.3 Fully Optimized KP-ABE scheme

We detail our full scheme:

Setup$(\tau, n) \rightarrow (MSK, PK, V_{att})$: The setup algorithm takes as input the security parameter τ, creates the parameters for a bilinear group $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that p is a prime in $\Theta(2^\tau)$, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order p where g_1 generates $\mathbb{G}_1$, g_2 generates $\mathbb{G}_2$ and $e : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is an admissible bilinear map. Then it chooses a random exponent $y \in \mathbb{Z}_p$, and computes $Y = e(g_2, g_1)^y$.

In addition, we will use as a collision-resistant hash function $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^n$ and $T : \{0,1\}^* \rightarrow \mathbb{Z}_p$ that we both model as random oracles. Let $T_1(i) = g_1^{T(i)}$ and $T_2(i) = g_2^{T(i)}$ The algorithm outputs the public parameters PK and the master secret key MSK as follows:

$$PK = \{g_1, g_2, e(g_2, g_1)^y\} \quad MSK = y \quad V_{att} \leftarrow \{T(x)\}_{x\in\delta}$$

The $T_1(i)$ values for the attributes that are most common for all users are precomputed and stored into a vector V_{att}.

KeyGen$(PK, MSK, \mathcal{T}) \rightarrow SK$: Let $T : \{0,1\}^* \rightarrow \mathbb{G}_1$ be a function that we will model as a random oracle. The key generation algorithm outputs a private key which enables the user to decrypt a message encrypted under a set of attributes γ, if and only if $\mathcal{T}(\gamma) = 1$. The algorithm calculates the randomized shares of y according to the access structure $\mathcal{T}$. The following secret values are handed to the user for each leaf node x in the tree:

$$D_x = g_1^{\lambda_x} \cdot T_2(i)^{r_x} \text{ where } i = att(x) \quad d_x = g_2^{r_x}$$

At the reception of its keys, the user generates the precomputation data from the d_{ρ_i} using algorithm 2 and stores it. Additionally the user computes and stores the $T_1(i)$ for all his most used attributes not already in V_{Att}.

Encrypt$(PK, \gamma, u) \rightarrow (Key, CT)$: The encryption algorithm takes as input the public parameters PK, a set of attributes γ, and an optional input seed $u \in \{0,1\}^k$ to a pseudo-random generator G. The algorithm first chooses a random $s \in \mathbb{Z}_p$ (if seed u is specified, then use G as source of randomness) and then returns the following:

$$Key \leftarrow H_1(M), \quad CT = (\gamma, E = M \cdot e(g_2, g_1)^{ys}, E'' = g_2^s, \{E_i = T_1(i)^s\}_{i\in\gamma})$$

Decrypt$(CT, SK) \to Key$: The decryption algorithm takes as input the ciphertext CT and the user's private key SK which embeds the access structure $\mathcal{T}$. The algorithm first determines whether the access structure $\mathcal{T}$ satisfies the attributes γ on the ciphertext (e.g., if $\mathcal{T}(\gamma) = 1$). If so, then we proceed to recover the Lagrange coefficients ω for the minimum set of attributes S necessary to satisfy $\mathcal{T}$. Therefore, for each such attribute $i \in S$, the corresponding coefficient ω_i and the corresponding SK components in $\mathcal{T}$ (e.g., defined by $D_{\rho(i)}$ and $d_{\rho(i)}$), if the ciphertext does not come from an identified stream, the algorithm compute

$$\frac{e(\prod_{i\in S} D_{\rho(i)}^{\omega_i}, E'')}{\prod_{i\in S} e(d_{\rho(i)}, E_i)^{\omega_i}}$$

If the ciphertext comes from the identified stream j, the algorithm computes:

$$\frac{e(V_{Dj}, E'')}{\prod_{i\in S} e(d_{\rho(i)}, E_i)^{\omega_i}}$$

The tho operations evaluate identically to:

$$= \prod_{i\in S} \frac{e(g_2, g_1)^{\lambda_x \omega_i s} \cdot e(g_2^{T(i)}, g_1)^{\omega_i r_x s}}{(g_2, g_1^{T(i)})^{\omega_i r_x s}} = \prod_{i\in S} e(g_2, g_1)^{\lambda_x \omega_i s} = e(g_2, g_1)^{ys}$$

The algorithm can then compute $Key = H_1(E/e(g_2, g_1)^{ys})$.

7.4 Results and benchmarks

Experiments were executed on a Desktop computer equipped with a 64-bit Intel(R) Core(TM) i7-4770 CPU, based on Haswell microarchitecture, at a frequency of 3.40 GHz and a maximal frequency of 3.90 GHz. The Operating System on which the experiments were realized is Ubuntu 16.04.6 LTS. The Desktop computer has a RAM of 16 GB, with a L1d cache of 32 KB, a L1i cache of 32 KB, a L2 cache of 256 KB and a L3 cache of 8192 KB.

In the following, we present the experiments and the results on RELIC and then on OpenABE.

7.4.1 RELIC

We compare the execution time of our implementation of the Fixed Argument Optimization based on the multi Optimal ate pairing with Costello and Stebila's original implementation of the multi Optimal ate pairing. We also evaluate the memory consumption of the pairing precomputations.

Experiments were carried out by varying the number of pairings based on a logarithmic scale in base 2, from 1 to 8,192 pairings. For each number of pairings we realized 128 computations, then removed the outliers and computed the average.

Figure 4 represents the speed-up percentage between the execution of the RELIC multi Optimal ate pairing and our implementation of the Fixed Argument Optimization using the precomputations.

Results show that a single pairing has a speed-up percentage of about 11.6 %. The speed-up grows consistently until 32 pairings are reached with a speed-up of about 31.6 %. After that number of pairings, the speed-up varies around a 32 % rate.

Those results confirms Costello and Stebila's results in [CS10].

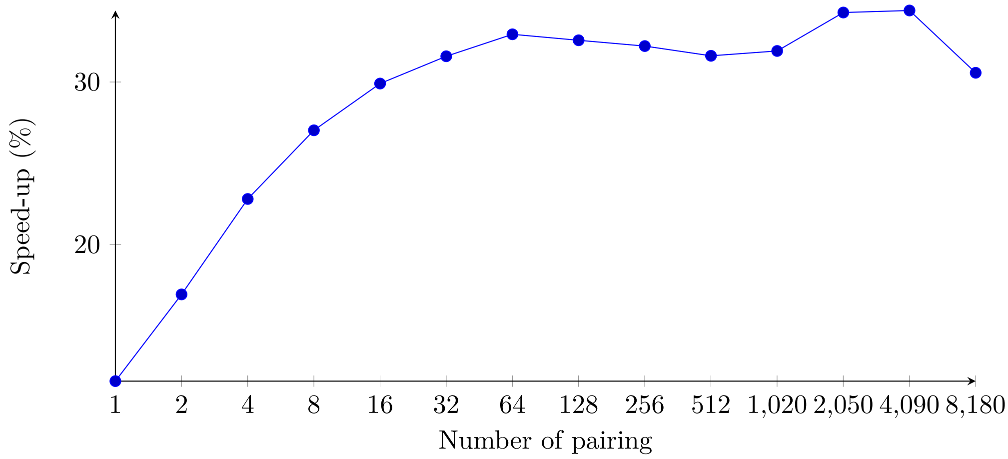

Figure 4: Speed-up of the multi Optimal ate pairing with the Fixed Argument Optimization over the original multi Optimal ate pairing

Table 2 represents the execution time (in ms) of RELIC's moatep and the corresponding execution times of our pairing precomputation (col. 3) and pairing computation (col. 4) using the precomputed data as a percentage of RELIC's execution time, and the memory consumption in kB of the precomputations (col.5).

Table 2: Evaluation of the memory consumption of the Fixed Argument Optimization and comparison of the execution time of the optimization with the original pairing

#pairings	moatep (ms)	moatep w/ precomp (%)		Memory (kB)
	w/o precomp	precomp	pairing	
1	1.85	11.85	88.40	88
2	2.48	17.81	83.07	176
4	3.75	24.10	77.19	353
8	6.27	29.20	72.97	705
16	11.30	32.54	70.10	1411
32	21.37	34.50	68.42	2823
64	41.77	35.44	67.06	5645
128	81.77	36.44	67.43	11291
256	162.59	37.08	67.79	22581
512	323.42	37.72	68.39	45163
1024	647.08	37.95	68.09	90325
2048	1336.32	36.97	65.73	180650
4096	2691.23	37.48	65.60	361300
8192	5325.37	39.40	69.43	722600

As expected, memory consumption grows linearly with the number of pairings precomputations stored. The theoretic memory consumption is expressed using the following function: $f(x) = x \cdot l \cdot (\|\mathbb{G}_2\| + \|dv_2\| + 7 \cdot \|\mathbb{F}_{p^2}\|)$ with x being the number of pairings and l being the number of Miller's loops precomputed (Algorithm 1).

The Elliptic Curve used for computation is the curve BN-256. One element of $\mathbb{G}_2$ is 1600-bits long. One element of dv_2 is 4352-bits long. One element of $\mathbb{F}_{p^2}$ is 512-bits long. The miller loop variable length l (in algorithm 1) is 74. Finally, the real memory consumption is expressed with the following function: $f(x) = x \cdot 74 \cdot (1600 + 4352 + 7 \cdot 512) = 705664 \cdot x$ bits.

By summing the two last columns, we can see that total percentage exceeds 100 %. For 1 pairing we exceed this value by 0.25 %, and for 8,192 pairing we exceed it by 8.83 %. This is due to the added time for storing and retrieving the precomputed variables.

Our implementation of the Fixed Argument Optimization performs better than the original pairing computation except if we use the precomputation only once. As soon as the second pairing computation, the percentage of calculation represent 94.3 % of the original pairing, but this optimization should not be used if the two arguments of the pairing are expected to come simultaneously.

7.4.2 OpenABE

We compare the execution time of the different algorithms of our Fully Optimized KP-ABE scheme with the implementation of the original KP-ABE in OpenABE. This comparison is carried out in two settings: a classical source-target scheme and a tripartite Multi-Locking scheme (with one relay).

Experiments were carried out by varying the number of attributes based on a logarithmic scale in base 2 from 1 to 4,096 attributes. For each number of attributes we realized 128 computations, then removed the outliers and computed the average. All experiments were realized by encrypting a random plaintext of 4,096 Bytes.

Asymmetric encryption usage Experiments were realized by computing the following algorithms: Setup, KeyGen, Encrypt and Decrypt. We executed all the four algorithms to get the execution times for the original KP-ABE scheme in the white columns. For the experiments on the Fully Optimized KP-ABE scheme, we executed the Setup and KeyGen, then the Encrypt and Decrypt two times each.

The first grey Encrypt and Decrypt column represent the start of a data stream, that do not yet benefit from already existing values in the V_D vector, they also store the new values in the vector.

The second ones represent the Fully Optimized KP-ABE at its best, with all possible precomputed variables already in the memory.

The Elliptic Curve used for computation is the curve BN-254, so the size of one element of $\mathbb{G}_2$ is 1600-bits. One element of dv_2 is 1280-bits long. One element of $\mathbb{F}_{p^2}$ is 512-bits long. The miller loop variable length l (Algorithm 1) is 70. Finally, the real memory consumption is expressed with following function: $f(x) = x \cdot 70 \cdot (1600 + 1280 + 7 \cdot 512) = 452480 \cdot x$ bits.

The results of our Fully Optimized scheme are the following ones:

- The Setup grows linearly with the number of attributes since hashed of the attributes are precomputed in this algorithm but this overhead is acceptable since it is an algorithm executed once in the system.

- The KeyGen takes more time to compute since the we count the precomputation part of the pairing in it, this algorithm will be executed each time a user joins the system and does precomputation based on a decryption key.

- The Encrypt takes less computation time since we used the precompute $T_1(i)$ from the Setup.

- Finally the first Decrypt is in the same order of magnitude as the Decrypt of the original KP-ABE scheme since it does not take into account the Stream Optimization. The second Decrypt takes less computation time since it used the V_D precomputations.

Table 3: Evaluation of the memory consumption of the optimization in RELIC and comparison of the execution time of the optimization with the original one

#att	Setup		KeyGen		Encrypt			Decrypt		
1	1.77	1.32	0.60	0.74	0.78	0.58	0.54	1.55	1.35	1.31
2	1.71	1.42	1.08	1.46	0.82	0.68	0.64	1.81	1.63	1.53
4	1.79	1.62	2.09	2.87	1.04	0.88	0.84	2.34	2.17	1.95
8	1.83	2.02	3.81	5.68	1.49	1.27	1.23	3.44	3.43	2.77
16	1.80	2.81	6.92	11.33	2.41	2.06	2.02	5.77	5.92	4.44
32	1.74	4.39	12.84	22.57	4.31	3.64	3.58	10.62	10.81	7.88
64	1.77	7.55	25.27	45.00	8.11	6.78	6.73	20.92	20.40	15.17
128	1.75	13.88	49.55	89.85	15.67	13.12	13.10	44.34	47.05	31.19
256	1.85	26.45	98.05	179.73	31.01	25.94	25.92	100.77	105.12	68.96
512	1.78	51.58	195.08	359.32	62.20	52.22	52.12	218.03	226.55	149.69
1024	1.64	101.68	389.03	719.57	127.72	107.52	107.27	456.90	469.29	308.26
2048	1.64	201.88	777.24	1440.04	268.01	227.28	227.26	943.23	960.85	643.30
4096	1.43	402.18	1554.45	2897.77	590.37	511.56	508.54	1969.09	2033.50	1363.45

Multi-Locking Experiments were realized by enforcing the scheme in Figure 5. We executed the following algorithms: Setup, KeyGen for the tree sub-universes Source-Relay, Source-Target and Relay-Target. Then, a first Encrypt for the Source-Target lock, a second Encrypt for the Source-Relay lock, a first Decrypt with Source-Relay keys, a last Encrypt for the Relay-Target lock, a second Decrypt with the Source-Relay keys and a final Decrypt using Source-Target Keys.

The access policy is a $\wedge$ over all attributes attached to the ciphertext. So the LSSS has a line for each attribute positioned.

This lock key graph is similar to the one in our example in section 6. Since this Lock-Key graph is planar, it can also be expressed with a scheme realizing over-encryption.

Indeed, we executed all the algorithms to compute the execution time for the original KP-ABE scheme. Like the previous experiment, we executed a first time all the algorithms with our Fully Optimized KP-ABE scheme. Then we executed all the Encrypt and Decrypt a second time to benefit of the stream optimization.

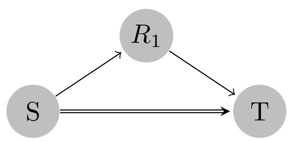

Figure 5: A Lock-key graph with 1 relay.

Figure 6 represents the comparison of standard over-encryption in OpenABE's KPABE and Multi-Locking with our Fully Optimized KP-ABE. The full line represents OpenABE times, and the dotted line represents our scheme when benefiting from all our improvements including Stream Optimization. The comparison of the two curves highlight the efficiency of our improvement.

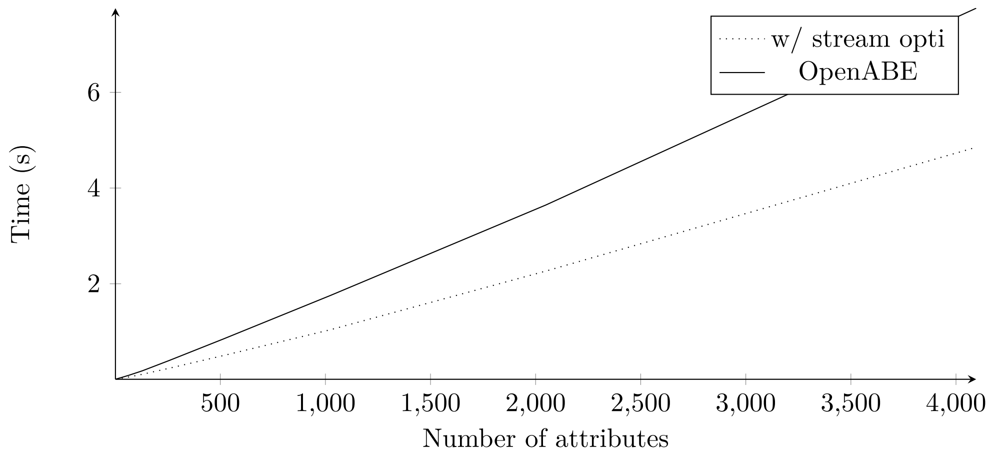

Figure 6: Advantage of Multi-Locking over super-encryption using ABE

8 Conclusion and future work

In this paper, we presented two major improvements of ABE. The Perfect Argument Order Optimization based on the Switch Argument method allows to apply the Fixed Argument Optimization to all pairings in a ABE scheme end thus speeding them up by 30%. We open new horizons on the construction of ABE schemes by introducing Multi-Locking families, allowing a better fit for a larger range of real-case scenarios. We checked that our improvements were applicable to nearly all schemes among a large survey. We benefited from the combination of a Large Universe KP-ABE and a Constant-Size Ciphertext KP-ABE to create a Cloud service allowing time-based access policies not relying on the decay or revocation of keys that also allows for the possibility to delegate access control to trusted relays. We implemented a model for such a device and studied its performances to demonstrate the efficiency of our optimizations.

Some aspects, notably of Multi-Locking can be improved: how can one be sure of the trustworthiness of its relays? Is it possible to design an ABE scheme with a traitor tracing or watermarking mechanism? When studying the scope of multi-lockable ABE schemes, we ignored post-quantum, lattice-based schemes. Is there a way to implement multi-locking on these schemes? We believe the framework we developed can provide ABE constructions fitting a wide variety of scenarios. For future work we would like to explore more deeply the impact on privacy in such scenarios.

References

[AG] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[AHL+12] Nuttapong Attrapadung, Javier Herranz, Fabien Laguillaumie, Benoît Libert, Elie De Panafieu, and Carla Ràfols. Attribute-Based Encryption Schemes with Constant-Size Ciphertexts. *Theoretical Computer Science*, 422:15–38, 2012.

[ALdP11] N. Attrapadung, B. Libert, and E. de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. *PKC 2011 LNCS 6571*, pages 90–108, 2011.

[AS15] Ram Govind Singh Anurag Singh. Various attacks over the elliptic curve-based cryptosystems. *International Journal of Engineering and Innovative Technology*, 5:50–52, 11 2015.

[B+98] Dan Boneh et al. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1998.

[BBD+08] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, and Francisco Rodríguez-Henríquez. A comparison between hardware accelerators for the modified tate pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, pages 297–315, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BSSV09] Vladimir Božović, Daniel Socek, Rainer Steinwandt, and Viktoria Villanyi. Multi-authority attribute-based encryption with honest-but-curious central authority. *IACR Cryptology ePrint Archive*, 2009:83, 01 2009.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, Berkeley, France, May 2007. IEEE.

[CC09] Melissa Chase and Sherman S.M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 121–130, New York, NY, USA, 2009. ACM.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, pages 515–534, Berlin, Heidelberg, 2007. Springer-Verlag.

[CS10] Craig Costello and Douglas Stebila. Fixed argument pairings. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *Progress in Cryptology – LATINCRYPT 2010*, pages 92–108, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CWM+13] Yong Cheng, Zhi-ying Wang, Jun Ma, Jiang-jiang Wu, Song-zhu Mei, and Jiang-chun Ren. Efficient revocation in ciphertext-policy attribute-based encryption based cryptographic cloud storage. *Journal of Zhejiang University SCIENCE C*, 14, 02 2013.

[GPSW06a] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 89–98, 01 2006.

[GPSW06b] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[HJSNS08] M Hinek, Shaoquan Jiang, Reihaneh Safavi-Naini, and Siamak Shahandashti. Attribute-based encryption with key cloning protection. *IACR Cryptology ePrint Archive*, 2008, 12 2008.

[HL10] M Jason Hinek and Charles CY Lam. Common modulus attacks on small private exponent rsa and some fast variants (in practice). *Journal of Mathematical Cryptology*, 4(1):58–93, 2010.

[HSMY12] Jinguang Han, Willy Susilo, Yi Mu, and Jun Yan. Privacy-preserving decentralized key-policy attribute-based encryption. *Parallel and Distributed Systems, IEEE Transactions on*, 23:2150–2162, 11 2012.

[Hur13] Junbeom Hur. Improving security and efficiency in attribute-based data sharing. *IEEE Transactions on Knowledge and Data Engineering*, 25:2271–2282, 2013.

[HW13] Susan Hohenberger and Brent Waters. Attribute-based encryption with fast decryption. In *International Workshop on Public Key Cryptography*, pages 162–179. Springer, 2013.

[LCH$^+$11] Zhen Liu, Zhenfu Cao, Qiong Huang, Duncan S. Wong, and Tsz Hon Yuen. Fully secure multi-authority ciphertext-policy attribute-based encryption without random oracles. In *Proceedings of the 16th European Conference on Research in Computer Security*, ESORICS'11, pages 278–297, Berlin, Heidelberg, 2011. Springer-Verlag.

[LCLS10] Huang Lin, Zhenfu Cao, Xiaohui Liang, and Jun Shao. Secure threshold multi authority attribute based encryption without a central authority. *Information Sciences*, 180(13):2618 – 2632, 2010.

[LCW10] Zhen Liu, Zhenfu Cao, and Duncan S. Wong. Efficient generation of linear secret sharing scheme matrices from threshold access trees. Cryptology ePrint Archive, Report 2010/374, 2010.

[LHC$^+$11] Jin Li, Qiong Huang, Xiaofeng Chen, Sherman Chow, Duncan Wong, and Dongqing Xie. Multi-authority ciphertext-policy attribute-based encryption with accountability. In *Proceedings of the 6th ACM Symposium on information, COmputer and COmmunications Security*, pages 386–390, 01 2011.

[LLLS10] Xiaohui Liang, Rongxing Lu, Xiaodong Lin, and Xuemin Sherman Shen. Ciphertext policy attribute based encryption with efficient revocation. *TechnicalReport, University of Waterloo, 2, 8*, 2010.

[LOS$^+$10] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 62–91, Berlin, Heidelberg, 2010. Springer-Verlag.

[Lyn07] Ben Lynn. *ON THE IMPLEMENTATION OF PAIRING-BASED CRYPTOSYSTEMS*. PhD thesis, STANFORD UNIVERSITY, June 2007.

[Mil86] Victor S. Miller. Short programs for functions on curves. In *IBM THOMAS J. WATSON RESEARCH CENTER*, 1986.

[MKE08] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert. Distributed attribute-based encryption. In *ICISC*, 2008.

[MKE09] Sascha Mueller, Stefan Katzenbeisser, and Claudia Eckert. On multi-authority ciphertext-policy attribute-based encryption. *Fraunhofer SIT*, 46, 07 2009.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. *IACR Cryptology ePrint Archive*, 2007:323, 01 2007.

[PSA18] Praveen Kumar P., Syam Kumar P., and Alphonse P.J.A. Attribute based encryption in cloud computing: A survey, gap analysis, and future directions. *Journal of Network and Computer Applications*, 108:37–52, 2018.

[PTMW10a] Matthew Pirretti, Patrick Traynor, Patrick Mcdaniel, and Brent Waters. Secure attribute-based systems. *J. Comput. Secur.*, 18(5):799–837, September 2010.

[PTMW10b] Matthew Pirretti, Patrick Traynor, Patrick Drew McDaniel, and Brent Waters. Secure attribute-based systems. *Journal of Computer Security*, 18(5):799–837, 10 2010.

[QLDJ14] Zhi Qiao, Shuwen Liang, Spencer Davis, and Hai Jiang. Survey of attribute based encryption. *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6, 2014.

[SA05] Waters B. Sahai A. Fuzzy identity-based encryption. *Cramer R. (eds) Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science, vol 3494. Springer, Berlin, Heidelberg*, 2005.

[SCA06] Michael Scott, Neil Costigan, and Wesam Abdulwahab. Implementing cryptographic pairings on smartcards. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 134–147, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Sco11] Michael Scott. On the efficient implementation of pairing-based protocols. In Liqun Chen, editor, *Cryptography and Coding*, pages 296–308, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. *Advances in Cryptology*, pages 47–53, 1985.

[SIM83] GUSTAVUS J. SIMMONS. A “weak” privacy protocol using the rsa crypto algorithm. *Cryptologia*, 7(2):180–182, 1983.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption:an expressive, efficient, and provably secure realization. *Catalano D., Fazio N., Gennaro R., Nicolosi A. (eds) Public Key Cryptography – PKC 2011. PKC 2011. Lecture Notes in Computer Science, vol 6571. Springer, Berlin, Heidelberg*, 2011.

[WLW10] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS ’10, pages 735–737, New York, NY, USA, 2010. ACM.

[WLWG11] Guojun Wang, Qin Liu, Jie Wu, and Minyi Guo. Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers. *Computers & Security*, 30(5):320 – 331, 2011. Advances in network and system security.