# Privacy-Enhanced Machine Learning with Functional Encryption

Tilen Marc[1,2,*], Miha Stopar[1,*], Jan Hartman[1], Manca Bizjak[1], and Jolanda Modic[1]

[1] XLAB d.o.o., Ljubljana, Slovenia
{name.surname}@xlab.si
[2] Faculty of Mathematics and Physics, Ljubljana, Slovenia

**Abstract.** Functional encryption is a generalization of public-key encryption in which possessing a secret functional key allows one to learn a function of what the ciphertext is encrypting. This paper introduces the first fully-fledged open source cryptographic libraries for functional encryption. It also presents how functional encryption can be used to build efficient privacy-enhanced machine learning models and it provides an implementation of three prediction services that can be applied on the encrypted data. Finally, the paper discusses the advantages and disadvantages of the alternative approach for building privacy-enhanced machine learning models by using homomorphic encryption.

**Keywords:** Functional Encryption · Cryptographic Library · Machine Learning · Homomorphic Encryption · Privacy.

## 1 Introduction

Today, almost every part of our lives is digitalized: products, services, business operations. With the constant increase in connectivity and digitalization, huge amounts of personal data are often collected without any real justification or need. On the other hand, there is a growing concern over who is in possession of this data and how it is being used. With increasingly more privacy-aware individuals and with ever stricter data protection requirements (GDPR, ePrivacy CCPA), organizations are seeking a compromise that will enable them to collect and analyse their users' data, to innovate, optimize, and grow their businesses, while at the same time comply with legal frameworks and keep trust and confidence of their users.

When individuals themselves use technologies like end-to-end encryption to protect their data, this can greatly improve their privacy online because the service providers never see raw data. But when a service provider does not have access to raw data, it cannot analyse the data and it thus cannot offer functionalities like search or data classification. Indeed, almost all rich functionality to which users are accustomed today is out of the question when encryption

* Both authors contributed equally to this work.

is used. However, there are encryption techniques which do not impose a drastic reduction of data utility and consequently functionality. Probably the most known such technique is Homomorphic Encryption (HE). HE enables additions and multiplications over the encrypted data, which consequently enables higher-level functionality such as machine learning on the encrypted data. However, HE is computationally expensive and significantly reduces service performance. Another technique, perhaps lesser known, is Functional Encryption (FE). Similarly as HE, it allows computation on encrypted data. More precisely, an owner of a decryption key can learn a function of the encrypted data. This gives a possibility to use the encrypted data for various analysis or machine learning models by controlling the information one can get from it. In this paper we present first two fully-fledged FE libraries, we outline how they can be used to build machine learning services on encrypted data, and we discuss strengths and limitations of FE compared to the HE approach.

While there exist schemes for general FE (see [14, 26, 27, 43]) they rely on non-standard, ill-understood assumptions and are in many cases extremely time-consuming. On the contrary, we focused on the implementation of efficient schemes of restricted functionality but still of practical interest. Our aim was a flexible and modular implementation that can be applied to various applications and does not predetermine usage. We offer our work as an open source, all the code with guidelines is available online on the FENTEC Github account [21].

**Contributions.** This paper addresses the lack of implementations of practical FE schemes that enable computation on the encrypted data through the following contributions:

1. *Implementation of FE libraries.* We present two fully-fledged FE cryptographic libraries, named GoFE and CiFEr. We overview the different underlying primitives (modular arithmetic, pairings, lattices) which can be chosen by the user of the library when instantiating an FE scheme. This is presented in Sections 2 and 3
2. *Performance evaluation of FE libraries.* In Section 4 we compare the efficiency of various FE schemes and underlying primitives.
3. *Design and implementation of privacy-enhanced machine learning services.* In Sections 5, 6, 7 we present the implementation and performance of three privacy-enhanced analysis services based on FE.
4. *Comparison of FE and HE approaches.* Furthemore, in Sections 5, 6, 7 we discuss the advantages and disadvantages of FE compared to the HE approach.

## 2 Functional Encryption Libraries

FE is a cryptographic procedure, which allows to delegate to third parties the computation of certain functions of the encrypted data. This can be achieved by generating specific secret keys for these functions. A FE scheme consists of a

set of five algorithms. The *setup* algorithm takes as input a security parameter and generates a mathematical group where operations take place. The *master key generation* creates a public key together with a master secret key. The *functional key derivation* algorithm takes as input the master secret key and a particular function $f$ to generate a key depending on $f$. To encrypt a message $x$, the *encryption* algorithm has to be run on input $x$ and using the public key (some schemes are private-key and require also a secret key) to obtain a ciphertext. Then, given the encryption of a message $x$, the holder of the key corresponding to the function $f$ is able to compute the value of $f(x)$ using the *decryption* algorithm but nothing else about the encrypted data is revealed.

Many recent papers [2, 3, 5, 15, 40] developed various FE encryption schemes with an aim to make such schemes practical. Nevertheless, most of them remain theoretical, since they do not provide implementation or practical evaluation of the schemes. We fill this gap by presenting two FE libraries: GoFE [25] and CiFEr [24]. GoFE is implemented in the programming language Go and is simpler to use, while CiFEr is implemented in C and aims at a lower level, possibly IoT related applications. Both provide the same FE schemes via a similar API, differences are due only to the different paradigms of the programming languages.

### 2.1 Implemented Schemes

Due to the computational complexity and impracticality of general purpose FE schemes, different schemes were designed for evaluation of various functions of lesser complexity. We separated them into three categories: inner-product schemes, quadratic schemes, and ABE schemes.

Schemes in GoFE and CiFEr use cryptographic primitives based on either modular arithmetic, pairings, or lattices. Most schemes can be instantiated from different primitives – the user can choose the primitive based on the performance requirements. In the following sections, we list the schemes and the security assumptions they are based on. The following assumptions are used: Decisional Diffie-Hellman (DDH), Decisional Composite Residuosity (DCR) (both modular arithmetic), Generic Group Model (GGM), Symmetric eXternal Diffie-Hellman (SXDH), Decisional Bilinear Diffie-Hellman (BDH), Decisional Linear (DLIN) (all pairings), Learning With Errors (LWE), and Ring Learning With Errors (ring-LWE) (both lattices).

**Inner-product schemes** Inner-product FE schemes allow an encryption of a vector $x \in \mathbb{Z}^n$ and independently generation of a key $sk_y$ depending on a vector $y \in \mathbb{Z}^n$, such that given the encryption of $x$ together with $sk_y$ one can perform a computation on the encrypted $x$ to obtain the value $x \cdot y$ (inner-product of $x$ and $y$). This simple function proves itself very useful: simple statistics of encrypted data, linear or logistic regression and more functions can be seen as computing certain inner-product of the data. We discuss two possible applications based on the inner-product in Sections 5 and 6.

The libraries currently provide inner-product schemes based on the following papers:

- **Simple Functional Encryption Schemes for Inner Products [2].** The first efficient schemes for inner-products, based on the DDH or LWE assumptions.
- **Fully Secure Functional Encryption for Inner Products, from Standard Assumptions [5].** Inner-product encryption schemes with a higher level of (adaptive) security. In addition to DDH- and LWE-based schemes a more efficient DCR-based scheme is introduced.
- **Multi-Input Functional Encryption for Inner Products: Function-Hiding Realizations and Constructions without Pairings [3].** Multi-input FE scheme for inner-products is a scheme supporting encryption of elements of vector distributed among different clients. The scheme can be instantiated on DDH, LWE, and DCR assumptions.
- **Decentralized Multi-Client Functional Encryption for Inner Product [15].** This scheme allows various users to generate ciphertexts supporting inner-product evaluation without the presence of a central authority and with functional decryption keys that can also be generated in a decentralized way. Based on SXDH assumption.

Additionally we implemented a prototype ring-LWE based inner-product scheme whose security will be dissuaded in a future work.

**Quadratic schemes** To provide an FE scheme able to evaluate an arbitrary function on encrypted data one needs to build an FE system computing polynomials of arbitrary order. Currently, no FE schemes for polynomials of order higher than 2 exist. Nevertheless, many complex functions can be realized as evaluations of quadratic polynomials. A quadratic FE scheme implemented in CiFEr and GoFE allows an encryption of a vectors $x_1, x_2 \in \mathbb{Z}^n$ and independently generation of a key $sk_H$ depending on a matrix $H \in \mathbb{Z}^{n \times n}$, such that given the encryption of $x_1, x_2$ together with $sk_H$ one can obtain the value $x_1^T H x_2$ (quadratic-product of $x_1, x_2$ and $H$). In particular if $x_1 = x_2$ this is a quadratic polynomial of values of $x_1$. Such functions are sufficient for performing many machine learning task on encrypted data. We demonstrate the use of it in Section 7 on a task of classifying encrypted images with a 2-layer neural network.

GoFE and CiFEr provide the implementation of the currently most efficient quadratic FE scheme:

- **Reading in the Dark: Classifying Encrypted Digits with Functional Encryption [40].** A scheme for quadratic multi-variate polynomials enabling efficient computation of quadratic polynomials on encrypted vectors. It can be instantiated on GGM assumption.

**ABE schemes** Attribute-based encryption (ABE) is not strictly classified as FE but it allows secure access control over data and constructions of certain functionalities on encrypted data [44]. For the latter reason, we included two such schemes in the libraries. The basic idea ob ABE is that users are given keys

depending on their attributes and are able to decrypt given data only if their attributes are sufficient.

- **Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data [30].** The first scheme which enables fine-grained sharing of encrypted data, by a key distribution process that enables decryption only for users in possession of specified attributes. Based on BDH assumption.
- **FAME: Fast Attribute-based Message Encryption [4].** A scheme that enables attribute based limitation of the access to encrypted data specified through the encryption process. Based on DLIN assumption.

## 3 Implementation of Cryptographic Primitives

GoFE and CiFEr aim at providing a flexible implementation of FE schemes. We do not use specially chosen groups and parameters which enable better performance (this can still be done by the user). Instead, we provide flexibility in terms of choosing the mathematical groups where operations take place and security parameters which determine the key lengths.

Practical FE schemes are based either on modular arithmetic, pairings, or lattices. Implementation of FE schemes based on modular arithmetic is relatively straight-forward. Our implementation is based on the representation of arbitrary big numbers using GMP library [42] in C and native package Big in Go. However, on the other hand, the implementation of schemes based on pairings and lattices requires lower-level math artillery.

Quite surprisingly we found only one pairings library which provides all required functionality. Furthermore, there is no fully-fledged library for lattice-based cryptography that could be easily reused. In what follows we present cryptographic primitives needed in FE schemes and address the issues of (lack of) their implementation.

### 3.1 Pairing schemes

Numerous libraries for pairings are available but most lack at least some essential functionality or performance optimization. The latter is crucial, since the pairing operation presents a bottleneck in many schemes. Considering existing open source implementations such as PBC [34], RELIC [22], Apache Milagro Cryptographic Library (AMCL) [9], the latter was chosen as an underlying pairing library for CiFEr because it is portable, small, and optimized to fit into the smallest possible embedded footprint. Choosing a Go pairing library to be used in GoFE was more challenging. Barreto-Naehrig [10] bilinear pairings are frequently used as they allow a high security and efficiency level. Two well-known Barreto-Naehrig pairing libraries exist for the Go programming language: BN256 [32] is a part of the official Go crypto library, Cloudflare BN256 [16] is an optimization of the latter for the improved performance. Neither of them provides hashing operations for pairing groups. We forked [32] and provided hashing operations for both groups. For $G_1$ we implemented the try-and-increment algorithm [11], while for $G_2$ we implemented the technique from [23].

### 3.2 Lattice schemes

The resistance of cryptographic protocols to post-quantum attacks is becoming ever more important as we get closer to the realization of quantum computers. Lattice-based cryptography is believed to be secure against quantum computers. Its cryptographic constructions are based on the presumed hardness of lattice problems (e.g. for example, the shortest vector problem). Currently, the most used constructions are based on the Learning With Errors (LWE) problem [39] or its algebraic ring variation (ring-LWE) [35]. Currently, FE schemes are build only on LWE assumption, however, there are two main bottlenecks in all such schemes. These are sampling random values distributed according to the discrete Gaussian distribution and matrix multiplications.

**Discrete Gaussian sampling** Discrete Gaussian sampling is a problem of sampling values distributed according to Gaussian distribution but limited only to discrete values. This issue has been tackled by many algorithms and software implementations, see [18,19,29,31]. Practical implementation of (ring-)LWE schemes available as open source libraries mostly solve this problem in two ways. Ether they avoid Gaussian sampling by replacing it with a uniform or binomial distribution or implement a fast sampler optimized by precomputations for chosen parameters. Neither of the two solutions is applicable in FE schemes. On one hand, proofs of the security of (ring-)LWE FE schemes depend on the distribution being Gaussian and can easily be broken for uniform distribution. Moreover, precomputations are not just in conflict with the flexibility of GoFE and CiFEr, but are not feasible due to greater variance needed in FE schemes.

For this reason, we implemented a discrete Gaussian sampler based on the algorithm from [19]. It is based on sampling discrete Gaussian values with small variance from precomputed tables together with uniform sampling. Such sampling is efficient but still presents a bottleneck of the schemes.

**Matrix multiplications** The second bottleneck of FE schemes based on the LWE problem is due to matrix-vector and matrix-matrix multiplications. The reason for this is that the matrices generated in the existing FE scheme have much greater dimensions and inputs. This cannot be fixed implementation-wise, thus the construction of efficient LWE based FE schemes remains an open problem. One way of avoiding costly operations and spacious public keys is by replacing LWE schemes with ring-LWE schemes [35]. We have implemented a prototype scheme using ring-LWE primitives. This replacement needs to be proved secure which we intend to do in the future work.

### 3.3 ABE schemes

ABE schemes provide functionality where a client can access or not access the decryption of a ciphertext based on a set of attributes that he or she possesses. Most ABE schemes use pairings as an underlying cryptographic primitive but

there is another, ABE specific, primitive needed: Linear Secret Sharing Scheme (LSSS) matrices.

A part of every ABE scheme is a policy that defines which entity can decrypt the ciphertext based on the attributes. A Monotone Span Program (MSP) is defined as a policy that accepts a subset of attributes as sufficient if a certain subset of chosen vectors spans a vector of ones. Hence, to create an MSP policy, one must carefully choose a set of vectors representing attributes in a way that they describe the desired rules of decryption. This set of vectors is also known as an LSSS matrix. On the other hand, expressing rules of decryption as a boolean expression is preferred for practical usage and interpretability. Therefore, we have implemented an algorithm that transforms a boolean expression into a MSP structure. We have chosen the Lewko-Waters Algorithm [33] for this task, due to its simplicity and efficiency. The algorithm can transform an arbitrary boolean expression that does not include a "NOT" operation ($\neg$) into a set of vectors (a matrix) whose dimensions only depend on the number of "AND" operators ($\wedge$) and the number of variables in the expression.

## 4 Benchmarks

In the following section, we focus on a practical evaluation of implemented schemes, comparing the benefits and downsides, and discussing their practicality for the possible uses. As noted in Section 3, the schemes are implemented with the goal of flexibility and having an easy-to-use API. Thus, the schemes can be initialized with an arbitrary level of security and other meta parameters. Since there is no universal benchmark to compare all the schemes, we evaluate them on various sets of parameters, exposing many properties of the schemes. Due to the space limitation, we do not present here the benchmarks of all the implemented schemes but rather focus on the demonstrative results. All of the benchmarks were performed on an Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz.

### 4.1 Inner-product schemes

Recall that an inner-product FE scheme is such that it allows encrypting a vector $x \in \mathbb{Z}^\ell$ and independently generating a key $sk_y$ depending on a vector $y \in \mathbb{Z}^\ell$, so that one can perform computations on the encrypted $x$ and use $sk_y$ to decrypt the inner-product $x \cdot y$ and nothing more.

As noted in Section 2, the schemes are based on different security assumptions. GoFE and CiFEr include implementation of five inner-product schemes (excluding decentralized and multi-client ones), where two of them are based on the DDH assumption, two of them are based on the LWE assumption, and one on the DCR assumption. Since both of the DDH-based and both of the LWE-based schemes have similar performance, we only compare the DDH-based scheme from [5], the LWE-based scheme from [2], and the DCR-based scheme from [5] which is also known as Paillier-based FE scheme.

| l | Paillier[Go] | Paillier[C] | LWE[Go] | LWE[C] | DDH[Go] | DDH[C] |
|---|---|---|---|---|---|---|
| 1 | 0.0664 | 0.0243 | 12.9523 | 7.3909 | 0.0080 | 0.0041 |
| 5 | 0.1922 | 0.0975 | 62.1945 | 46.2466 | 0.0402 | 0.0204 |
| 10 | 0.3551 | 0.1878 | 122.7627 | 74.8795 | 0.0840 | 0.0411 |
| 20 | 0.6720 | 0.3682 | 266.5059 | 196.6151 | 0.1584 | 0.0849 |
| 50 | 1.6293 | 0.9120 | 878.3684 | 559.6070 | 0.3954 | 0.2055 |
| 100 | 3.2427 | 1.8345 | N/A | N/A | 0.7829 | 0.4149 |
| 200 | 6.4278 | 3.6544 | N/A | N/A | 1.5710 | 0.8190 |

**Table 1.** Performance of key generation (in seconds) in inner product schemes w.r.t. vector length l

The DDH schemes assume the difficulty of computing a discrete logarithm in a quadratic residues subgroup of $\mathbb{Z}_p^*$, where the security of such assumption depends on the bit size of the prime number $p$. To achieve resistance to all known attacks with complexity less than $O(2^{128})$ it is a common practice to pick $p$ to be a safe prime with 3072 bits. The DCR assumption depends on distinguishing the so-called $n$-residues in $\mathbb{Z}_{n^2}^*$ group which further depends on the difficulty of factoring a large number $n$. We choose $n$ to be a 2048 bit number and a product of two safe primes as it is considered safe for attacks with complexity in $O(2^{128})$.

The security level of the LWE assumption is harder to access due to its novelty. The papers developing the LWE-based FE schemes argue its security based on the original work of Regev [39] while it has become a common practice in the recent proposals of (non FE) (ring-)LWE-based [7], [12], [6] schemes to evaluate this security through evaluation of attacks on the assumption. For this reason, we implemented a setup procedure that generates the parameters for each instantiation of the scheme that are secure for the so-called primal and dual attack on LWE. In fact, this was necessary since the originally proposed parameters are estimated to possess significantly less security than claimed. For the additional information on the attacks, we direct the reader to the above references.

Each inner-product scheme comprises five parts: setup, generation of master keys, encryption, derivation of a inner-product key, and decryption. In the following tables, we evaluate the performance for key generation, encryption, and decryption. The complexity of functional key derivation process is negligible in all the schemes compared to the other steps, while the setup procedure is quite consuming but can be avoided for practical applications since generating a new group for every deployment does not bring additional security.

We demonstrate the efficiency of the schemes depending on parameters $\ell$, defining the dimensionality of the encrypted vectors, and $b$, being the upper bound for the coordinates of the inner product vectors. All the results are averages of many runs on different random inputs.

In Table 1 we compare the key generation procedure across different schemes with fixed $b = 1000$ and increasing $\ell$. The values show that for practical param-

| l | Paillier[Go] | Paillier[C] | LWE[Go] | LWE[C] | DDH[Go] | DDH[C] |
|---|---|---|---|---|---|---|
| 1 | 0.0267 | 0.0149 | 4.4486 | 7.1557 | 0.0120 | 0.0062 |
| 5 | 0.0759 | 0.0447 | 8.0726 | 7.5744 | 0.0276 | 0.0145 |
| 10 | 0.1390 | 0.0807 | 5.1718 | 7.4500 | 0.0473 | 0.0246 |
| 20 | 0.2638 | 0.1546 | 5.8097 | 8.6669 | 0.0864 | 0.0464 |
| 50 | 0.6434 | 0.3709 | 12.0178 | 12.0974 | 0.2048 | 0.1078 |
| 100 | 1.2741 | 0.7483 | N/A | N/A | 0.4027 | 0.2103 |
| 200 | 2.5452 | 1.4871 | N/A | N/A | 0.7984 | 0.4141 |

**Table 2.** Performance of encryption (in seconds) in inner product schemes w.r.t. vector length $\ell$

eters the generation of keys in inner-product schemes is linearly dependent on conventionality $\ell$. This is in contrast with the dependency on $b$ (not shown in the table), increasing which only mildly increases the generation if at all, assuming it is not extremely large. The table shows that LWE-based schemes are practical only for small parameters. Note a slightly slower performance of the Paillier scheme compared to the DDH-based scheme which is attributed to the need of Gaussian sampling, described in Section 3, and computations being computed in a bigger group, i.e. modular operations are computationally more demanding. In Table 2 similar observations can be done for the encryption process.

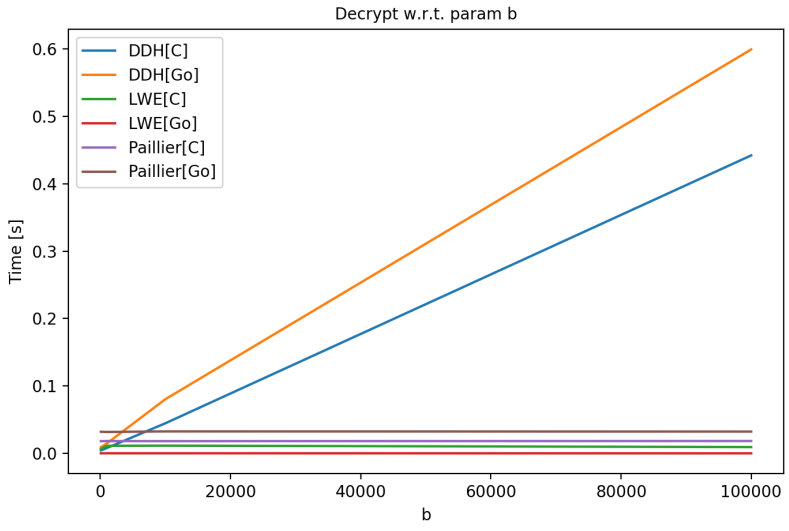



**Fig. 1.** Performance graph of decryption in inner product schemes w.r.t. bound b

The biggest difference of the schemes is demonstrated in Figure 2 measuring the decryption times of the schemes depending on the bound $b$ of the inputs. While the Paillier scheme has only a slight linear increase in computation times when $b$ is increased, DDH-based schemes prove themselves practical only for vectors with a small bound $b$. The latter is contributed to finding a discrete logarithm in its decryption procedure, the performance of which is directly connected with the size of the decrypted value. Interestingly, LWE-based schemes have the fastest decryption. Figure 2 shows the dependency for bounded random vectors.

### 4.2 Decentralized inner-product scheme

Decentralized schemes eliminate the need for the central trusted authority for key generation and derivation. See Section 6 for an application of a decentralized scheme.

| params | KeyGen[D] | KeyGen[Q] | Encrypt[D] | Encrypt[Q] | Decrypt[D] | Decrypt[Q] |
|---|---|---|---|---|---|---|
| b=100, l=1 | 0.0082 | 0.0001 | 0.0016 | 0.0241 | 0.0373 | 0.0910 |
| b=100, l=5 | 0.0087 | 0.0001 | 0.0016 | 0.1104 | 0.0990 | 0.7502 |
| b=100, l=20 | 0.0108 | 0.0002 | 0.0016 | 0.4334 | 0.3273 | 9.8077 |
| b=500, l=1 | 0.0092 | 0.0001 | 0.0016 | 0.0251 | 0.1090 | 0.4986 |
| b=1000, l=1 | 0.0092 | 0.0001 | 0.0016 | 0.0251 | 0.1512 | 1.1859 |
| b=1000, l=20 | 0.0108 | 0.0002 | 0.0016 | 0.4541 | 0.3273 | 92.3084 |

**Table 3.** Performance of the decentralized (D) and quadratic (Q) schemes in GoFE (in seconds)

The implemented decentralized inner-product scheme [15] is based on pairings (SXDH assumption). The results are presented in Table 3. Note that the generation of keys and the encryption process have a better performance as basic inner-product schemes since both are distributed among users and counted only per user. The communication overhead is not included in the measurements. The decryption process involves computing a discrete logarithm as well as performing a pairing operation.

### 4.3 Quadratic scheme

Quadratic schemes are a powerful tool for evaluating more complex functions on encrypted data. Table 3 evaluates the performance of a quadratic scheme [40]. The decryption process turns out to be time-consuming as it requires computing a discrete logarithm and pairing operation. Note that the input value for a discrete logarithm is bigger compared to the inner-product schemes due to the quadratic operations applied on the input vector $x$. We demonstrate in Section 7 that performance is still sufficient for the real-world use cases.

## 5 Privacy-Friendly Prediction of Cardiovascular Diseases

In this section, we demonstrate how FE can enable privacy-enhanced analyses. We show how the risk of general cardiovascular disease (CVD) can be evaluated using only encrypted data.

The demonstrator comprises the following components: Key Server is a central authority component generating keys, Analyses Service is a component to which the user sends encrypted data and obtains the risk evaluation of CVD, and Client component which obtains the public key from the Key Server, encrypts user's data with the public key and sends it to the Service.

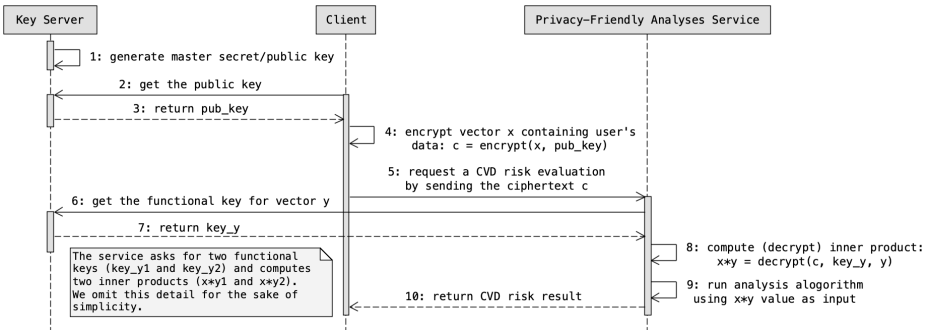



**Fig. 2.** Interactions between CVD demonstrator components

The Framingham heart study [20] followed patients from Framingham, Massachusettes, for many decades starting in 1948. Many multivariable risk algorithms used to assess the risk of specific atherosclerotic cardiovascular disease events have been developed based on the original Framingham study. Algorithms most often estimate the 10-year or 30-year CVD risk of an individual.

The input parameters for algorithms are sex, age, total and high-density lipoprotein cholesterol, systolic blood pressure, treatment for hypertension, smoking, and diabetes status. The demonstrator shows how the risk score can be computed using only the encrypted values of the input parameters. The user specifies the parameters in the Client program; these are encrypted and sent to the Analyses Service component. The service computes the 30-year risk [37] and returns it to the user.

The source code for all three components is available on FENTEC Github account [38]. We use the inner-product FE scheme based on Paillier cryptosystem [5] due to its fast decryption operation. The Client component prepares a vector $x$ which contains the 8 input parameters, which in GoFE looks like:

```
x := data.NewVector([]*big.Int{sex, age, systolicBloodPressure,
totalCholest, hdlCholest, smoker, treatedBloodPressure, diabetic})
```

Framingham risk score algorithms are based on Cox proportional hazards model [17]. Part of it is multiplication of the input parameters by regression factors which are real numbers. In the 30-year algorithm the vector $x$ is multiplied by two vectors (inner-product):

```
y_1 = (0.34362, 2.63588, 1.8803, 1.12673, -0.90941, 0.59397,
0.5232, 0.68602)
y_2 = (0.48123, 3.39222, 1.39862, -0.00439, 0.16081, 0.99858,
0.19035, 0.49756)
```

Regression factors need to be converted into integers because cryptographic schemes operate with integers. This is straight-forward in FE schemes: we multiply factors by the power of 10 to obtain whole numbers. The Client encrypts vector $x$ using public key obtained from the Key Server:

```
ciphertext, err := paillier.Encrypt(x, masterPubKey)
```

The Client then sends ciphertext to the Service. Service beforehand obtained two functional encryption keys from the Key Server: a key to compute the inner-product of $x$ and $y_1$, and a key to compute the inner-product of $x$ and $y_2$. Now it can compute the inner-products:

```
xy_1, err := paillier.Decrypt(ciphertext, key_1, y_1)
xy_2, err := paillier.Decrypt(ciphertext, key_2, y_2)
```

To obtain the risk score the algorithm computes $e^{xy_1-21.29326612}, e^{xy_2-20.12840698}$ followed by $1340 \cdot 1340$ power functions, $1340 \cdot 3$ multiplications, and 1340 additions on the obtained values. For details please refer to [37] or the source code [38]. These operations are executed by the Service and returned to the Client component.

A user thus does not need to know anything about the algorithm to obtain the personal CVD risk score and at the same time the Service does not know anything about the user's parameters (except the inner-products of $x$ with vectors $y_1$ and $y_2$).

However, it has to be noted that the Service does know the risk score. This is one of the main differences with HE. HE computes the encryption of the risk score which is then decrypted by the user (and thus known only by the user).

Paper [13] reports on the implementation of the 10-year CVD risk score using HE. While this approach has a clear advantage of prediction service not knowing the risk score, it is also far less efficient than the approach with FE. In a setup which enables the evaluation of higher degree polynomials (such as 7), one multiplication of ciphertexts requires around 5 seconds on a modern laptop (Intel Core i7-3520M at 2893.484 MHz). Note that higher degree polynomials are needed to approximate the exponential function by a Taylor series. While in the 10-year CVD risk algorithm there is only one evaluation of the exponential function, the 30-year algorithm uses two evaluations. An evaluation of the exponential function in [13] requires more than 30 seconds since computing the

Taylor series of the degree 7 takes more than 30 seconds (the powers of x already require 6 multiplications at 5 seconds each). On the contrary, our FE approach returns result in a matter of milliseconds.

Furthermore, there is a significant communication overhead in HE approach as the ciphertext can grow to roughly one megabyte (16384 coefficients of 512-bit). Communication messages in FE are much smaller – a few kilobytes.

HE approach could be sped up with computing the encryption of only the inner-products (as it is in FE). However, as the prediction service would know only the encryption of the inner-product, the rest of the risk score algorithm would need to be computed at the user's side and would require to move significant parts of the prediction logic to the Client component. In many scenarios, this might not be desirable, especially if the prediction logic is computationally expensive. As a matter of fact, for all services where prediction logic is computationally expensive, the FE approach is far more performant, but at the expense that prediction service knowns the predicted value.

## 6 Londond Underground Anonymous Heatmap

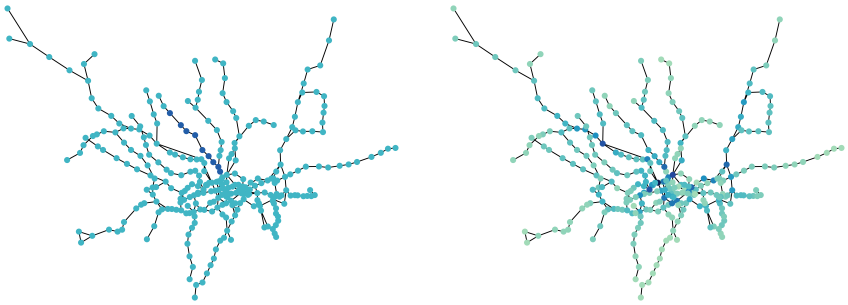
In this section, we demonstrate how traffic heatmap can be generated based on encrypted data. Given the encrypted information about users of the London Underground, our service can measure the traffic density at each particular station. Thus, congestions and potential increases of traffic can be detected while the user data is encrypted and remains private.

DMFCE scheme [15] is used for the demonstration [8]. The scheme allows each user to encrypt the location data in a way that neither the central service nor the other users can know it. The only information that the central service can obtain is the information about all the users, preserving the privacy of each individual. Furthermore, the functional keys needed by the central service are derived in a decentralized manner, without a centralized authority for generating keys. Indeed, functional key parts are provided by the users and then combined together by the central service.

Each user encrypts locally the vector specifying the path that was traveled. The length of the vector is the same as the number of the stations. It consists of 0s and 1s: 1 for stations were the user traveled (see Figure 3a for a visual representation). In GoFE the code looks like:

```
// pathVec[i] is the value of i-th station, label its name,
// c[i] is its encryption
label = station[i]
c[i], _ := client.Encrypt(pathVec[i], label)
```

While we use randomly generated user data for this demonstration, one can easily imagine a smartphone app which tracks the user's path, generates a vector, encrypts it (all operations performed locally), and finally sends it to the central service.



(a) Path of one user (b) Heatmap

**Fig. 3.** Information of one user vs. information the central service obtains.

In the decentralized scheme [15], the FE keys are generated by the users (no trusted authority is needed). The users thus provide a functional key to a central service component. In our case, a functional key for an inner-product vector $y$ of 1s is provided (the vector length is the number of users). This is because the central authority decrypts the sum of all the users that traveled through that station, i.e. a value that can be represented as an inner-product of $y$ and a vector $x$ of 0s and 1s indicating which users traveled through that station. Each user provides a key share:

```
// create a vector of 1s:
vecOfOnes := data.NewConstantVector(numClients, big.NewInt(1))
// keyShares is a vector of all the key shares
keyShares[k], _ := clients[k].GenerateKeyShare(vecOfOnes)
}
```

The central service component collects all the key shares and can now compute (decrypt) the density for each station. The code for this looks like:

```
for i := 0; i < numStations; i++ {
    label := stations[i]
    dec = fullysec.NewDMCFEDecryptor(vecOfOnes, label, ciphers[i],
    keyShares, numClients)
    heatmap[i], _ = dec.Decrypt()
}
```

Using a described approach, a variety of other analysis services can be built on the encrypted data, for example, power consumption of a group of houses in a neighborhood, measurements from IoT devices, etc. In the former case, the power consumption could be encrypted for each hour and sent to the central component. The central component could then compute (decrypt) the overall consumption (across all houses) for each particular hour. Based on such privacy-enhanced

computations various prediction services can be built using only encrypted data. Note that all such applications cannot be build with HE, since the derivation of a functional decryption key is needed for the central service to decrypt the results.

## 7 Neural Networks on Encrypted MNIST Dataset

In the previous two sections, we saw how to implement privacy-friendly predictive services by using efficient FE for inner-products. Using linear functions (inner-products) many efficient machine learning models can be built based on linear regression or linear logistic.

However, linear models in many cases do not suffice. One of such tasks is image classification where linear classifiers mostly achieve significantly lower accuracy compared to the higher-degree classifiers. For example, classifiers for the well-known MNIST dataset where handwritten digits need to be recognized. A linear classifier on MNIST dataset is reported to have 92% accuracy (TensorFlows tutorial [41]), while more complex classifiers achieve over 99% accuracy.

GoFE and CiFEr include a scheme [40] for quadratic multi-variate polynomials which enables computation of quadratic polynomials on encrypted vectors. This enables richer machine learning models and even basic versions of neural networks. We provided a machine learning project [36] to demonstrate how an accurate neural network classifier can be built on the MNIST dataset and how FE can be used to apply a classifier on the encrypted dataset. This means that an entity holding an FE key for a classifier can classify encrypted images, i.e., can classify each image depending on the digit in the encrypted image, but cannot see anything else within the image (for example, some characteristics of the handwriting).

The demonstrator uses the GoFE library and the widely-used machine learning library Tensor-Flow [1]. MNIST dataset consists of 60 000 images of handwritten digits. Each image is a $28 \times 28$ pixel array, where each pixel is represented by its gray level. The model we used is a 2-layer neural network with quadratic function as non-linear activation function. Training of the model needs to be done on unencrypted data, while prediction is done on encrypted images. The images have been presented as 785-coordinate vectors ($28 \cdot 28 + 1$ for bias). We achieved the accuracy of 97%, a result that is reported also in [40]. The decryption of one image (applying the trained model on the encrypted image) takes under 20 seconds.

Similarly, CryptoNets [28], an HE approach for applying neural networks to encrypted data, needs an already trained model. The model they use is significantly more complex than ours (the trained network has 9 layers) and provides an accuracy of 99%. Note that as currently no efficient FE schemes exist for polynomials of degree greater than 2, no such complex models are possible with FE. On the other hand, the execution when using HE approach is significantly slower. Applying the network on encrypted data using CryptoNets takes 570 seconds on a PC with a single Intel Xeon E5-1620 CPU running at 3.5GHz. But note that

applying the network allows executing many predictions simultaneously, if this is needed.

Thus, compared to the FE approach, HE can provide more complex machine learning models and consequently ones with a higher accuracy. Nevertheless, HE has a limitation which is particularly important in the present application. HE can only serve as privacy-friendly outsourcing of computation, while the result of this computation can be decrypted only by the owner of the secret key. FE allows the third party to decrypt the result, in our case the digit in the image, without exposing the image itself. One can easily imagine a more complex FE alert system on encrypted video, where the system detects the danger without violating the privacy of the subjects in the video when there is none. Currently, only primitive versions of such a system are possible as more efficient schemes (in terms of performance and polynomial degree) are needed.

## 8 Conclusions and Future Work

In this paper, we presented the first two fully-fledged functional encryption libraries. The two libraries are implemented in Go and C programming languages and offer an easy-to-use API to various FE schemes. We focused on creating a flexible and efficient implementation to support various use cases. We have demonstrated the practicality by presenting three possible applications of the libraries: an online privacy-friendly predictor of cardiovascular diseases, anonymous traffic heatmap service, and image classification on encrypted data. We compared the solutions that FE has to offer with HE on the latter examples, showing how FE can offer new applications or improve performance by revealing some information. The libraries are filling the gap between academic research of FE schemes and their applications to real-life scenarios. As such they offer a platform for the developers to prototype their products as well as a test place for academic research on FE.

In our future work, we plan to implement further FE schemes, in particular recent multi-client and multi-input schemes which enable a wide range of applications like running queries on encrypted databases, computation over encrypted data streams, and multi-client delegation of computation. Furthermore, we plan to implement and evaluate function-hiding schemes which enable privacy-preserving queries to the prediction services. Also, further optimizations will be applied.

## Acknowledgements

The research was supported, in part, by grant H2020-DS-2017-780108 (FENTEC).

## References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine

learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)
2. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: IACR International Workshop on Public Key Cryptography. pp. 733–751. Springer (2015)
3. Abdalla, M., Catalano, D., Fiore, D., Gay, R., Ursu, B.: Multi-input functional encryption for inner products: function-hiding realizations and constructions without pairings. In: Annual International Cryptology Conference. pp. 597–627. Springer (2018)
4. Agrawal, S., Chase, M.: Fame: fast attribute-based message encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 665–682. ACM (2017)
5. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Annual International Cryptology Conference. pp. 333–362. Springer (2016)
6. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
7. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchangea new hope. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 327–343 (2016)
8. Anonymous heatmap: https://github.com/fentec-project/anonymous-heatmap
9. Apache Milagro Crypto Library: https://github.com/milagro-crypto/amcl
10. Barreto, P.S., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: International Workshop on Selected Areas in Cryptography. pp. 319–331. Springer (2005)
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 514–532. Springer (2001)
12. Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1006–1018. ACM (2016)
13. Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. Journal of biomedical informatics **50**, 234–243 (2014)
14. Boyle, E., Chung, K.M., Pass, R.: On extractability obfuscation. In: Theory of Cryptography Conference. pp. 52–73. Springer (2014)
15. Chotard, J., Sans, E.D., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 703–732. Springer (2018)
16. Cloudflare implementation of Barreto-Naehrig bilinear pairings: https://github.com/cloudflare/bn256
17. Cox, D.R.: Regression models and life-tables. Journal of the Royal Statistical Society: Series B (Methodological) **34**(2), 187–202 (1972)
18. De Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-lwe encryption. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. pp. 339–344. EDA Consortium (2015)
19. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Annual Cryptology Conference. pp. 40–56. Springer (2013)

20. Dagostino, R.B., Vasan, R.S., Pencina, M.J., Wolf, P.A., Cobain, M., Massaro, J.M., Kannel, W.B.: General cardiovascular risk profile for use in primary care. Circulation **117**(6), 743–753 (2008)
21. FENTEC project Github accunt: https://github.com/fentec-project
22. de Freitas Aranha, D., Gouvea, C.P.L., Markmann, T.: RELIC. https://github.com/dis2/bls12, https://github.com/dis2/bls12
23. Fuentes-Castaneda, L., Knapp, E., Rodríguez-Henríquez, F.: Faster hashing to g2. In: International Workshop on Selected Areas in Cryptography. pp. 412–430. Springer (2011)
24. Functional encryption library in C: https://github.com/fentec-project/CiFEr
25. Functional encryption library in Go: https://github.com/fentec-project/gofe
26. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM Journal on Computing **45**(3), 882–929 (2016)
27. Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure attribute based encryption from multilinear maps. IACR Cryptology ePrint Archive **2014**, 622 (2014)
28. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning. pp. 201–210 (2016)
29. Göttert, N., Feller, T., Schneider, M., Buchmann, J., Huss, S.: On the design of hardware building blocks for modern lattice-based encryption schemes. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 512–529. Springer (2012)
30. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 89–98. Acm (2006)
31. Knuth, D., Yao, A.: Algorithms and complexity: New directions and recent results, chapter the complexity of nonuniform random number generation (1976)
32. Langley, A., Burke, K., Valsorda, F., Symonds, D.: Package bn256. https://godoc.org/golang.org/x/crypto/bn256 (2012), https://godoc.org/golang.org/x/crypto/bn256
33. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 568–588. Springer (2011)
34. Lynn, B.: The Pairing Based Cryptography library. https://crypto.stanford.edu/pbc/, https://crypto.stanford.edu/pbc/
35. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–23. Springer (2010)
36. Neural network on encrypted data: https://github.com/fentec-project/neural-network-on-encrypted-data
37. Pencina, M.J., D'Agostino Sr, R.B., Larson, M.G., Massaro, J.M., Vasan, R.S.: Predicting the thirty-year risk of cardiovascular disease: the framingham heart study. Circulation **119**(24), 3078 (2009)
38. Private prediction analyses: https://github.com/fentec-project/private-analyses
39. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 34 (2009)
40. Sans, E.D., Gay, R., Pointcheval, D.: Reading in the dark: Classifying encrypted digits with functional encryption. IACR Cryptology ePrint Archive **2018**, 206 (2018)

41. Tensorflow tutorial: https://www.tensorflow.org/tutorials#evaluating_our_model
42. The GNU Multiple Precision Arithmetic Library: https://gmplib.org
43. Waters, B.: A punctured programming approach to adaptively secure functional encryption. In: Annual Cryptology Conference. pp. 678–697. Springer (2015)
44. Zheng, Q., Xu, S., Ateniese, G.: Vabks: verifiable attribute-based keyword search over outsourced encrypted data. In: IEEE INFOCOM 2014-IEEE Conference on Computer Communications. pp. 522–530. IEEE (2014)