SPADE: Digging into Selective and PArtial DEcryption using Functional Encryption*

Camille Nuoskala[1], Hossein Abdinasibfar[1], and Antonis Michalas[1,2]

[1] Tampere University, Tampere, Finland
[2] RISE Research Institutes of Sweden, Gothenburg, Sweden

Abstract. Functional Encryption (FE) is a cryptographic technique established to guarantee data privacy while allowing the retrieval of specific results from the data. While traditional decryption methods rely on a secret key disclosing all the data, FE introduces a more subtle approach. The key generation algorithm generates function-specific decryption keys that can be adaptively provided based on policies. Adaptive access control is a good feature for privacy-preserving techniques. Generic schemes have been designed to run basic functions, such as linear regression. However, they often provide a narrow set of outputs, resulting in a lack of thorough analysis. The bottom line is that despite significant research, FE still requires appropriate constructions to unleash its full potential in securely analyzing data and providing more insights. In this article, we introduce SPADE – a novel FE scheme that features multiple users and offers fine-grained access control through partial decryption of the ciphertexts. Unlike existing FE schemes, our construction also supports qualitative data, such as genomics, expanding the applications of privacy-preserving analysis to enable a comprehensive study of the data. SPADE is a significant advancement that balances privacy and data analysis with clear implications in healthcare and finance. To verify its applicability, we conducted extensive experiments on datasets used in sleep medicine (hypnogram data) and DNA analysis (genomic records).

Keywords: Access Control · Data Privacy · Functional Encryption · Genomic Analysis · Partial Decryption · Sleep Analysis

1 Introduction

Although data has always been important to society, its current role is more crucial than ever before. Today, every aspect of human life is fueled with data. When it comes to modern civilization, data is more like the air we breathe than the oil we burn. However, the ability to collect huge amounts of data is a double-edged sword. On the one hand, it empowers governments, businesses, and individuals to make better decisions. On the other hand, it raises big questions about how we can use that data while protecting people's right to privacy.

*This work was funded by the HARPOCRATES EU research project (No. 101069535).

However, as we have already seen multiple times, privacy is a large and very nebulous target that is hard to approach directly. The main reason for this lies in the fact that *if data is collected, it can always be abused.* Nevertheless, with the development of several advanced cryptographic schemes that allow operations on encrypted data, such as Homomorphic Encryption (HE) [24,13] and Functional Encryption (FE) [8], we are not entirely outside the realm of possibility in creating genuine privacy-respecting services.

FE enables the encryption of sensitive data under a master public key and allows an analyst to learn about the results of a specific function by holding the corresponding decryption key. This technique is based on creating decryption keys for particular functions, which, in contrast to HE, prevent access to undesired computations, thus opening up various new applications in the field of encrypted data operations. However, despite the apparent potential of FE, research has been focused on designing standard schemes for a limited range of functionality. For instance, a well-studied and popular FE construction, namely Inner Product Functional Encryption (IPFE), consists of computing the inner product, i.e., a weighted addition, on the entries of a ciphertext. Eventually, while a plethora of FE schemes support the inner product [4,9,11,20,6], no construction has been proposed, to the best of our knowledge, to handle the encryption and management of qualitative data.

This work tries to overcome this issue by creating SPADE – a construction that allows an analyst to perform qualitative analysis on encrypted data. Hence, being able to build richer ways of capturing knowledge in a privacy-preserving way. To do so, we distinguish two types of data: quantitative and qualitative. The first refers to data that can be measured as numbers, such as revenue, grades, and percentages. These data are practical as they support addition, multiplication, and inner product. On the other hand, qualitative data, if they can be represented as numbers, do not support this kind of operation. They mostly include medical data, such as DNA, Glasgow Coma Scale, and Hypnograms, where addition and multiplication can be irrelevant when analyzing the collected information.

To test our approach and verify the applicability of SPADE in qualitative analysis, we take into consideration the following two use cases that are widely used in medical research: *(1)* Analysis of Hypnogram Data for sleep medicine and *(2)* Analysis of Genomic Records. In the first use case, we utilized hypnograms, which are recordings of the different stages of sleep, and they are qualitative: *awaken state*, *deep sleep*, *abnormal sleep*, etc. Similarly, for the second use case, genomic records made from DNA sequences have been used, where each DNA sequence is represented by very long lines of characters corresponding to the four nucleotides: *adenine (A)*, *cytosine (C)*, *guanine (G)*, and *thymine (T)*. So, SPADE instead of just calculating a typical numerical function (e.g. addition) over encrypted data, it is capable of counting and locating the number of occurrences of a given value in a dataset, like the number of occurrences of one particular basis in a DNA sequence – a process that is widely used when analyzing medical data.

It is important to note that, unlike most generic FE constructions, SPADE can be used on both quantitative and qualitative data. This has the potential to solve the confusing problem of what information you can and cannot share because of patient privacy. In most cases, there are no easily available tools or technology that facilitate broad data-sharing and access while at the same time providing certain guarantees about the privacy of the users. SPADEhas the potential to simplify how researchers and data scientists around the world work together, share, and analyze data in order to make advances in their field.

Contribution Our contribution can be summarized as follows:

C1. We formally define SPADE, a scheme that permits partial decryption of both qualitative and quantitative encrypted data in a multi-user setup. Additionally, we propose a system model for our scheme that fits realistic scenarios.
C2. We extend our construction with a decentralized variant that does not rely on a trusted authority, permitting completely autonomous management of the keys by the users themselves.
C3. To illustrate the applicability of SPADE, we provide an open-source implementation along with experiments and benchmarks for two different use cases, namely Analysis of Hypnogram Data and Analysis of Genomic Records.

2 Related Work

Functional Encryption FE was introduced by Boneh *et al.* in [8] as a generalization of Public-Key Encryption (PKE). This primitive has subsequently been studied and applied to different approaches [15,14,29]. As its name suggests, FE constructions are usually narrowed to specific functionalities for specific use cases. For the last few years, one of the most studied FE techniques has been IPFE. In terms of data analysis, this functionality is indeed particularly relevant for linear regression models or other Privacy privacy-preserving machine learning (PPML) techniques [25,18,21]. As a follow-up to IPFE, Quadratic Functional Encryption (QFE) [25,28] is another emerging technique that aims at evaluating both addition and multiplication. Their implementation in Go language through the library GoFE [1] demonstrates the applicability of these schemes and contributes to the craze for these techniques.

While these constructions are very promising, they are limited to standard operations. However, data analysis often requires more complex and specific functions, and it is important to broaden the range of functions that can be used. Even though they can be represented as integers, genomics data are qualitative; hence, implementing IPFE or QFE does not make sense. To the best of our knowledge, our article proposes the first FE-based protocol addressing this problem, exploring a new scope of capabilities offered by FE.

Partial Decryption and Qualitative Data In this article, we mainly focus on what we call partial decryption, that is, decrypting only part of the ciphertext. While the field of FE does not properly address this question, partial decryption

has been studied in other fields. For example, the study of genomics data [16,5,17] requires techniques identifying specific patterns or behaviours in a ciphertext. To achieve the problem of count query, Hasan *et al.* approach [16] is to browse an encrypted tree using the Paillier cryptosystem. As for Ayday *et al.* [5], they essentially rely on permutations. In short, none of these approaches uses FE and its assets in terms of key management.

3 Background

This section contains the tools used in our protocols. We start by providing the notation employed in this article and present the mathematical concepts necessary for understanding our constructions. Finally, we define the cryptographic and security notions on which our protocols rely.

Notation For a positive integer n, $[n]$ is the set $\{1, \ldots, n\}$. Vectors are denoted by bold lowercase letters. The inner product of two vectors $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_n)$ is $\langle \mathbf{x}, \mathbf{y} \rangle = x_1 y_1 + \ldots + x_n y_n$. We denote $y \xleftarrow{\$} \mathcal{Y}$ when y is chosen uniformly at random from a set $\mathcal{Y}$. For $\mathcal{Y} \subset \mathbb{R}$, $\mathcal{Y}^* := \mathcal{Y} - \{0\}$. $\mathcal{A}$ is said to be a PPT (*probabilistic polynomial time*) adversary if it is a randomized algorithm such that there exists a polynomial $p(y)$ such that for any input y, the running time of $\mathcal{A}(y)$ is bounded by $|p(y)|$. For $a \in \mathbb{N}$ and $g \in \mathbb{G}$, $[a]_g$ denotes the exponentiation g^a in $\mathbb{G}$. Two elements a, b are said to be indistinguishable, denoted $a \sim b$, if $\mathsf{Pr}[\beta' = \beta \,;\, c \leftarrow \beta \cdot a + (1-\beta) \cdot b \,|\, \beta \xleftarrow{\$} \{0,1\}] = 1/2 + negl(\cdot)$. A function $f : \mathbb{N} \mapsto \mathbb{R}$ is negligible if $\forall c \in \mathbb{N}, \exists \varepsilon_0 \in \mathbb{N}$ such that $\forall \varepsilon \geq \varepsilon_0$, $f(\varepsilon) < \varepsilon^{-c}$. An arbitrary negligible function is denoted $negl(\cdot)$.

Definition 1 (Cyclic Group). *A cyclic (multiplicative) group is a finite group $\mathbb{G}$ generated by a single element $g \in \mathbb{G}$. g is called generator of $\mathbb{G}$ and we denote $\mathbb{G} = \langle g \rangle$.*

For the sake of readability, we denote exponentiation in a cyclic group with brackets. For $a \in \mathbb{N}$ and $g \in \mathbb{G}$, $[a]_g = g^a$.

Definition 2 (Fermat number). *A Fermat number is a prime number of the form $2^{2^k} + 1$, where $k \in \mathbb{N}$.*

Note that any prime number of the form $2^n + 1$ for $n \in \mathbb{N}^*$ is a Fermat number. Moreover, there is an infinite number of Fermat numbers. The proof of these statements is omitted here as it goes beyond the scope of this article.

Theorem 1 (Lagrange's theorem). *Let $\mathbb{G}$ be a finite group of cardinal n and neutral element 1. For an element $g \in \mathbb{G}$, call order of g the smallest $k \in \mathbb{Z}^*$ such that $g^k = 1$. Lagrange's theorem states that the order of any element of $\mathbb{G}$ divides n. It follows that if $\mathbb{G}$ is a cyclic group of generator g, the order of g equals n.*

We omit the proof of Lagrange's theorem, as it is a well-known result in group theory.

3.1 Functional Encryption

We now move on with the definition of FE – the core cryptographic primitive this paper relies on. We consider a secret key setup where the encryption is performed using a secret key, also called the private key or identifier. For the rest of this article, we will refer to it as α or α_j.

Definition 3 (Multi-Client Functional Encryption (MCFE)). *Let $\mathcal{M}$ and $\mathcal{C}$ be the message and ciphertext spaces, respectively. Denote $\mathcal{F}$ the set of functionalities supported. A Multi-Client (Secret-Key) Functional Encryption scheme $\mathsf{MCFE}_{\mathcal{M},\mathcal{C},\mathcal{F}}$ is a tuple of four algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KDer}, \mathsf{Dec})$ defined as follow:*

- $\mathsf{Setup}\left(1^\lambda\right)$: *The setup algorithm is a probabilistic algorithm that takes as input a security parameter λ and outputs the master secret key msk and the master public key mpk;*
- $\mathsf{Enc}\left(\mathsf{mpk}, \mathbf{x}, \alpha\right)$: *The encryption algorithm is a probabilistic algorithm that takes as input the master public key mpk, a message $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{M}$ and a private identifier α to output a ciphertext $\mathbf{c} = (c_1, \ldots, c_n) \in \mathcal{C}$;*
- $\mathsf{KDer}\left(\mathsf{msk}, f, \alpha\right)$: *The key derivation algorithm is a deterministic algorithm that takes as input the master secret key msk, a function $f \in \mathcal{F}$ and a private identifier α to output the decryption key dk_f;*
- $\mathsf{Dec}\left(\mathsf{dk}_f, \mathbf{c}\right)$: *The decryption algorithm is a deterministic algorithm that takes as input a decryption key dk_f and a well-defined ciphertext $\mathbf{c} = \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \alpha) \in \mathcal{C}$ and outputs $f(x_1, \ldots, x_n)$ if dk_f and $\mathbf{c}$ are generated using the same identifier α, and $\perp$ otherwise.*

Correctness MCFE *is correct, that is:*

$$\mathbb{P}[\mathsf{Dec}\left(\mathsf{dk}_f, \mathbf{c}\right) \neq f(x_1, \ldots, x_n) \,\|\, (\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}\left(1^\lambda\right) \\ \mathbf{c} \leftarrow \mathsf{Enc}\left(\mathsf{mpk}, \mathbf{x}, \alpha\right) \; ; \; \mathsf{dk}_f \leftarrow \mathsf{KDer}\left(\mathsf{msk}, f, \alpha\right)]$$

is negligible.

4 Core construction of SPADE

In this section, we present the algorithms that constitute the construction of SPADE. We rely on the MCFE construction defined in definition 3.

Parameters The space of messages is $\mathcal{M} = \mathbb{Z}_t^n$ where $n > 0$ is the number of entries and $t > 0$ their range. For a power of prime $q > t + 1$, the space of secret keys $\mathsf{msk} = (s_1, \ldots, s_n)$ is $\mathbb{Z}_q^n$. The corresponding public key $\mathsf{mpk} = ([s_1]_g, \ldots, [s_n]_g)$ lies in $\mathbb{G}^n$ where $\mathbb{G}$ is a (cyclic) group of order q. Additionally, we denote $\alpha_j \in \mathbb{Z}_q$ the secret identifier of a user u_j. In this section we do not specify the parameters t and q but in practice we take q a Fermat number, $t < q-1$ and $\mathcal{S} = \{2\,k+1, k \in \mathbb{Z}_{\lfloor q/2 \rfloor - 1}\}$ the set of odd integers in $\mathbb{Z}_q$, as detailed in subsection 4.2.

Following, we introduce the entities of our system model to describe the main construction. The full system model, including the security assumptions, is detailed in section 5.

System model's entities The model we consider consists of three entities: *(i)* Key Curator (**KC**), *(ii)* Users (**U**) and *(iii)* Data Analyst (**DA**). The role, capabilities, and trust level of the different parties are detailed in section 5.

4.1 Algorithms

First of all, **KC** generates the master secret key and master public key $(\mathsf{msk}, \mathsf{mpk})$ through the setup algorithm 1. Every user u_j who wants to register samples an element $\alpha_j \xleftarrow{\$} \mathbb{Z}_q$ and sends a registration request containing $[\alpha_j]_g$ to **KC**. The latter stores these values to generate the decryption keys as detailed in algorithm 3.

Algorithm 1 $\mathsf{Setup}\left(1^\lambda, n\right)$

1: Generate the master secret key $\mathsf{msk} = (s_1, \ldots, s_n) \xleftarrow{\$} \mathbb{Z}_q^n$
2: Compute the master public key $\mathsf{mpk} \leftarrow ([s_1]_g, \ldots, [s_n]_g)$
3: **for** each user u_j **do**
4: Generate $\alpha_j \xleftarrow{\$} \mathbb{Z}_q$ and send $[\alpha_j]_g$ to **KC**

A user u_j encrypts a message $\mathbf{x} = (x_1, \ldots, x_n)$ using both the master public key mpk and her private key g^{α_j}, as described in algorithm 2. To do so, she first generates a n-tuple of random integers, or noise, sampled in a subset $\mathcal{S}$ of $\mathbb{Z}_q$. The specification of $\mathcal{S}$ is detailed in subsection 4.2. Then, for each entry x_i, she produces both a *helping information* for the decryption $h_i = [\alpha_j + r_i]_g$ and the encryption itself $c_i = [\alpha_j\, s_i]_g \cdot [r_i\, x_i]_g$. Eventually, she sends the ciphertext to the curator **KC**.

Algorithm 2 $\mathsf{Enc}\left(\mathsf{mpk}, \mathbf{x}, \alpha_j\right)$

User u_j:
1: Samples $r_1, \ldots, r_n \xleftarrow{\$} \mathcal{S} \subseteq \mathbb{Z}_q$
2: **for** $i \in [n]$ **do**
3: Compute $h_i \leftarrow [\alpha_j + r_i]_g$ and $c_i \leftarrow [\alpha_j\, s_i]_g \cdot [r_i\, x_i]_g$
4: Sets $\mathbf{h} = (h_1, \ldots, h_n)$ and $\mathbf{c} = (c_1, \ldots, c_n)$
5: Outputs $(\mathbf{h}, \mathbf{c})$

Consider now an external data analyst **DA** who wants to partially decrypt $\mathbf{c}$ to know the position of $v \in \mathbb{Z}_t$ in the vector $\mathbf{x}$. To do so, he sends $\mathsf{req}(j, v)$ to

KC, where $\mathsf{req}(j, v)$ is a request for the identifier j of u_j and a value v. The key curator runs algorithm 3 and computes the decryption key $\mathsf{dk}_{j,v} = (\mathsf{k}_1, \ldots, \mathsf{k}_n)$, where each $\mathsf{k}_i = [\alpha_j (v - s_i)]_g$ is constructed with the entry s_i of the secret key msk. Note that this decryption key is specific to both the user's identifier and the value v. Eventually, the curator sends back $\mathsf{dk}_{j,v}$.

Algorithm 3 $\mathsf{KDer}(\mathsf{msk}, v, j)$

1: **for** $i \in [n]$ **do**
2: $\quad \mathsf{k}_i \leftarrow [\alpha_j (v - s_i)]_g$
3: Outputs $\mathsf{dk}_{j,v} = (\mathsf{k}_1, \ldots, \mathsf{k}_n)$

With the knowledge of $\mathsf{dk}_{j,v}$, **DA** can run algorithm 4 to decrypt $\mathbf{c}$ partially. For each entry of $\mathbf{c}$, **DA** computes $y_i \leftarrow c_i \cdot h_i^{-v} \cdot k_i$, that is $y_i = [r_i (x_i - v)]_g$. If the parameters are chosen according to subsection 4.2, $y_i = 1$ if $x_i = v$ and y_i' otherwise, with y_i' providing no information on the plaintext x_i. Therefore, **DA** obtains the expected result and nothing more.

Algorithm 4 $\mathsf{Dec}(\mathsf{dk}_{j,v}, \mathbf{c})$

1: **for** $i \in [n]$ **do**
2: $\quad$ Computes $y_i \leftarrow c_i \cdot h_i^{-v} \cdot k_i$; it follows:
3: $\quad y_i = [\alpha_j s_i + r_i x_i]_g \cdot [-v\,\alpha_j - v\,r_i]_g \cdot [\alpha_j (v - s_i)]_g$
4: $\quad\quad = [r_i (x_i - v)]_g$
5: Outputs $\mathbf{y} = (y_1, \ldots, y_n)$

As stated, each entry of the vector $\mathbf{y} = (y_1, \ldots, y_n)$ is equal to 1 *iff* q divides $r_i (x_i - v)$, where q is the modulus of the group $\mathbb{G} = \langle g \rangle$ and $r_i \in \mathcal{S}$ a noise sampled during the encryption. We claim that if $\mathcal{S}$ is well-chosen, $y_i = 1$ *iff* $x_i = v$ hence the result. This matter is discussed in subsection 4.2.

4.2 Sampling the noise

Below, we prove that the set $\mathcal{S} \subset \mathbb{Z}_q$ from which the noise is sampled can be adaptively chosen to guarantee the correctness of SPADE.

By theorem 1, the order of g is $q - 1$, since g is the generator of the cyclic group $\mathbb{G}$ of order $q - 1$. Hence, $[r_i (x_i - v)]_g = 1$ *iff* $r_i (x_i - v)$ is a multiple of the order of g, that is $q - 1$. If $x_i = v$, it is apparent that we retrieve the desired result. However, we want to ensure that this is the only case by making sure that $r_i (x_i - v)$ is not a non-trivial multiple of $q - 1$. To do so, we propose to choose a Fermat number $2^{2^k} + 1$, for $k \in \mathbb{N}$, as described in definition 2. In that case, $q - 1$ is a power of 2. By choosing $r \in \mathbb{Z}_q$ odd, we ensure that $q - 1$ and r_i are relatively prime, hence $q - 1$ divides $r_i (x_i - v)$ *iff* $q - 1$ divides $(x_i - v)$, that is

$x_i - v = 0 \mod q - 1$. If we choose $x_i, v \in \mathbb{Z}_t$ with $t < q - 1$, this equation has a single solution $x_i = v$.

5 SPADE protocol

This subsection defines the entities, system model, and threat model we endow to our algorithms from subsection 4.1 when building our protocols. Our first protocol relies on a pragmatic approach (subsection 5.1), where an authority manages the keys and the users simply register to the system. Commonly called SPADE, this is the protocol we refer to unless stated otherwise. Our second approach relies on a *decentralized* setup, requiring greater involvement from the users. This particular model is defined in subsection 5.2.

5.1 System and threat model

System model The model we consider consists of three entities: *(i)* Key Curator (**KC**), *(ii)* Users (**U**) and *(iii)* Data Analyst (**DA**).

- Key Curator (**KC**): The curator is a fully trusted authority responsible for the generation of user master keys $\mathsf{msk}, \mathsf{mpk}$ and the storage of user private keys $\alpha_1, \ldots, \alpha_m$ registered into the system. Upon request of an analyst **DA** for a value v and user u_j, it is responsible for the generation of the corresponding decryption key $\mathsf{dk}_{j,v}$;
- Users ($\mathbf{U} = (u_1, \ldots, u_m)$): The users are the owners of the data. They register to the system by sending their private key α_j to **KC**. They are responsible for the encryption under mpk and α_j of said data into ciphertexts. They have to trust **KC** with their data but do not have to trust each other and do not interact with **DA**. The set of users is not fixed, and we assume that a rightful user u_{m+1} can dynamically register to the system by sending a request containing their private key α_{m+1} to **KC**;
- Data Analyst (**DA**): The analyst is an external party wishing to learn the number of times a value v occurs in a message $\mathbf{x}$. With exclusive access to this as a ciphertext, **DA** sends a request to **KC** for this value and a specific user u_j, which provides the corresponding decryption key $\mathsf{dk}_{j,v}$, hence the result of the computation.

Threat model We assume the key curator is a fully trusted authority. For the users, **KC** has the ability to generate all the functional decryption keys, hence accessing all data. Likewise, for the analyst, **KC** knows the information v and j requested by **DA**. Furthermore, the latter can not check if $\mathsf{dk}_{v,j}$ is indeed the key for the requested values v and j. It is important to highlight that *verification* methods that rely on discrete logarithms exist to solve this last issue, but this is out of the scope of this article. Finally, the users do not have to trust each other. The existence of the secret integer α_j not only ensures the privacy of data but also guarantees that a user can not encrypt under the identifier of another user.

Limitations Relying on a trusted authority permits perfect management of both encryption and decryption keys. However, trusting a third party significantly constrains privacy-preserving protocols. In the above-mentioned system model, our protocol revolves around the trusted authority.

To resolve this problem, we propose an alternate protocol, called *decentralized*, that proposes the same functionalities as the classical SPADE presented in section 5.

5.2 Decentralized protocol

The principle of a *decentralized* FE scheme is to rely only on users for key management. As such, users are responsible for generating the public and private keys and the functional decryption key. Due to this approach, one can avoid relying on a trusted authority and using a server to store the ciphertexts.

Decentralized system model The decentralized construction consists of three entities: *(i)* Server (**S**), *(ii)* Users (**U**) and *(iii)* Data Analyst (**DA**).

- Server (**S**): The server is responsible for storing the ciphertexts the users produce. Upon request of an analyst, it can provide specific ciphertexts.
- Users ($\mathbf{U} = (u_1, \ldots, u_m)$): Users are the owners of the data. In the decentralized protocol, they are responsible for the system's setup. Together, they generate the master public key mpk using *multi-party computation* (MPC) techniques described in section 5.3. They encrypt their data under mpk and α_j and store the corresponding ciphertexts on **S**. They do not have to trust the server, nor do they have to trust each other. Upon request of **DA**, a user u_j generates the decryption key $\mathsf{dk}_{j,v}$ corresponding to their own identifier and a value v;
- Data Analyst (**DA**): The analyst is an external party wishing to learn the number of times a value v occurs in a message $\mathbf{x}$. Having only access to this message as a ciphertext, **DA** sends a request to a user u_j, who provides the corresponding decryption key $\mathsf{dk}_{j,v}$, hence the result of the computation.

Remark that an external user u_{m+1} can dynamically register to the system by generating their own key α_{m+1} and encrypting their data under public $g^{\alpha_{m+1}}$ and mpk. However, since the system has already been set up, she has to trust the users $u_1, \ldots, u_m$ with the master key mpk.

Threat model The threat model we propose is similar to section 5 other than the fact that the trusted authority no longer exists.

- Users ($\mathbf{U} = (u_1, \ldots, u_m)$): If the MPC generation of the master public key is done securely, users do not have to trust each other. However, if an external user u_{m+1} dynamically registers to the system after the keys have been generated, she has to trust at least one of the users $u_1, \ldots, u_m$ who participated in generating the master key. Regarding functional decryption, in section 6 shows that a malicious data analyst, given a decryption key for a user, can

not forge a decryption key for another user. It follows that user data remains secure if a malicious data analyst colludes with any number of other users.
– Data Analyst (**DA**): To retrieve the result of a computation, the data analyst interacts with a user on one side and the server on the other. He has to trust both parties with the data, as we do not provide the verification feature. Thus, he has to trust that a ciphertext $\mathbf{c}$ is well-formed by the owner of the data, stored properly by the server, and that the decryption key $\mathsf{dk}_{j,v}$ indeed corresponds to the value v and user j.

To this end, users need to communicate directly with the analyst.

5.3 Decentralized key generation

In the decentralized setup, users are responsible for generating the (master) public key. In practice, it means that users $(u_1, \ldots, u_m)$, $m \in \mathbb{N}$ have to generate altogether $([s_1]_g, \ldots, [s_n]_g)$, $n \in \mathbb{N}$ such that the two following properties are fulfilled: *(i)* $[s_i]_g$ depends on u_j, $\forall (i,j) \in [n] \times [m]$, *(ii)* $\forall i \in [n]$, $[s_i]_g \sim [s_k]_g \sim g'$ are indistinguishable for $i \neq k$ and $g' \xleftarrow{\$} \mathbb{G}$.

This article does not detail these techniques; we only claim their existence. We refer the reader to related works [12,31] for papers addressing this specific field. Eventually, this MPC algorithm permits the generation of $\mathsf{mpk} = ([s_1]_g, \ldots, [s_n]_g) \in \mathbb{G}^n$ where $s_1, \ldots, s_n$ are secret parameters unknown by the users. This restriction guarantees the absence of a backdoor for the decryption. Additionally, in this setup, generating the functional decryption key for a user u_j requires the knowledge of the identifier α_j. Indeed, $(\mathsf{dk}_{j,v})_i = (g^v)^{\alpha_j} \cdot (g^{s_i})^{\alpha_j}$. Without the secret key $\mathsf{msk} = (s_1, \ldots, s_n)$ previously possessed by **KC**, computing $[s_i\, \alpha_j]_g$ requires α_j.

6 Security Analysis

This section establishes the *semantic* security of the construction given in section 4 and the protocol described in section 5. We recall this notion of semantic security in definition 4. First, we consider security against an external eavesdropper, that is, security in the sense of Public Key Encryption.

Definition 4 (IND-CPA security).

Let $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KDer}, \mathsf{Dec})$ *be an MCFE scheme following definition 3. We define the semantic (ciphertext indistinguishability) security of MCFE as the security of the underlying PKE scheme.*

MCFE *is said to be* $\lambda - IND - CPA$ *secure if and only if the advantage* $\epsilon(\lambda) = \left|\frac{1}{2} - Pr[\beta' = \beta]\right|$ *is negligible. More generally,* MCFE *is* $IND - CPA$ *secure if it is* $\lambda - IND - CPA$ *secure for a polynomial security parameter* λ.

IND-CPA Security

Initialize:
The challenger runs $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}\left(1^\lambda\right)$ and gives mpk to $\mathcal{A}$

Challenge:
$\mathcal{A}$ submits two messages $\mathbf{x_0}, \mathbf{x_1} \in \mathcal{M}$
The challenger runs $\mathsf{Enc}\left(\mathsf{mpk}, \mathbf{x}_\beta\right) \rightarrow \mathbf{c}_\beta$ and sends $\mathbf{c}_\beta$ to $\mathcal{A}$

Guess:
$\mathcal{A}$ outputs a guess on the value of β

6.1 Semantic security of SPADE

In this section, we prove that SPADE satisfies the security assumption of definition 4. We implement a reduction to the Multi-Users Additive ElGamal scheme, recalled in the box below. Note that ElGamal maintains its security in a setting involving multiple users even when the same random value r is reused, as shown in [7]. This differs from the traditional single-user ElGamal setup, where reusing the same randomness r would compromise security. However, in scenarios with $n > 1$ distinct key pairs, this vulnerability does not apply.

Additive ElGamal Algorithms

$\mathsf{EG.KGen}\left(1^\lambda\right)$:
Sample $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$ and compute $\mathsf{pk} \leftarrow \left(g^{\mathsf{sk}}\right)$
Return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{EG.Enc}\left(\mathsf{pk}, x\right)$:
Sample $r \xleftarrow{\$} \mathbb{Z}_q$ and compute $c \leftarrow (g^r, \mathsf{pk}^r g^x)$
Return c

$\mathsf{EG.Dec}\left(\mathsf{sk}, c\right)$:
Compute $y \leftarrow g^{r \cdot \mathsf{sk} + x} \cdot \left((g^r)^{\mathsf{sk}}\right)^{-1} = g^x$
Then solve the discrete logarithm $g^x \mapsto x$
Return x

This double game is modeled using three PPT algorithms: on the one hand, the challenger and its adversary $\mathcal{B}$ for the ElGamal game, and on the other hand, the adversary $\mathcal{A}$ for SPADE game, where $\mathcal{B}$ plays the role of the challenger.

ElGamal game: The challenger generates n pairs of keys $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{EG.KGen}(1^\lambda)$ and sends $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ to $\mathcal{B}$. The latter picks two n-tuples of distinct messages $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ and sends them to the challenger, who randomly picks one of them. Without a lack of generality, assume that the challenger encrypts $\mathbf{x}$. She produces a ciphertext $\mathbf{c} \leftarrow ([r]_g, c_1, \ldots, c_n)$, with $c_i = [x_i + r \cdot \mathsf{sk}_i]_g$ and sends it to $\mathcal{B}$.

SPADE game $\mathcal{B}$ forwards $([\mathsf{sk}_1]_g, \ldots, [\mathsf{sk}_n]_g, [r]_g)$ to $\mathcal{A}$. Then, $\mathcal{A}$ generates a noise $r' \xleftarrow{\$} \mathcal{S}$ according to SPADE parameters and produces the ciphertext $\mathbf{c}' \leftarrow ([r \cdot r']_g, [r' \cdot x_1 + r' \cdot \mathsf{sk}_i]_g, \ldots, [r' \cdot x_n + r' \cdot r \cdot \mathsf{sk}_i]_g)$ from $\mathbf{c}$. By renaming $\alpha \leftarrow r \cdot r'$ and computing $[\alpha + r']_g \leftarrow [\alpha]_g \cdot [r']_g$ she can submit $([\alpha + r']_g, [r' \cdot x_1 + \alpha \cdot \mathsf{sk}_1]_g, \ldots, [r' \cdot x_n + \alpha \cdot \mathsf{sk}_n]_g)$.

Conclusion Denote $\epsilon(\lambda)$ the advantage of $\mathcal{A}$ on the SPADE game. Then, she has at least an advantage $\epsilon(\lambda)$ on the n instances of the ElGamal game. As it is proven to be λ-IND-CPA secure, the advantage on the n ElGamal games is negligible. It follows that $\epsilon(\lambda)$ is negligible, hence SPADE is λ-IND-CPA secure.

6.2 Security of the decryption key

Having proved the IND-CPA security, we analyze the leakage of information produced by using a decryption key. In fact, in PKE, a decryption key $\mathsf{dk}_{j,v}$ could be simply seen as a *trapdoor*, that is an additional insight, exterior to the protocol, leaking information on the public keys or ciphertexts. While this leakage is voluntary in FE, it is mandatory to guarantee that $\mathsf{dk}_{j,v}$ does not give more insights than the requested result the analyst is allowed to possess. To this end, we must proceed in two steps: (i) prove the decryption key's indistinguishability, and (ii) prove the decryption key's unforgeability. The latter step is divided into two sub-steps: unforgeability for another value and unforgeability for another user's identifier. These two attacks are detailed thereunder.

Attack 1 ((Value) Key Forgery Attack) *Let $\mathcal{A}$ be a PPT adversary. $\mathcal{A}$ successfully performs a Key Forgery Attack for another value if they manage to forge a decryption key $\mathsf{dk}_{j,v'}$ with the knowledge of nothing else than mpk, $\mathsf{dk}_{j,v}$ for $v \neq v'$ and a polynomial number of ciphertexts.*

Attack 2 ((Identifier) Key Forgery Attack) *Let $\mathcal{A}$ be a PPT adversary. $\mathcal{A}$ successfully performs a Key Forgery Attack for another user's identifier if they manage to forge a decryption key $\mathsf{dk}_{j',v}$ with the knowledge of nothing else than mpk, $\mathsf{dk}_{j,v}$ for $j \neq j'$ and a polynomial number of ciphertexts $\mathbf{c_1}, \ldots, \mathbf{c_n}$.*

Indistinguishability of the decryption key We rely on the Decisional Diffie-Hellman (DDH) hard problem and assume that an eavesdropper has access to $h_i = [\alpha_j + r_i]_g, \mathsf{pk}_i = [s_i]_g$ and $(\mathsf{dk}_{j,v})_i = [\alpha_j(s_i - v)]_g$. In this proof, we make the stronger assumptions that the adversary $\mathcal{A}$ knows v and also has access to $[\alpha_j]_g$ to consider the DDH triplet $([\alpha_j]_g, [s_i]_g, [\alpha_j\, s_i]_g)$. We will show that if $\mathcal{A}$ has a non-negligible advantage to distinguish between $[\alpha_j(s_i - v)]_g$ and a random element $[a]_g \xleftarrow{\$} \mathbb{G}$, he has a non-negligible advantage to distinguish the DDH distribution introduced before. Denote ϵ_0 the advantage of $\mathcal{A}$ to distinguish between $([\alpha_j]_g, [s_i]_g, [\alpha_j(s_i - v)]_g)$ and $([\alpha_j]_g, [s_i]_g, [a]_g)$. $\mathcal{A}$ can easily forge $[\alpha_j\, s_i]_g \leftarrow [\alpha_j(s_i - v)]_g \cdot [\alpha_j]_g^v$ hence giving him an advantage ϵ_0 to distinguish $[\alpha_j\, s_i]_g$ from a random value $[a]_g$. As the last scenario is the traditional DDH hard problem, the advantage ϵ_0 is negligible, which concludes.

Value Key Forgery Attack Soundness Let $\mathcal{A}$ be a PPT adversary that possesses a decryption key $\mathsf{dk}_{j,v}$ for a user u_j and a value v, the public parameters and a polynomial number of ciphertexts. We hereby prove that $\mathcal{A}$ can not forge $\mathsf{dk}_{j,w}$ for the same user u_j and value $v \neq w$ with non-negligible probability. In this scenario, we assume that $\mathcal{A}$ does not know $[\alpha_j]_g$. This is a valid assumption as this identifier is known only by the fully-trusted **KC** and by the user u_j, who only leaks their data if they collude with $\mathcal{A}$. Besides, to retrieve $[\alpha_j]_g$ from a ciphertext, $\mathcal{A}$ has to be able to compute $[\alpha_j]_g$ from $[\alpha_j + r_i]_g$ that is impossible without knowing $[r_i]_g$. To prove the unforgeability, we proceed by contrapose and denote ϵ_1 the advantage of $\mathcal{A}$ on forging $[\alpha_j(s_i - w)]_g$ for any $v \neq w$. We can reduce to the case $w = 0$ as the knowledge of two decryption keys $\mathsf{dk}_{j,v}$ and $\mathsf{dk}_{j,w}$ permits the construction of $\mathsf{dk}_{j,0}$. Forging $\mathsf{dk}_{j,w}$ for $w = 0$ is equivalent to retrieve $[\alpha_j\, s_i]_g$ that is $[\alpha_j(s_i - v)]_g \cdot [-\alpha_j\, s_i]_g$ that is $[\alpha_j]_g$. It follows that, given the DDH triplet $([\alpha_j]_g, [s_i]_g, [a]_g)$, $\mathcal{A}$ can determine if $[a]_g = [\alpha_j\,(s_i - v)]_g$ or if $a \xleftarrow{\$} \mathbb{Z}$ with advantage ϵ_1. However, as $\mathcal{A}$ can not distinguish these quantities with a non-negligible advantage, ϵ_1 is negligible. Finally, the advantage of forging $\mathsf{dk}_{j,w}$ knowing only $\mathsf{dk}_{j,v}$ is negligible.

Unforgeability for another user Let $\mathcal{A}$ be a PPT adversary that possesses a decryption key $\mathsf{dk}_{j,v}$ for a user u_j and a value v, the public parameters and a polynomial number of ciphertexts. We hereby prove that $\mathcal{A}$ can not forge $\mathsf{dk}_{k,v}$ for the same value v and user u_k, $k \neq j$ with non-negligible probability. As before, we reasonably assume that $\mathcal{A}$ does not know $[\alpha_k]_g$. To prove the unforgeability, we proceed by contrapose and denote ϵ_2 the advantage of $\mathcal{A}$ on forging $[\alpha_k(s_i - v)]_g$ for any $k \neq j$. In particular, $\mathcal{A}$ can forge such a key for $\alpha_k = \alpha_j + 1$, hence being able to retrieve $[s_i]_g \leftarrow [s_i - v]_g \cdot [v]_g$ from $[(s_i - v)(\alpha_k - \alpha_j)]_g \leftarrow [(s_i - v)\alpha_k]_g \cdot [-(s_i - v)\alpha_j]_g$. As in the previous proof, it follows that $\mathcal{A}$ can distinguish $\mathsf{dk}_{j,v}$ from a random value with advantage ϵ_2. Assuming that DDH is secure, this advantage is negligible.

7 Evaluation

In this section, we first examine the running time and memory usage of SPADE through different benchmarks. Then, we present two real-world use cases of SPADE namely *(1) Analysis of Hypnogram Data* and *(2) Analysis of Genomic Records* in a client-server setting. In the first use case, SPADE will be used to analyze *hypnogram* files and provide necessary information to an analyst for sleep medicine. In the second use case, it will process *DNA sequences* and allow an analyst to execute a search query on encrypted data. To make our implementation more usable, we implemented SPADE in a client-server setting using Golang v1.21.5 as the programming language and the gRPC [2] and Protocol Buffer [3] libraries. Employing gRPC as a remote procedure call tool and Protocol Buffer to serialize structured data for handling messages enables inner communication between SPADE's components and the client-server communication for users and service providers, regardless of the platform. Our experiments were conducted

in a single-threaded environment, with the client side operating on a laptop equipped with an Intel Core i5-9300H CPU @ 2.40GHz and 16GB memory and the server side on a PC powered by an Intel Core i7-8700 CPU @ 3.20GHz with 64GB memory.

7.1 Open Science

To support open science and reproducible research, we made our implementation source code available online[3].

7.2 SPADE Benchmarks

Following our four algorithms in Section 4.1, we implemented our scheme with a modular approach including different components as $\mathsf{SPADE} = \{\mathsf{Setup}, \mathsf{Register}, \mathsf{Encryption}, \mathsf{KeyDerivation}, \mathsf{Decryption}\}$. We separated the user registration phase from the setup phase to benchmark its performance precisely when we have different numbers of users. To comprehensively compare performance across different system scales, we selected a set of benchmarks, denoted as $\texttt{Benchmarks} = \{\mathbf{B1}, \mathbf{B2}, \mathbf{B3}, \mathbf{B4}\}$. The smallest benchmark, **B1**, involves 10 users, each with an input vector containing 1,000 elements, while the largest one, **B4**, includes 10,000 users, each with an input vector containing 100,000 elements.

As presented in Table 1, with the security parameter $\lambda = 128$, the average running time and memory usage for each benchmark gradually increase. For the smallest benchmark **B1**, the Setup for 10 users with input vectors containing 1,000 elements takes $0.013ms$ and requires $1.79KB$ of memory. The Register phase for 10 users takes $0.000072ms$ and uses $0.02KB$ of memory. Additionally, the Encryption, KeyDerivation, and Decryption results are for plaintext and ciphertext vectors, each containing 1,000 elements. Encrypting an input vector takes $0.03ms$ and requires $5.52KB$ of memory. Similarly, key derivation and decryption processes take $0.0065ms$ and $0.0022ms$ in running time and require $2.27KB$ and $0.86KB$ of memory, respectively.

While in the most significant benchmark, **B4**, the Setup for 10,000 users with input vectors containing 100,000 elements takes $0.658ms$ and requires $179.69KB$ of memory. The Register phase for 10,000 users takes $0.065ms$ and uses $17.81KB$ of memory. Also, the Encryption, KeyDerivation, and Decryption results are for plaintext and ciphertext vectors, each containing 100,000 elements. Encrypting an input vector takes $3.905ms$ and requires $551.71KB$ of memory. Similarly, key derivation and decryption processes take $0.756ms$ and $0.168ms$ in running time and require $226.56KB$ and $85.94KB$ of memory, respectively.

As shown in Figure 1 and Figure 2, increasing the number of users and the size of the input vectors results in a linear increase in running time and memory usage, respectively. Furthermore, it reveals that, as expected, the most expensive step of the SPADE scheme is the Encryption, which takes less than $4ms$ even for the largest benchmark **B4** and requires only $550KB$ memory.

[3] https://github.com/hosseinabdinf/SPADE

Table 1. SPADE Benchmarks

Component	Time (ms/op)	Memory Allocations (KB/op)
B1: Number of Users=10 – Vector Size=1,000		
Setup	0.013474	1.798828125
Register	0.000072	0.019531251
Encryption	0.030317	5.520507813
KeyDerivation	0.006535	2.267578125
Decryption	0.002233	0.860351563
B2: Number of Users=100 – Vector Size=10,000		
Setup	0.072073	17.97949219
Register	0.001419	0.179687511
Encryption	0.216872	55.20800781
KeyDerivation	0.078610	22.66503906
Decryption	0.015531	8.597656252
B3: Number of Users=1,000 – Vector Size=50,000		
Setup	0.326208	89.84765625
Register	0.006678	1.783203125
Encryption	1.024057	275.8828125
KeyDerivation	0.353401	114.0654297
Decryption	0.078852	42.97363281
B4: Number of Users=10,000 – Vector Size=100,000		
Setup	0.65899	179.6943359
Register	0.06459	17.81347656
Encryption	3.90593	551.7138672
KeyDerivation	0.75656	226.5664063
Decryption	0.16835	85.94628906

7.3 SPADE Use Cases

As mentioned previously in Section 5, a user (j) can encrypt a vector of integers ($\mathbf{x}$) using (mpk, α_j). This encrypted vector can then be stored on an *untrusted* server for later use. An analyst who holds the corresponding decryption key ($\mathsf{dk}_{j,v}$) can request access to the user's vector of ciphertexts and decrypt it for further analysis. As shown in Figure 3, as the result of decryption, the analyst will have a vector consisting of ones (1) for the indices that match the query value (v) and unknown values (❄) for the remaining indices.

Use Case#1: Analysis of Hypnogram Data Studying sleep behaviour for sleep medicine has been a focus in healthcare research for a long time [27]. The American Academy of Sleep Medicine (AASM) provides standardized rules to score sleep stages. The sleep stages are assigned to each 30-second epoch in the recording. The assigned epochs are visualized in a *hypnogram* file, and

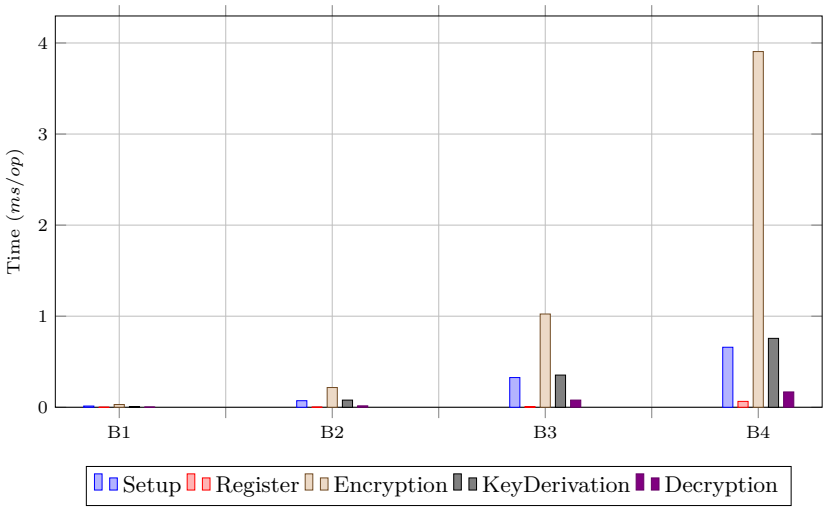

Fig. 1. Running Time for Different Benchmarks

quantitative measures are extracted to provide more information about the sleep cycle [30]. As an application, assume that an analyst wants to extract information regarding a specific sleep stage (v) by running a query over a hypnogram file to answer the following questions:

Q1. How many times does the value v appear in the hypnogram's values? This reveals the total length of the sleep stage v.

Q2. How many changes exist where the value v jumps to the other values? This shows the changes from the sleep stage v to other sleep stages.

Q3. How many times does the value v appear in a sequence? This indicates the length of sleep stage v before a change in sleep stage happens.

To do so, we implemented a client-server application, utilizing the SPADE scheme with a *Server/Curator* and two different types of clients, namely, *User* and *Analyst*. The trusted entity (*Curator*) generates the system parameters and master public/private keys. Then, it shares the public parameters with all other entities. The curator transfers the keys to users and handles the user's registration phase. The user encrypts a hypnogram file and sends it to an untrusted server to be stored. An analyst with the corresponding decryption key for value v requests access to the user's encrypted hypnogram to partially decrypt the ciphertexts and extract the information for sleep stage v.

Storage Costs Our experiments were conducted on 590 hypnogram files from the SIESTA dataset [22], each with 1,000 elements ranging from 1 to 10, indicating different sleep stages. As shown in Table 2, the storage cost for saving a single hypnogram file is around $2KB$, increasing to $53.98KB$ after encryption by SPADE. Likewise, the entire hypnogram dataset requires $1138.7KB$ to be stored, whereas the storage cost for all encrypted hypnograms on the server

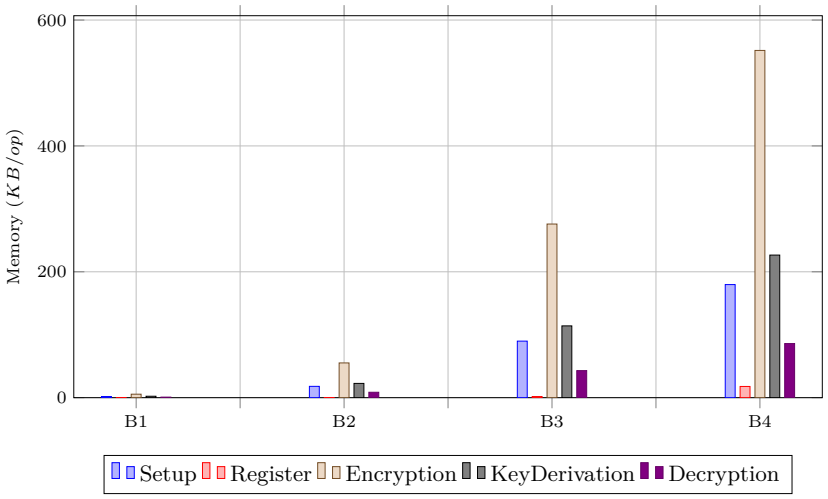

Fig. 2. Memory Usage for Different Benchmarks

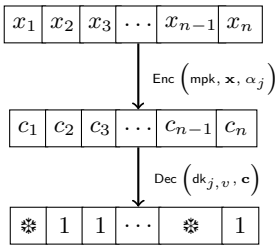

Fig. 3. Encryption and Decryption in SPADE

is $31,911.94KB$. Therefore, in this use case, the SPADE's storage cost increases by $28\times$ for each $1KB$ of data for the security parameter $\lambda = 128$.

Running Time & Communication Costs We conducted the experiments using a 1GB/s LAN connection. On the user side, the process involves sending a request for public parameters $\{q, g, \mathsf{mpk}\}$, encrypting hypnogram data, and sending a message that contained $\{id, g^{\alpha_{id}}, \mathbf{c_{id}}\}$ to the server. As shown in Table 3, each user took approximately $36ms$ to complete the process, requiring $54.05KB$ of network bandwidth. Similarly, the analyst sends a request to the server for public parameters. Then, the analyst sends a request to the server, including the user identifier and the sleep stage value as $\{id, v\}$, respectively. In response, the analyst receives decryption keys and the user's encrypted hypnogram, represented as $\{\mathsf{dk}_{id,v}, \mathbf{c_{id}}\}$, respectively. The analyst can then partially decrypt the user's hypnogram. It takes approximately $16ms$ for each analyst to complete the process and requires $72.02KB$ of network bandwidth.

Table 2. Storage Costs for Use Case #1

Description	Size (KB)
Plain Hypnogram Record	1.93
Encrypted Hypnogram Record	53.98
Plain Hypnograms Dataset	1138.7
Encrypted Hypnograms Dataset	31911.94

Table 3. Running Time & Communication of Use Case #1

Client	Running Time (ms)	Communication (KB)
User	35.59	54.05
Analyst	15.75	72.02

Use Case#2: Analysis of Genomic Records The collection of genomic data has grown remarkably in many applications in the healthcare sector. Medical researchers use genetic information for various applications, including sequencing [19], genomic sequence assembly [26], and Genomic-Wide Association Studies (GWAS) [10]. The size of genomic databases is enormous and contains DNA sequences. Each DNA sequence represents each individual's genetic variation. These sequences are typically represented by characters corresponding to the four nucleotides, adenine (A), cytosine (C), guanine (G), and thymine (T), and are extensively long.

One of the main purposes of DNA sequence analysis is to investigate the relationship between various genomic variations in humans and biological or health-related traits by querying genomic databases [17]. For example, an analyst wants to know how many users in a dataset have specific genotypes at certain locations in their genome. Despite the importance of running analysis queries over genomic databases, preserving genomic data privacy is crucial. Researchers in [16,17] analyzed and proposed different privacy-preserving approaches to perform the *Count Query* over genomic databases. The comparison of these approaches is out of the scope of this article. Yet, we found their research question **"How many records in a genomic database contain a specific set of genotypes at a certain location?"** as an interesting application for our scheme. By utilizing SPADE, analysts can query specific genotypes (value v) to access the **number** and **location** of matches.

Storage Costs We utilized the publicly available Harvard dataset of Single Nucleotide Polymorphisms (SNPs) [23] as input for our experiment. Due to the dataset's size, we selected 5,012 files and transformed the DNA sequences into dinucleotides, as already represented by the authors in [16] for conducting count queries. A dinucleotide is a pair of two different nucleotides, for example, "AG", "CC", and so on. As given in Table 4, the SNP file in our dataset has a storage size of $0.078MB$ and contains DNA sequences with 78,214 elements, each representing a dinucleotide. Since SPADE only works with integers, we mapped the total possible combinations of nucleotide pairs into an integer set ranging

from 1 to 16 for using them in our scheme. After encryption, the storage size for the DNA record increases to $4.223MB$. Similarly, the entire DNA dataset requires $392.02MB$ to be stored, whereas the storage cost for all encrypted DNA sequences on the server is $21,186.14MB$. Therefore, in this use case, the SPADE's storage cost increases by $54\times$ for each $1KB$ of data for the security parameter $\lambda = 128$.

Table 4. Storage Costs for Use Case #2

Description	**Size** (MB)
Plain DNA Record	0.078
Encrypted DNA Record	4.223
Plain DNA Dataset	392.02
Encrypted DNA Dataset	21186.14

Running Time & Communication Costs Similarly, we conducted the experiments using a 1GB/s LAN connection for this use case. On the user side, the process involves sending a request for public parameters $\{q, g, \mathsf{mpk}\}$, encrypting the DNA data, and sending a message that contains $\{id, [\alpha_{id}]_g, \mathbf{c_{id}}\}$ to the server. As given in Table 5, each user took approximately 1750ms ($1.75s$) to complete the process, requiring $4.22MB$ of network bandwidth. Similarly, the analyst sends a request to the server for public parameters. Then, the analyst sends a request to the server, including the user identifier and the sleep stage value as $\{id, v\}$, respectively. In response, the analyst receives decryption keys and the user's encrypted DNA, represented as $\{\mathsf{dk}_{id,v}, \mathbf{c_{id}}\}$, respectively. The analyst can then partially decrypt the user's DNA. Each analyst takes approximately $770ms$ ($0.77s$) to complete the process and requires $5.63MB$ of network bandwidth.

Table 5. Running Time & Communication of Use Case #2

Client	**Running Time** (ms)	**Communication** (MB)
User	1750	4.22
Analyst	770	5.63

7.4 Comparing Two Use Cases

We presented the results of two different use cases for our scheme: *(1) the hypnogram use case* for quantitative data analysis, and *(2) the DNA use case* for qualitative data analysis to show the potential of SPADE. Despite the different nature

of each use case, to perform encryption and decryption using SPADE, the input data should be vectors of integers. Therefore, it is necessary to map non-integer data into integers through a pre-processing function. The mapping process can be relatively cheap since it does not require complicated operations; for example, in the case of DNA data, the mapping function only takes tens of milliseconds. Besides, it can be done offline or during data collection.

The number of users (number of input files, one per user) for the DNA use case is 5,012, while for the hypnogram use case is 590. Also, each input vector for the DNA use case contains 78,214 elements, while the hypnogram use case contains 1,000 elements. Regarding storage costs, on the user side, each DNA file is approximately 40× larger than the hypnogram file, while on the server side, each DNA file is approximately 78× larger than the hypnogram file.

The size of the ciphertext depends on the maximum size of the input vector. During the setup phase, the key curator generates the keys $(\mathsf{msk}, \mathsf{mpk})$ based on the maximum size of the input vector. Hence, we have larger input vectors in the DNA use case; the ciphertext size is expected to be larger than the ciphertext size in the hypnogram use case, which results in more storage costs. The same results apply to the communication costs since the users send their ciphertexts to be stored in the server, and the analyst also asks the server to receive them. Thus, to clearly present the storage and communication costs, we used different measurement units for each use case: Kilobytes (KB) for the hypnogram use case and Megabytes (MB) for the DNA use case.

Regarding the running time and computation costs, as shown in Table 1, the most expensive operation is encryption, where the users encrypt the input vector using the master public key (mpk) and their private key (α_j). Based on the difference in the input vector size between the two use cases, in the hypnogram application, all operations for the users and analysts can be completed in tens of milliseconds. For the DNA application, these operations can take up to 2 seconds.

8 Conclusion

As every aspect of human life is nowadays fueled with data, the need for secure solutions to use that data while protecting people's privacy continues to grow. In this work, we proposed and implemented a practical and efficient FE solution for analyzing *qualitative* data in a multi-user setup. Our scheme is capable of providing targeted insights on qualitative datasets, such as counting the occurrences of a specific value in a DNA sequence. Our goal is to illustrate the wide range of applications that modern cryptographic methods like FE can enable. By doing so, we aim to pave the way for creating trustworthy privacy-respecting services.

References

1. Gofe. Online: https://github.com/fentec-project/gofe (06 2024), goFE

2. grpc v1.64. Online: https://github.com/grpc/grpc-go (06 2024), gRPC
3. protobuf v1.34. Online: https://github.com/protocolbuffers/protobuf-go (06 2024), protocol Buffers
4. Abdalla, M., Benhamouda, F., Gay, R.: From single-input to multi-client inner-product functional encryption. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 552–582. Springer International Publishing, Cham (2019)
5. Ayday, E., Raisaro, J.L., Hengartner, U., Molyneaux, A., Hubaux, J.P.: Privacy-preserving processing of raw genomic data. In: Garcia-Alfaro, J., Lioudakis, G., Cuppens-Boulahia, N., Foley, S., Fitzgerald, W.M. (eds.) Data Privacy Management and Autonomous Spontaneous Security. pp. 133–147. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
6. Bakas, A., Michalas, A., Dimitriou, T.: Private lives matter: A differential private functional encryption scheme. In: Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy. p. 300–311. CODASPY '22, Association for Computing Machinery, New York, NY, USA (2022)
7. Bellare, M., Boldyreva, A., Staddon, J.: Randomness re-use in multi-recipient encryption schemeas. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. pp. 85–99. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
8. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Theory of Cryptography Conference. pp. 253–273. Springer (2011)
9. Castagnos, G., Laguillaumie, F., Tucker, I.: Practical fully secure unrestricted inner product functional encryption modulo p. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018. pp. 733–764. Springer International Publishing, Cham (2018)
10. Cho, H., Wu, D.J., Berger, B.: Secure genome-wide association analysis using multiparty computation. Nature biotechnology **36**(6), 547–551 (2018)
11. Datta, P., Okamoto, T., Tomida, J.: Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography – PKC 2018. pp. 245–277. Springer International Publishing, Cham (2018)
12. Dimitriou, T., Michalas, A.: Multi-party trust computation in decentralized environments in the presence of malicious adversaries. Ad Hoc Networks **15**, 53–66 (2014), smart solutions for mobility supported distributed and embedded systems
13. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009)
14. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (2014)
15. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run turing machines on encrypted data. In: Annual Cryptology Conference. pp. 536–553. Springer (2013)
16. Hasan, M.Z., Mahdi, M.S.R., Sadat, M.N., Mohammed, N.: Secure count query on encrypted genomic data. Journal of Biomedical Informatics **81**, 41–52 (2018)
17. Jiang, B., Seif, M., Tandon, R., Li, M.: Answering count queries for genomic data with perfect privacy. IEEE Transactions on Information Forensics and Security (2023)
18. Marc, T., Stopar, M., Hartman, J., Bizjak, M., Modic, J.: Privacy-enhanced machine learning with functional encryption. In: Computer Security – ESORICS

2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I. p. 3–21. Springer-Verlag, Berlin, Heidelberg (2019)
19. Motahari, A.S., Bresler, G., David, N.: Information theory of dna shotgun sequencing. IEEE Transactions on Information Theory **59**(10), 6273–6289 (2013)
20. Nuoskala, C., Rabbaninejad, R., Dimitriou, T., Michalas, A.: Fe[r]chain: Enforcing fairness in blockchain data exchanges through verifiable functional encryption. In: Proceedings of the 29th ACM Symposium on Access Control Models and Technologies. p. 183–191. SACMAT 2024, Association for Computing Machinery, New York, NY, USA (2024)
21. Panzade, P., Takabi, D., Cai, Z.: Privacy-preserving machine learning using functional encryption: Opportunities and challenges. IEEE Internet of Things Journal **PP**, 1–1 (01 2023)
22. Penzel, T., Glos, M., Garcia, C., Schoebel, C., Fietze, I.: The siesta database and the siesta sleep analyzer. In: 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. pp. 8323–8326. IEEE (2011)
23. Ricke, D.: 11 Million SNP Profiles datasets (2018)
24. Rivest, R., Adleman, L., Deaouzos, M.: On data banks and privacy homomorphism. In: Foundations of Secure Computation, Academic Press, New York, 169-179 (1978)
25. Ryffel, T., Dufour-Sans, E., Gay, R., Bach, F., Pointcheval, D.: Partially encrypted machine learning using functional encryption. Curran Associates Inc., Red Hook, NY, USA (2019)
26. Si, H., Vikalo, H., Vishwanath, S.: Information-theoretic analysis of haplotype assembly. IEEE Transactions on Information Theory **63**(6), 3468–3479 (2017)
27. Swihart, B.J., Caffo, B., Bandeen-Roche, K., Punjabi, N.M.: Characterizing sleep structure using the hypnogram. Journal of Clinical Sleep Medicine **4**(4), 349–355 (2008)
28. Tomida, J.: Unbounded quadratic functional encryption and more from pairings. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 543–572. Springer Nature Switzerland, Cham (2023)
29. Waters, B.: A punctured programming approach to adaptively secure functional encryption. In: Annual Cryptology Conference. pp. 678–697. Springer (2015)
30. van der Woerd, C., van Gorp, H., Dujardin, S., Sastry, M., Garcia Caballero, H., van Meulen, F., van den Elzen, S., Overeem, S., Fonseca, P.: Studying sleep: towards the identification of hypnogram features that drive expert interpretation. Sleep p. zsad306 (2023)
31. Zhao, C., Zhao, S., Zhao, M., Chen, Z., Gao, C.Z., Li, H., an Tan, Y.: Secure multi-party computation: Theory, practice and applications. Information Sciences **476**, 357–372 (2019)