D6.5 Performance Analyses 2

Document Identification			
Status	Final	**Due Date**	31/7/2020
Version	1.0	**Submission Date**	28/7/2020

Related WP	WP6	**Document Reference**	D6.5
Related Deliverable(s)	D6.2, D6.3, D6.4	**Dissemination Level(*)**	PU
Lead Participant	XLAB	**Lead Author**	Tilen Marc (XLAB)
Contributors	FUAS	**Reviewers**	Yolan Romailler (KUD) Clement Gentilucci (FUAS)

Keywords:
Functional Encryption, Implementation, Performance

This document is issued within the frame and for the purpose of the FENTEC project. This project has received funding from the European Union's Horizon2020 under Grant Agreement No. 780108. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

(*) Dissemination level.-PU: Public, fully open, e.g. web; CO: Confidential, restricted under conditions set out in Model Grant Agreement; CI: Classified, Int = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.

Document Information

List of Contributors	
Name	Partner
Tilen Marc	XLAB
Miha Stopar	XLAB

Document History			
Version	Date	Change editors	Changes
0.1	22/5/2020	Tilen Marc (XLAB)	Skeleton
0.2	10/6/2020	Tilen Marc (XLAB)	Benchmarks of new schemes
0.3	21/6/2020	Tilen Marc (XLAB)	Performance optimizations
0.4	2/7/2020	Miha Stopar (XLAB)	Editing
1.0	28/7/2020	Miha Stopar (XLAB)	Finalization

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable Leader	Miha Stopar (XLAB)	28/7/2020
Technical Manager	Michel Abdalla (ENS)	28/7/2020
Quality Manager	Diego Esteban (ATOS)	28/7/2020
Project Coordinator	Francisco Gala (ATOS)	28/7/2020

Table of Contents

List of Figures

List of Acronyms

Acronym	Description
ABE	Attribute Based Encryption
API	Application Programming Interface
CP-ABE	Ciphertext Policy Attribute-Based Encryption
CVD	Cardiovascular Diseases
DCR	Decisional Composite Residuosity
DDH	Decisional Diffie-Hellman
DMCFE	Decentralized Multi-Client Functional Encryption
FE	Functional Encryption
HE	Homomorphic Encryption
IoT	Internet of Things
KP-ABE	Key Policy Attribute-Based Encryption
MSP	Monotone Span Program
LWE	Learning With Errors
RLWE	Ring-Learning With Errors

Executive Summary

In this deliverable D6.5 “Performance Analyses 2”, we discuss the performance of the two functional encryption libraries developed in FENTEC. We compare both libraries, GoFE and CiFEr, as well as the performance of different schemes between each other. We summarize the optimizations that have been implemented in the first half of the last year of the project. Additionally, we provide an evaluation of the integration of GoFE into two real-world scenarios and compare the performance of the functional encryption approach against the homomorphic encryption approach. We demonstrate that functional encryption is practical and can be used in real-world applications. This document is an extension of D6.4 “Performance Analyses 1”, where the performance of the initial set of implemented schemes has been discussed.

1 Introduction

The main objective of Work Package 6 is to implement state-of-the-art functional encryption libraries. Deliverable D6.3 “Final Functional Encryption Toolset API” provides an overview of the two functional encryption libraries – GoFE, written in the Go programming language; and CiFEr, written in the C programming language. Both libraries offers an API to state-of-the-art inner-product, quadratic, and attribute-based encryption schemes. This deliverable presents the performance evaluation of the two libraries.

1.1 Purpose of the Document

The goal of this deliverable is to present the benchmark results for GoFE and CiFEr. Thus, the reader can compare the differences between the two libraries, as well as the differences between the various schemes (for example, GoFE and CiFEr provide an implementation of various inner-product schemes).

The document can serve as a helping point for choosing between different schemes – while some schemes excel for example at the encryption phase, some others are far more performant at the decryption phase.

The performance results and more comparisons with a homomorphic encryption approach have been provided in the paper that has been accepted at the ESORICS [1] conference.

1.2 Structure of the Document

This deliverable is structured as follows. Section 2 provides the benchmark results for GoFE and CiFEr libraries. Section 3 describes performance optimizations for GoFE and CiFEr libraries. Section 4 discusses the performance of the GoFE library in the privacy-enhanced health analysis use case and in the machine learning application. The deliverable concludes with a summary and an outlook in Section 5.

2 Benchmarks

In this section, we give a performance evaluation of implemented schemes. We compare the benefits and downsides of the schemes and discuss their practicality for the possible uses. The schemes are implemented with the goal of being flexible and having an easy-to-use API. For this reason, 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. All of the benchmarks were performed on an Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz.

2.1 Inner-Product Schemes

Recall that an inner-product functional encryption scheme allows the encryption of a vector $x \in \mathbb{Z}^\ell$ and the key derivation of a functional key sk_y for a vector y, where the vectors x and y are totally independent. The decryption of the ciphertext of x under the functional key sk_y only reveals the inner-product $x \cdot y$ and nothing more.

Each inner-product scheme comprises five parts: setup, master key generation, encryption, functional (inner-product) key derivation, and decryption. In what follows, we give performance results of inner-product schemes based on different security assumptions.

We demonstrate the efficiency of the schemes depending on the parameters ℓ (length of the encrypted vectors) and b (upper bound for the coordinates of the inner-product vectors). All the results are averages of many runs on different random inputs. Note that the implementation of the schemes enables a user to choose the level of security. However, by increasing the level of security, the performance of the scheme is lowered. In the benchmarks, we tested all the schemes with parameters chosen to be considered safe by various security standards (2048-bit security).

In Tables 1, 2, 3, 4 and Figures 1, 2, 3, 4, we compare the operations across different schemes with fixed $b = 1000$ and increasing vector length ℓ.

ℓ	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
1	0.1549	0.0657	12.9523	7.3909	0.0080	0.0041
5	0.5612	0.2938	62.1945	46.2466	0.0402	0.0204
10	1.0600	0.5756	122.7627	74.8795	0.0840	0.0411
20	2.0551	1.1384	266.5059	196.6151	0.1584	0.0849
50	5.0520	2.8410	878.3684	559.6070	0.3954	0.2055
100	10.0916	5.7032	N/A	N/A	0.7829	0.4149
200	20.0883	11.3700	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 first operation that needs to be executed is the setup operation. In the DDH based schemes as well as in the Paillier schemes, setup consists of constructing a mathematical group in which the operations will be performed. The major part of this procedure is finding large safe prime numbers. For all the tests in this section this is a 2048 bit prime number in the case of DDH and modular arithmetics based schemes, and two 1024 bit prime numbers in the case of Paillier scheme. It takes on average 72.77s to generate a

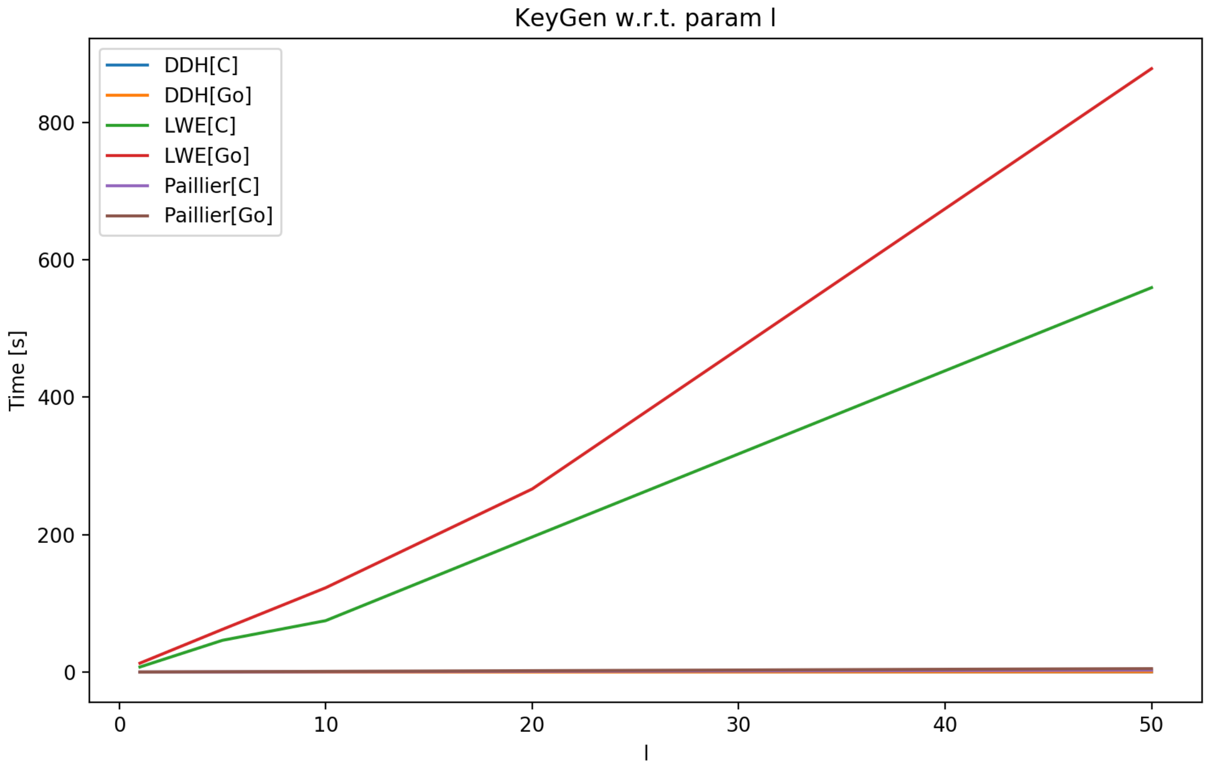

Figure 1: Performance of key generation (in seconds) in inner-product schemes w.r.t. vector length ℓ

ℓ	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
1	< 0.0001	< 0.0001	0.0001	0.0001	< 0.0001	< 0.0001
5	< 0.0001	< 0.0001	0.0003	0.0001	< 0.0001	< 0.0001
10	< 0.0001	< 0.0001	0.0003	0.0001	< 0.0001	< 0.0001
20	< 0.0001	< 0.0001	0.0007	0.0002	< 0.0001	< 0.0001
50	< 0.0001	< 0.0001	0.0022	0.0002	< 0.0001	< 0.0001
100	< 0.0001	< 0.0001	N/A	N/A	< 0.0001	< 0.0001
200	0.0001	< 0.0001	N/A	N/A	0.0001	< 0.0001

Table 2: Performance of key derivation (in seconds) in inner-product schemes w.r.t. vector length ℓ

ℓ	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
1	0.0796	0.0461	4.4148	6.5212	0.0120	0.0062
5	0.2389	0.1389	5.5039	6.8358	0.0276	0.0145
10	0.4367	0.2528	6.3218	7.6660	0.0473	0.0246
20	0.8357	0.4840	7.2797	8.9215	0.0864	0.0464
50	2.0245	1.1751	7.8941	12.6611	0.2048	0.1078
100	4.0087	2.3266	N/A	N/A	0.4027	0.2103
200	7.8847	4.6275	N/A	N/A	0.7984	0.4141

Table 3: Performance of encryption (in seconds) in inner-product schemes w.r.t. vector length ℓ

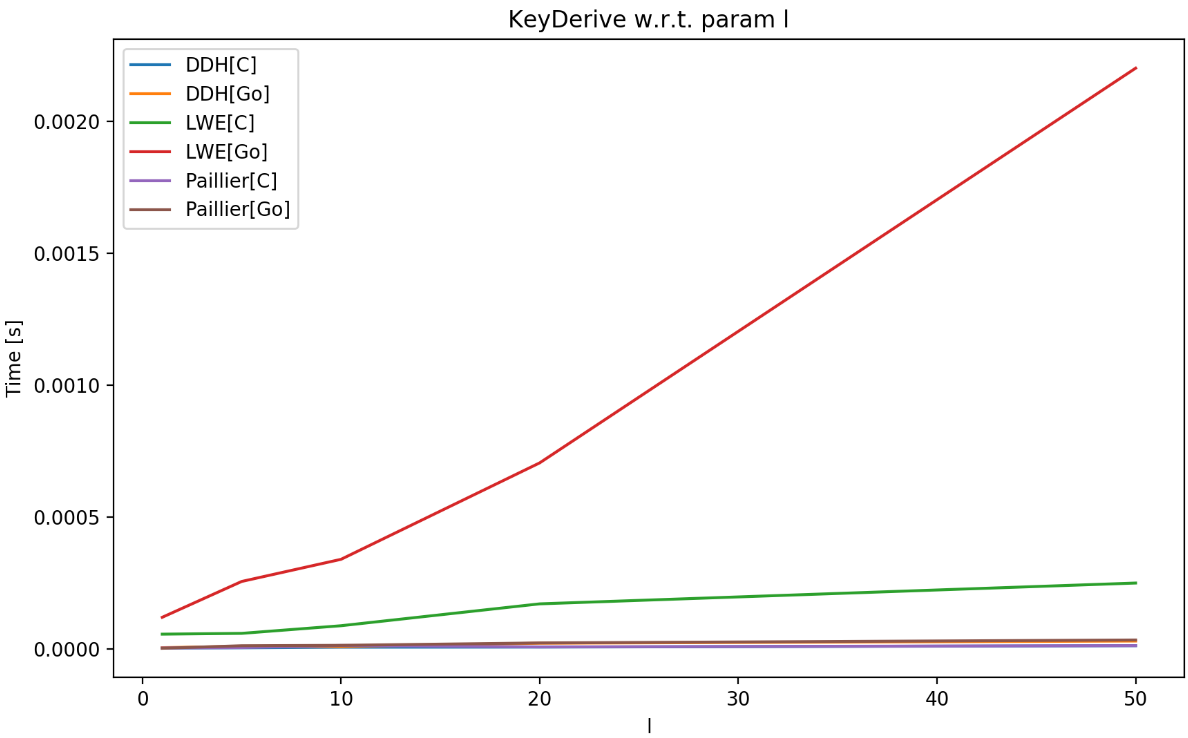

Figure 2: Performance of key derivation (in seconds) in inner-product schemes w.r.t. vector length ℓ

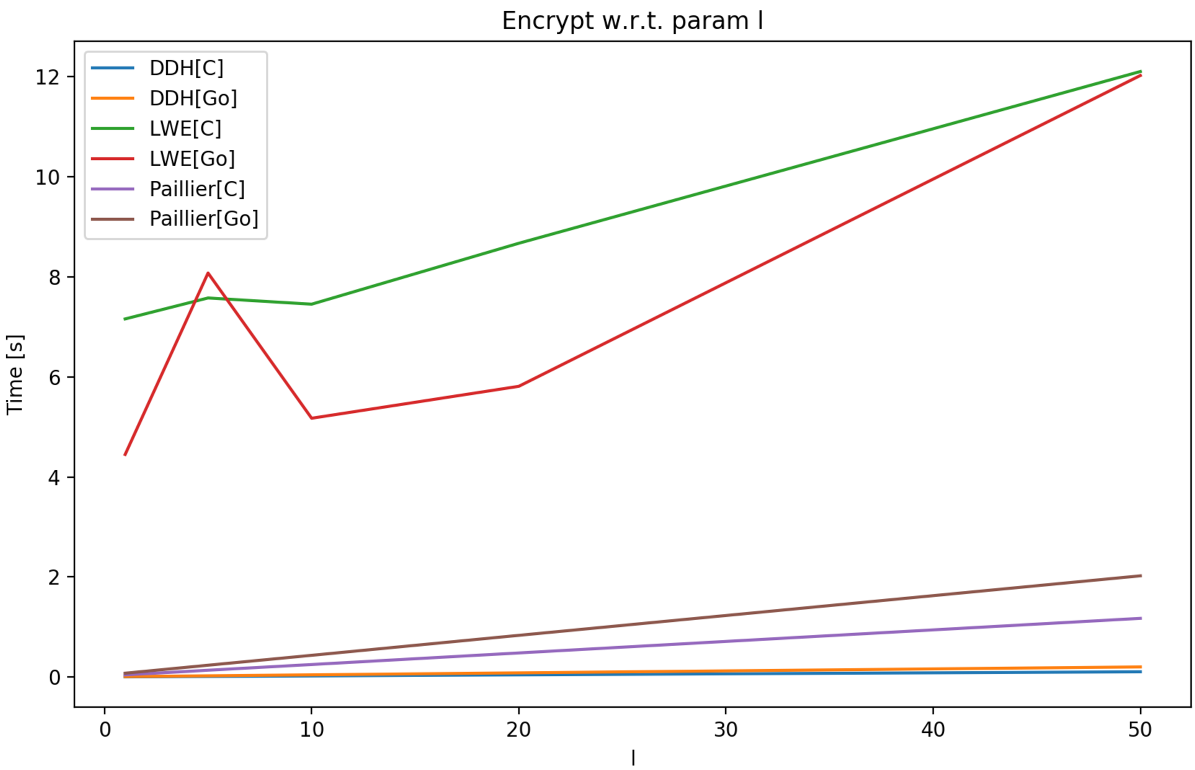

Figure 3: Performance of encryption (in seconds) in inner-product schemes w.r.t. vector length ℓ

ℓ	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
1	0.0348	0.0183	0.0001	0.0118	0.0138	0.0071
5	0.0334	0.0187	0.0002	0.0095	0.0182	0.0127
10	0.0343	0.0194	0.0001	0.0072	0.0196	0.0146
20	0.0358	0.0194	0.0001	0.0076	0.0220	0.0209
50	0.0417	0.0214	0.0015	0.0222	0.0264	0.0259
100	0.0509	0.0257	N/A	N/A	0.0342	0.0351
200	0.0715	0.0322	N/A	N/A	0.0478	0.0628

Table 4: Performance of decryption (in seconds) in inner-product schemes w.r.t. vector length ℓ

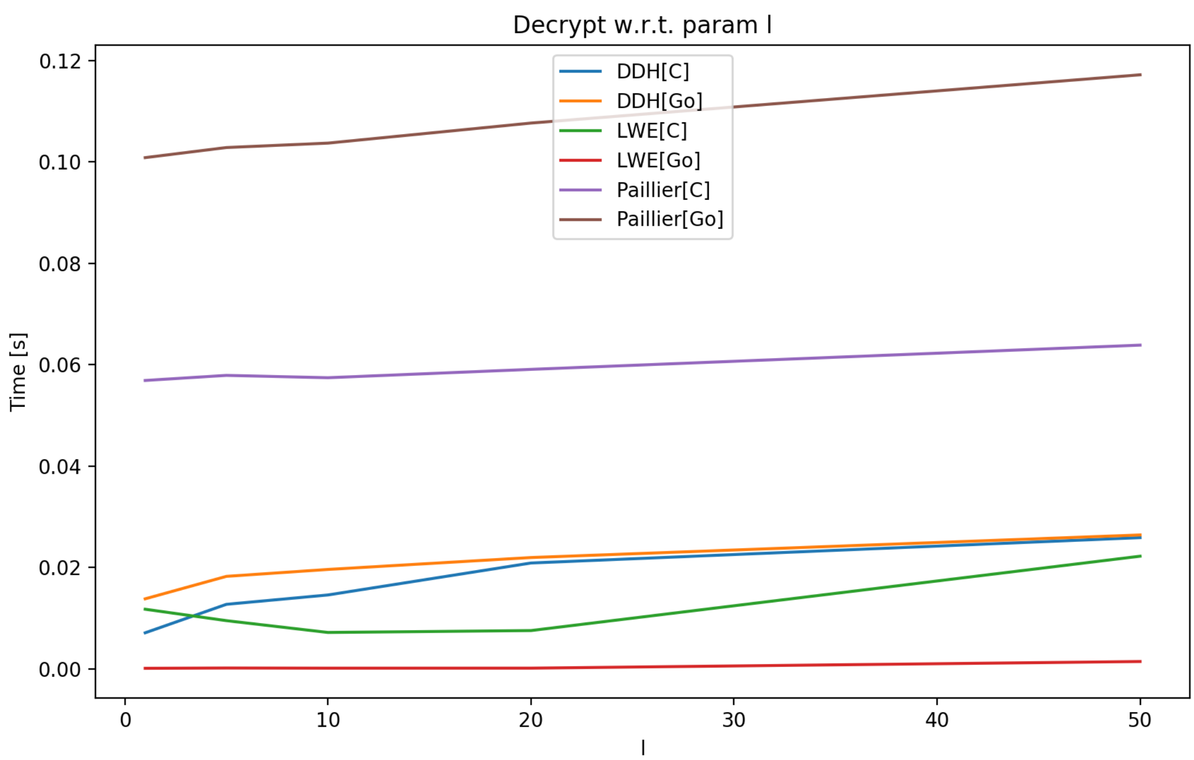

Figure 4: Performance of decryption (in seconds) in inner-product schemes w.r.t. vector length ℓ

2048 bit safe prime number in GoFE (similar time in CiFEr) and on average 5.49s to generate a product of two 1024 bits safe prime numbers. In the case of DDH schemes, this can be avoided using precomputed primes (see Section 3). Note that this cannot be done in the case of Paillier scheme. In the case of the LWE based schemes, a big uniformly random matrix needs to be generated which takes on average 256.88s in GoFE.

We can see that the most time-consuming operations are setup and key generation. However, these operations are executed only once (at the deployment of the system) and do not affect the performance of the system. Key derivation is executed every time the functional decryption keys needs to be derived, but the complexity and execution times are negligible. Encryption and decryption operations are linearly dependent on the parameter ℓ. The performance of encryption and decryption operations varies heavily – some schemes are highly efficient at the encryption phase, others at the decryption phase. The choice of the scheme should thus be based on the use case requirements. It can be observed 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, and computations being computed in a bigger group, i.e. modular operations are computationally more expensive. Similar observations can be made for the encryption process.

In Tables 5, 6, 7, 8 and Figures 5, 6, 7, 8, we compare the operations across different schemes with fixed $\ell = 1$ and increasing b. The setup procedure has in practice an equivalent complexity independently of the bound.

b	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
100	0.0644	0.0243	5.6113	3.2190	0.0086	0.0042
1000	0.0644	0.0240	12.0923	8.5338	0.0080	0.0042
10000	0.0655	0.0240	12.8254	6.9244	0.0082	0.0043
100000	0.0657	0.0240	15.1264	7.5691	0.0086	0.0041
1000000	0.0641	0.0241	15.8633	8.3038	0.0088	0.0042
10000000	0.0652	0.0248	15.2363	7.8872	0.0082	0.0042

Table 5: Performance of key generation (in seconds) in inner-product schemes w.r.t. parameter b

b	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
100	< 0.0001	< 0.0001	0.0002	< 0.0001	< 0.0001	< 0.0001
1000	< 0.0001	< 0.0001	0.0001	0.0001	< 0.0001	< 0.0001
10000	< 0.0001	< 0.0001	0.0001	0.0001	< 0.0001	< 0.0001
100000	< 0.0001	< 0.0001	0.0002	0.0029	< 0.0001	< 0.0001
1000000	< 0.0001	< 0.0001	0.0001	0.0031	< 0.0001	< 0.0001
10000000	< 0.0001	< 0.0001	0.0001	0.0032	< 0.0001	< 0.0001

Table 6: Performance of key derivation (in seconds) in inner-product schemes w.r.t. parameter b

The biggest difference of the schemes can be seen in Table 8 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 observation is attributed to the computation of a discrete logarithm in its decryption procedure, the performance of which is directly connected to the size of the decrypted value. Interestingly, LWE-based schemes have the fastest decryption. The results indicate that the Paillier

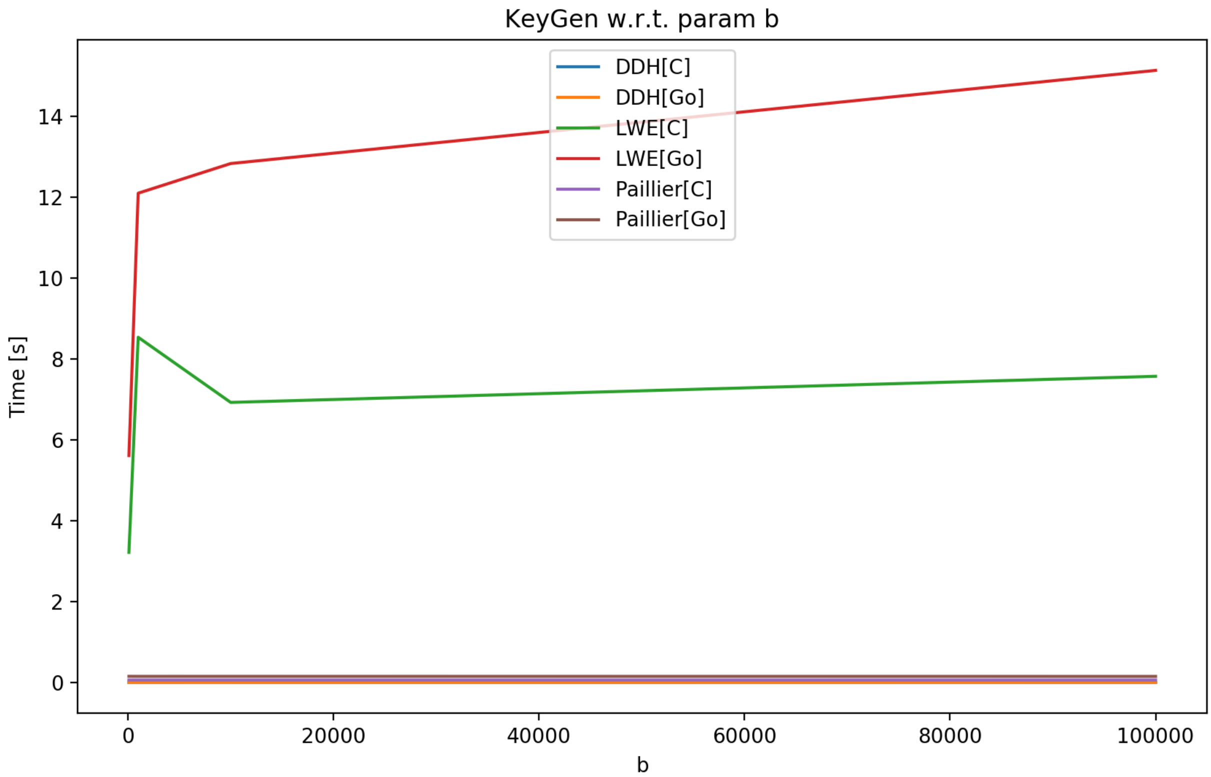

Figure 5: Performance of key generation (in seconds) in inner-product schemes w.r.t. parameter b

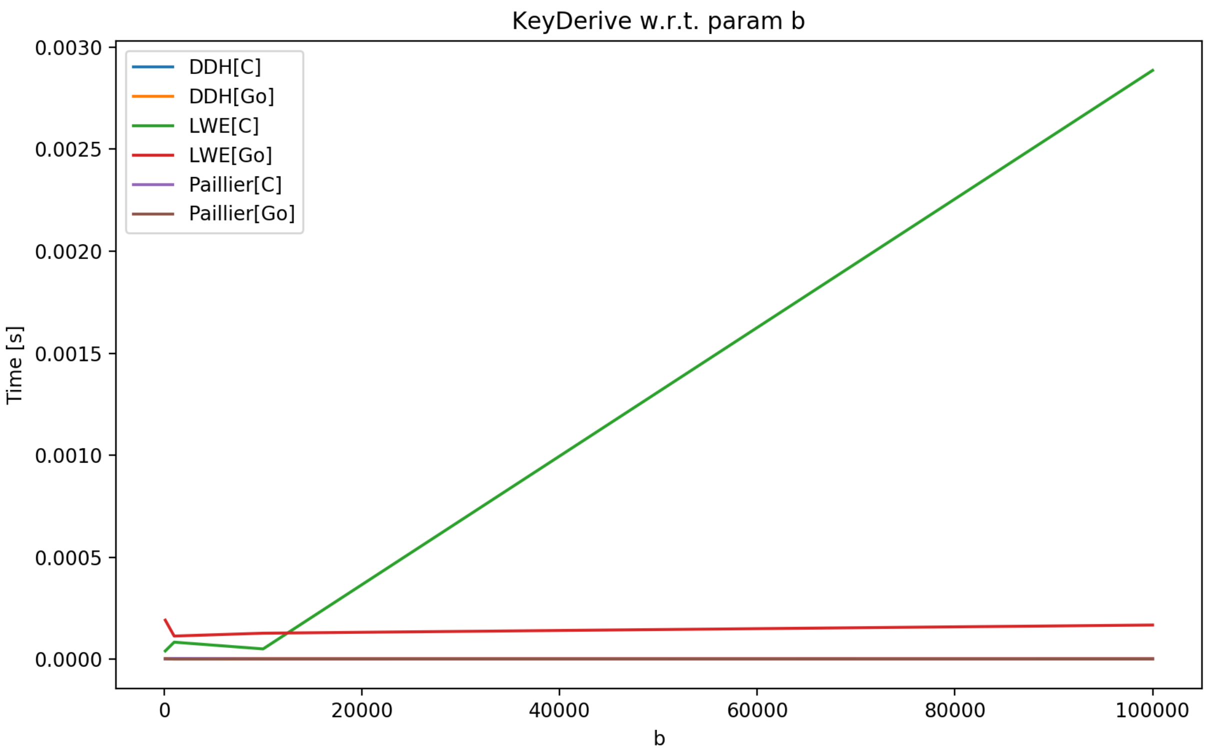

Figure 6: Performance of key derivation (in seconds) in inner-product schemes w.r.t. parameter b

b	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
100	0.0253	0.0146	2.3037	3.6163	0.0128	0.0061
1000	0.0252	0.0146	4.3737	6.7322	0.0118	0.0060
10000	0.0258	0.0148	8.0981	6.9698	0.0119	0.0064
100000	0.0253	0.0148	5.7783	7.2934	0.0133	0.0063
1000000	0.0254	0.0148	4.8875	7.4155	0.0125	0.0061
10000000	0.0255	0.0149	5.6765	6.9367	0.0122	0.0062

Table 7: Performance of encryption (in seconds) in inner-product schemes w.r.t. parameter b

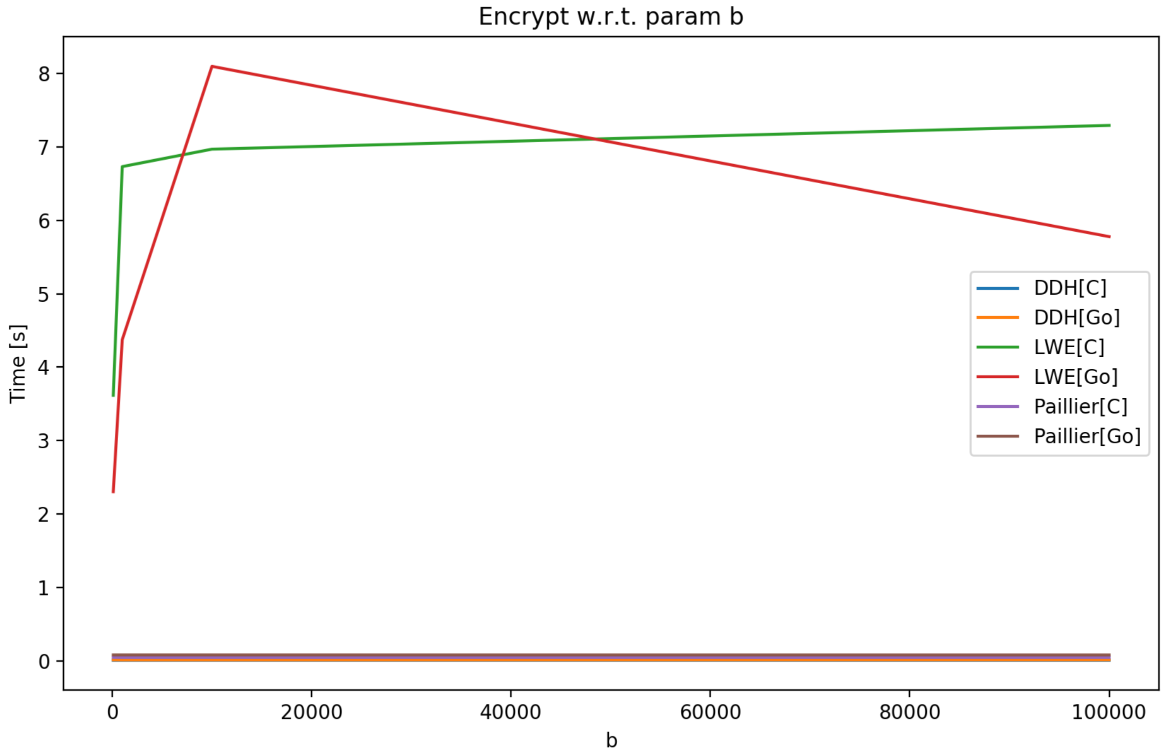

Figure 7: Performance of encryption (in seconds) in inner-product schemes w.r.t. parameter b

b	Paillier[Go]	Paillier[C]	LWE[Go]	LWE[C]	DDH[Go]	DDH[C]
100	0.0321	0.0181	0.0001	0.0065	0.0091	0.0044
1000	0.0318	0.0183	0.0002	0.0113	0.0147	0.0079
10000	0.0326	0.0182	0.0002	0.0115	0.0803	0.0446
100000	0.0323	0.0184	0.0001	0.0094	0.5992	0.4420
1000000	0.0324	0.0182	0.0001	0.0102	6.2379	4.8445
10000000	0.0320	0.0185	0.0001	0.0092	63.9508	40.4848

Table 8: Performance of decryption (in seconds) in inner-product schemes w.r.t. parameter b

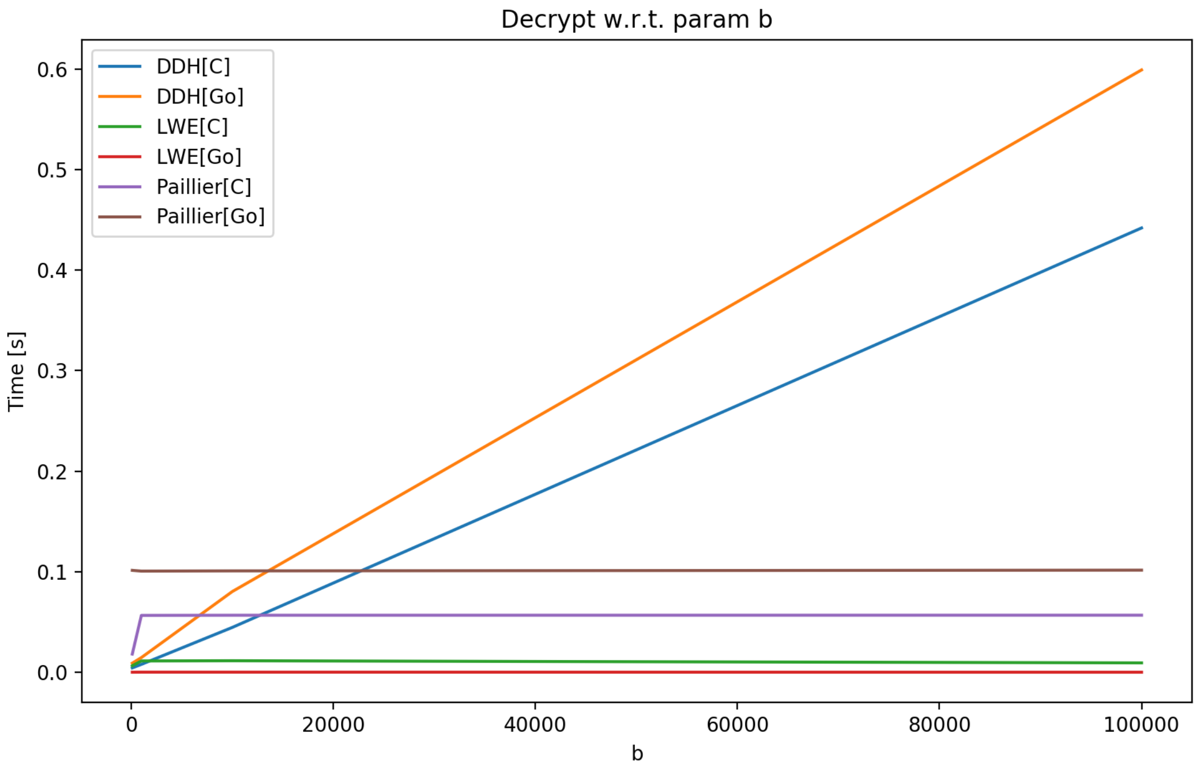

Figure 8: Performance of decryption (in seconds) in inner-product schemes w.r.t. parameter b

scheme is to be preferred unless the bound and dimensionality of the input can be bounded or the use case guarantees the resulting inner-product to be small. In this case, DDH based schemes seem to excel, while LWE based schemes are advised only for small parameters when fast decryption times justify slow key generation and encryption or if quantum security is needed.

2.2 Decentralized Inner-Product Scheme

Recall that an inner-product functional encryption scheme allows the encryption of a vector $x \in \mathbb{Z}^{\ell}$ and the derivation of a functional key sk_y for a vector y, where the vectors x and y are totally independent. The decryption of the ciphertext of x under the functional key sk_y only reveals the inner-product $x \cdot y$ and nothing more.

In many practical cases, the data that needs to be encrypted comes from multiple sources. Basic inner-product schemes demand that the data is encrypted by a single client. This problem is solved by the Multi-Client Inner-Product Encryption (MCIPE) which allows multiple sources of data to encrypt separately. Such schemes depend on a trusted third party to delegate keys to the clients.

In many scenarios, the latter assumption is not acceptable. Hence, decentralized schemes were developed to eliminate the need for the central trusted authority for key generation. The first such scheme was developed in [6] and is implemented in GoFE and CiFEr.

Recently, a new approach to decentralization was developed in [3]. This approach allows to take an arbitrary (centralized) MCIPE scheme that satisfies certain security properties and turns it into a secure decentralized scheme. GoFE and CiFEr provides the API to a decentralized scheme defined in [3] too. It uses the MCIPE scheme based on the DDH assumption in Z_p^* group. We now present the performance results depending on various parameters. The schemes depend on the parameters ℓ (length of the encrypted vectors) and b (upper bound for the coordinates of the inner-product vectors). All the results are averages of many runs on different random inputs.

While the scheme from [6] is based on a pairing group which is fixed in advance, the implementation of the scheme from [3] allows to choose the size of the group. The latter allows the user to adjust the security. To benchmark the scheme the group is chosen to have p elements where p is a 2048 bit prime number, which is a standard choice for a minimum of 128 bit security.

2.2.1 Benchmarking based on parameter ℓ

We compare the performance of the first DMCFE scheme from [6] with the newly developed one from [3]. In Tables 9, 10, 11 we compare both schemes running on the same random inputs with fixed $b = 1000$ and increasing ℓ. The key generation procedure in the case of the decentralized schemes differs from the usual procedure since it is distributed among clients. For this reason, it is a two-round procedure where each client firstly creates its own public key and then sends the data to other clients. After receiving the data from others, it finalizes its secret keys in the second round of computation. Note that all the algorithms besides decryption are distributed among clients hence they are also measured per client.

The times needed to perform the setup operation of the schemes is not presented in the tables since it takes less than a millisecond independently of the parameters ℓ and b.

ℓ	Dec. DDH scheme [3]	DMCFE [6]
1	6.71	0.189
5	5.826	0.207
10	5.73	0.241
20	5.623	0.196
50	5.615	0.204
100	5.949	0.207

(a) First round of key generation

ℓ	Dec. DDH scheme [3]	DMCFE [6]
1	11.777	0.0
5	34.351	0.827
10	62.124	1.849
20	119.404	3.802
50	296.097	9.744
100	600.541	19.667

(b) Second round of key generation

Table 9: Performance of algorithms (in milliseconds) in decentralized inner-product schemes w.r.t. vector length ℓ

2.2.2 Benchmarking based on parameter b

In Tables 12, 13, 14, we compare the performance of both schemes with fixed $\ell = 10$ and increasing b.

2.2.3 Interpretation

Performance analysis suggests that the decentralized schemes are practical enough for many use cases. This was also demonstrated in ESORICS paper [12] about GoFE and CiFEr where a demonstration of using DMCFE scheme for creating anonymous heatmaps was presented. Comparing the two schemes, in

ℓ	Dec. DDH scheme [3]	DMCFE [6]
1	0.003	2.065
5	0.006	4.583
10	0.012	4.601
20	0.019	4.559
50	0.032	4.584
100	0.083	4.639

(a) FE key derivation

ℓ	Dec. DDH scheme [3]	DMCFE [6]
1	23.7	0.552
5	22.413	0.563
10	22.404	0.563
20	22.353	0.556
50	22.68	0.577
100	22.962	0.556

(b) Encryption

Table 10: Performance of algorithms (in milliseconds) in decentralized inner-product schemes w.r.t. vector length ℓ

ℓ	Dec. DDH scheme [3]	DMCFE [6]
1	30.624	45.02
5	89.26	114.624
10	147.284	115.572
20	270.756	207.103
50	636.517	205.73
100	1353.023	374.925

(a) Decryption

Table 11: Performance of algorithms (in milliseconds) in decentralized inner-product schemes w.r.t. vector length b

b	Dec. DDH scheme [3]	DMCFE [6]
10	6.314	0.468
100	6.116	0.188
1000	5.73	0.241
10000	7.05	0.191

(a) First round of key generation

b	Dec. DDH scheme [3]	DMCFE [6]
10	68.653	1.987
100	73.164	1.846
1000	62.124	1.849
10000	81.388	1.82

(b) Second round of key generation

Table 12: Performance of algorithms (in milliseconds) in decentralized inner-product schemes w.r.t. bound b

b	Dec. DDH scheme [3]	DMCFE [6]
10	0.016	4.64
100	0.022	4.71
1000	0.012	4.601
10000	0.022	4.555

(a) FE key derivation

b	Dec. DDH scheme [3]	DMCFE [6]
10	24.239	0.575
100	29.998	0.573
1000	22.404	0.563
10000	26.606	0.559

(b) Encryption

Table 13: Performance of algorithms (in milliseconds) in decentralized inner-product schemes w.r.t. bound b

most of the procedures, DMCFE scheme with pairings from [6] is faster, since the algorithms used are simpler but depending on a theoretically wider cryptographic assumption. This is particularly important in the key generation operation, since the secret keys generated in the decentralized DDH scheme from [3] are much bigger in size. Performance difference is reflected in the measured time, but would also be noted in memory consumption. Nevertheless, in many practical cases the bound on the inputs can be relatively big, hence the decryption is a bottleneck. Since operations in a $\mathbb{Z}_p^*$ group are slightly faster than in a pairing group, decentralized DDH scheme from [3] is preferred in this case.

2.3 Function Hiding Inner-Product Schemes

FE allows to decrypt a value $f(x)$ without revealing the encrypted data x. In certain scenarios, it is preferred that the function f also remains hidden to the decryptor. For example, if FE is used to predict values based

b	Dec. DDH scheme [3]	DMCFE [6]
10	124.573	9.856
100	132.616	20.372
1000	147.284	115.572
10000	404.663	915.511

(a) Decryption

Table 14: Performance of algorithms (in milliseconds) in decentralized inner-product schemes w.r.t. bound b

on private data, the provider of the prediction might not want to reveal the model used. Recently, a few inner-product FE schemes were developed to allow this functionality. We have implemented three such schemes in GoFE and CiFEr since they offer different functionality.

Probably the simplest function hiding inner-product scheme was presented in [11], together with possible use cases. In [7], a more sophisticated function hiding scheme that allows encryption from multiple sources with the presence of a central authority was introduced. Moreover, as part of developing a new public key quadratic FE scheme in [8], a function hiding inner-product scheme which allows certain public key style encryption was developed. In what follows, we compare the performance of these three schemes.

As in the case of decentralized inner-product FE schemes, the performance depends on the bound of the absolute values of inputs b and the length of input vectors ℓ. Thus, we perform different measurements based on these parameters. In the case of [7] which supports inputs from different sources, we evaluate it as if each input coordinate is encrypted by a different source, where there are ℓ sources and their times are summed up.

2.3.1 Benchmarking based on parameter ℓ

In Tables 15 and 16, we compare the performance of the three schemes with fixed $b = 1000$ and increasing ℓ. As in the previous section, the times needed to perform the *Setup* of the schemes is not presented in the tables since it takes less than a millisecond independently of the parameters ℓ and b.

ℓ	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
1	0.842	3.624	1.597
5	1.359	7.179	2.118
10	4.418	12.199	4.023
20	12.12	20.501	5.641
50	127.344	44.043	15.763
100	888.525	93.734	25.424
200	5883.28	167.13	44.579

(a) Key generation

ℓ	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
1	0.557	7.397	4.138
5	1.349	25.214	5.554
10	2.676	46.167	8.827
20	7.159	94.95	15.01
50	73.361	220.075	37.182
100	400.195	457.761	70.521
200	3042.225	875.552	128.012

(b) FE key derivation

Table 15: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length ℓ

ℓ	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
1	1.304	1.854	1.925
5	3.771	7.223	2.013
10	6.909	13.258	2.933
20	13.262	29.888	5.002
50	33.832	65.235	11.635
100	63.905	138.579	22.003
200	129.491	262.719	39.82

(a) Encryption

ℓ	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
1	44.066	67.863	52.23
5	129.184	192.469	134.857
10	135.723	238.706	143.605
20	249.66	474.407	284.614
50	350.541	842.555	339.756
100	530.436	1630.319	598.154
200	675.452	3051.101	673.243

(b) Decryption

Table 16: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length ℓ

2.3.2 Benchmarking based on parameter b

In Table 17 and 18, we compare the performance of both schemes with fixed $\ell = 10$ and increasing bound b. All results are obtained as averages of repeatedly running the algorithms on random inputs.

b	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
10	4.013	10.861	4.209
100	2.65	9.928	2.904
1000	4.418	12.199	4.023
10000	2.637	9.923	2.975

(a) Key generation

b	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
10	3.157	44.179	9.321
100	2.679	44.237	10.485
1000	2.676	46.167	8.827
10000	2.691	43.473	9.098

(b) FE key derivation

Table 17: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length b

2.3.3 Interpretation

Function hiding schemes offer additional functionality on top of the inner-product schemes. The performance is notably worse than in the usual inner-product schemes, but we believe that it is still practical for many real-world scenarios. In most cases, the newest public key type encryption scheme from [8] outperforms the other two. The main reason for this is that both, [11] and [7], require computing a rather big matrix over $\mathbb{Z}_p^*$ and inverting it, while the scheme from [8] avoids such computations. This means that

b	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
10	6.948	13.306	2.997
100	6.817	13.082	3.649
1000	6.909	13.258	2.933
10000	6.823	13.22	4.81

(a) Encryption

b	FHIPE [11]	FH Multi IPE [7]	PK FHIPE [8]
10	23.934	145.993	30.436
100	31.428	154.137	45.297
1000	135.723	238.706	143.605
10000	1096.941	1092.892	1121.35

(b) Decryption

Table 18: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length b

the scheme from [8] is the preferred choice in the case of data coming from one source, while the scheme from [11] needs to be used in the multi-input scenario. Note that all schemes include computing a discrete logarithm in the decryption phase, which is the main bottleneck if the output cannot be guaranteed to be small.

2.4 Quadratic Scheme

Recall that a quadratic FE scheme allows a user to encrypt vectors x, y, and independently derive a FE key depending on a matrix F, such that with the FE key one can decrypt the value $y^T F x$ without revealing x or y. This functionality is a powerful generalization of the inner-product FE schemes and allows many practical use cases.

Two quadratic schemes have been implemented in GoFE and CiFEr. The first one, named SGP, is the implementation of the paper [13], the second one is the implementation of [8]. A major downside of the former scheme is that it does not allow encryption using a public key, meaning that for the encryption a private secret key is needed. The latter one [8] allows public key encryption. Interestingly, it is based on the PK FHIPE, a partially function hiding inner-product FE scheme benchmarked in Section 2.3. In what follows, we compare the performance of both schemes.

2.4.1 Benchmarking based on parameter ℓ

In Tables 19 and 20 we compare the performance of both schemes with fixed $b = 1000$ and increasing ℓ. The times needed to perform the *Setup* of the schemes are not presented in the tables since it takes less than a millisecond independently of the parameters ℓ and b.

2.4.2 Benchmarking based on parameter b

In Table 21, 22 we compare the performance of both schemes with fixed $\ell = 10$ and increasing bound b. All results are obtained as averages of repeatedly running the algorithms on random inputs.

2.4.3 Interpretation

Quadratic FE schemes allow the evaluation of much more complicated functions on encrypted data than inner-product ones. For this reason, it is expected to execute slower, but still fast enough for many

ℓ	QUAD. from [8]	SGP from [13]
1	8.547	0.006
2	11.061	0.014
5	25.669	0.037
10	52.215	0.215
20	110.902	0.139
50	288.893	0.235

(a) Key generation

ℓ	QUAD. from [8]	SGP from [13]
1	7.622	1.232
2	8.84	1.442
5	18.289	1.381
10	34.109	2.175
20	65.199	2.136
50	161.16	2.062

(b) FE key derivation

Table 19: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length ℓ

ℓ	QUAD. from [8]	SGP from [13]
1	9.039	1.826
2	17.704	3.585
5	63.162	9.033
10	196.496	16.799
20	677.315	34.239
50	3820.934	83.368

(a) Encryption

ℓ	QUAD. from [8]	SGP from [13]
1	54.476	41.311
2	86.147	85.268
5	254.163	265.622
10	470.814	636.11
20	906.298	1977.007
50	1777.779	10695.916

(b) Decryption

Table 20: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length ℓ

(real-life) use cases as it was demonstrated in D6.4 by evaluating a 2-layer neural network. Comparing the two schemes, the SGP scheme performs faster in most of its algorithms than the quadratic (QUAD) scheme from [8]. This is the price to pay for the schem [8] to allow public key encryption. However, the difference is not big. Since many practical scenarios need public key encryption, the QUAD scheme performs comparably well. Note that the main bottleneck of the quadratic schemes is the calculation of the discrete logarithm in the decryption which can be difficult since a quadratic function can return a much greater output than what its inputs were.

2.5 Attribute-Based Encryption Scheme

In this section, we compare the performance of the two implemented ABE schemes: a CP-ABE scheme FAME [4] and a KP-ABE scheme GPSW [10]. ABE schemes allow encryption of a message together with a decryption policy determining which attributes are needed for a client to be able to decrypt. Such policies can be expressed with boolean expressions or equivalently Monotone Span Program (MSP) structures. The performance of most of the operations depends on the number of attributes used in the scheme specifying the decryption policy. We measure the computational times based on the parameter a counting the number of used attributes.

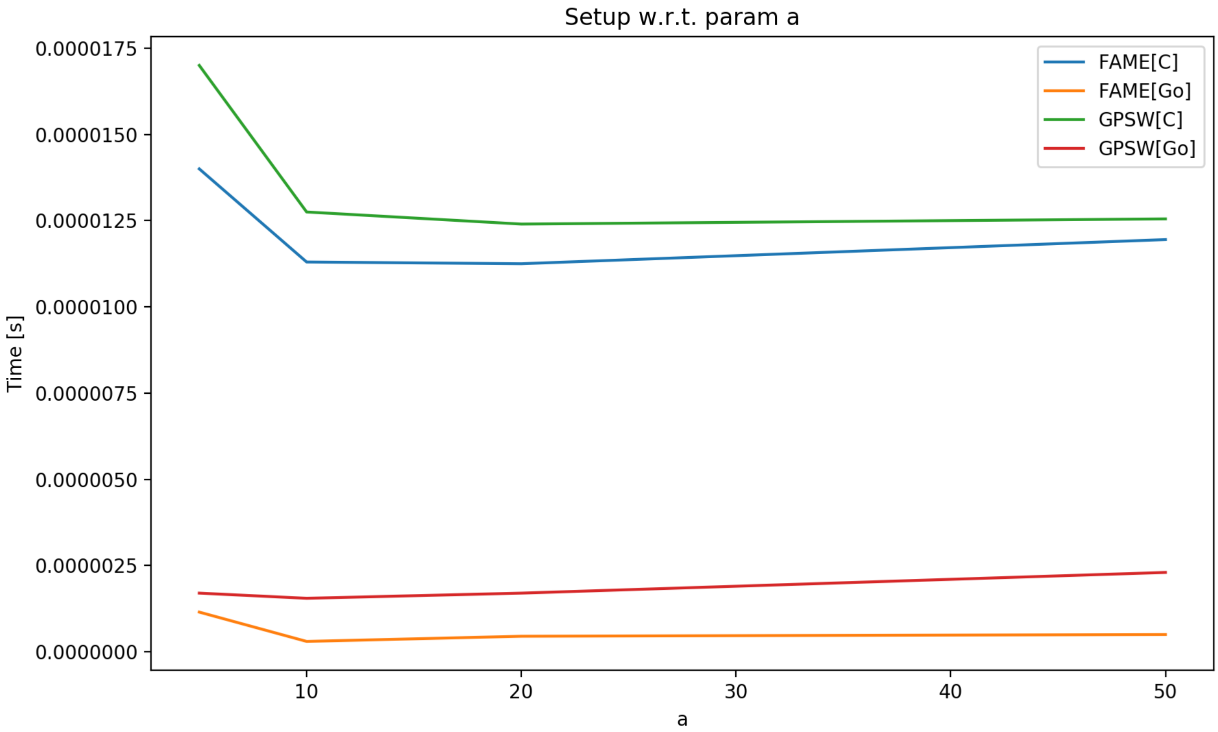

Figure 9: Performance of the ABE setup (in seconds) w.r.t. parameter a

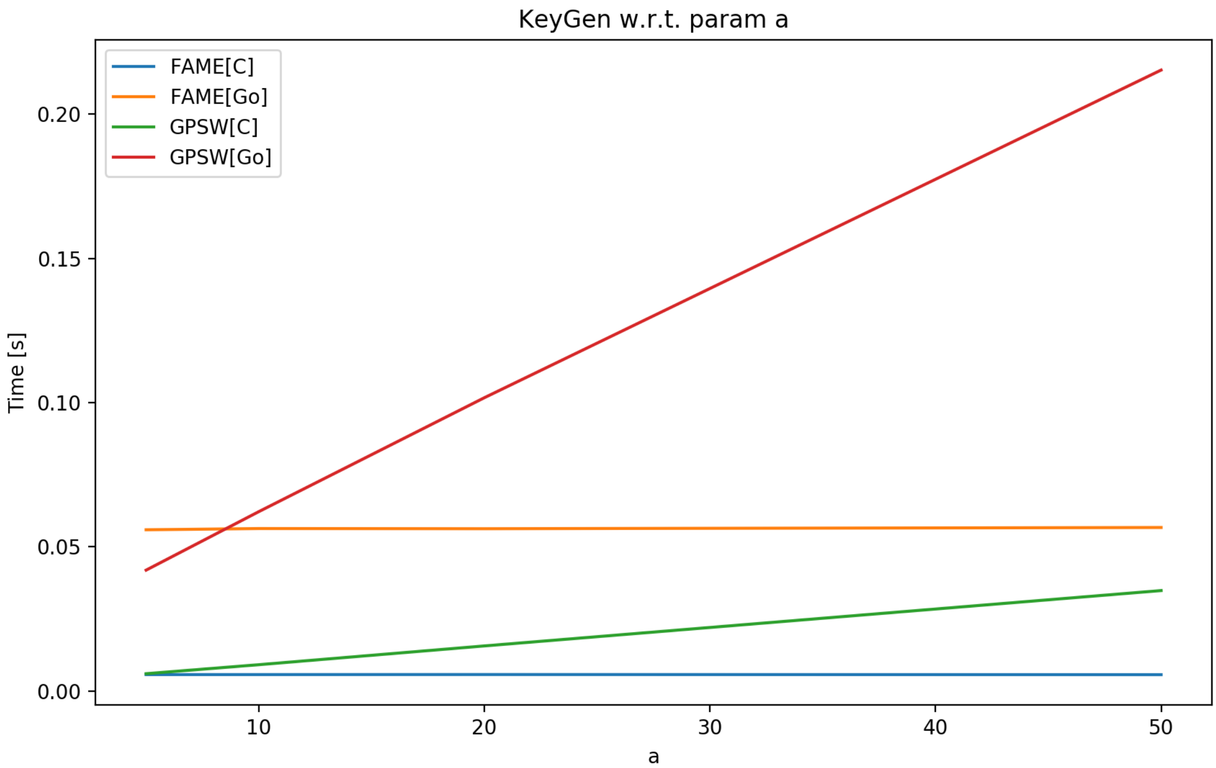

Figure 10: Performance of the ABE key generation (in seconds) w.r.t. parameter a

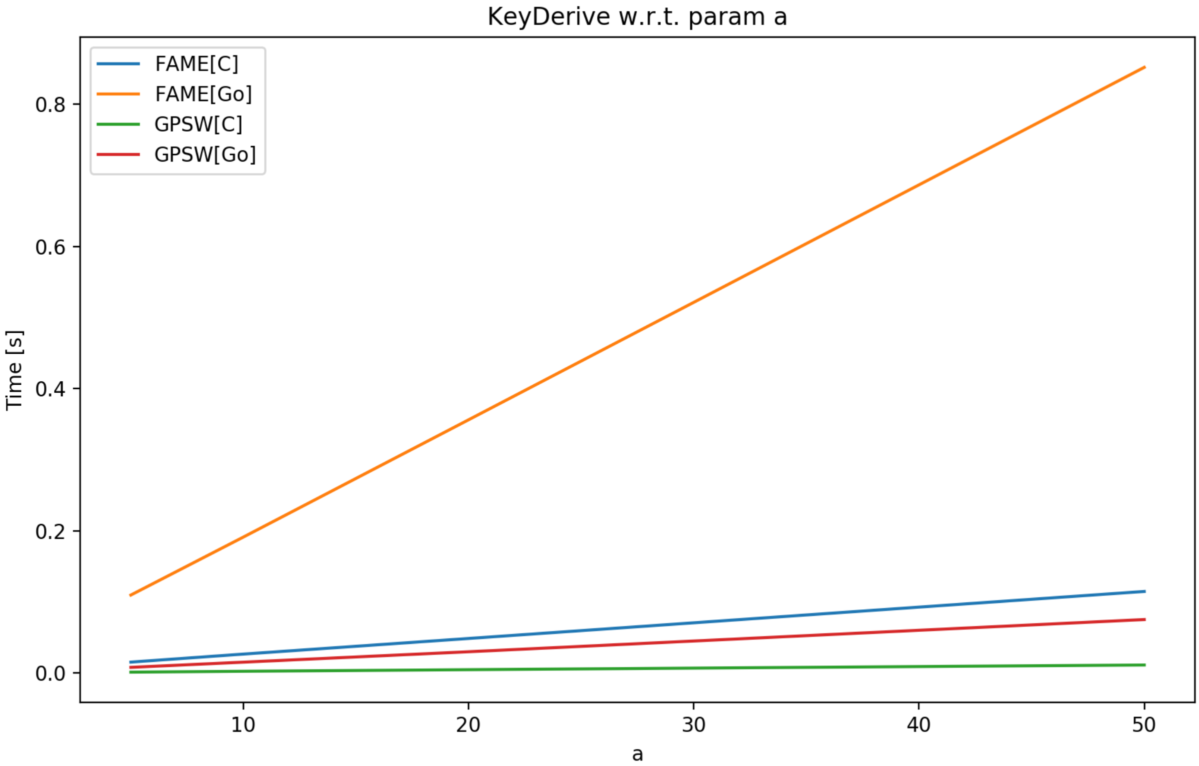

Figure 11: Performance of ABE key derivation (in seconds) w.r.t. parameter *a*

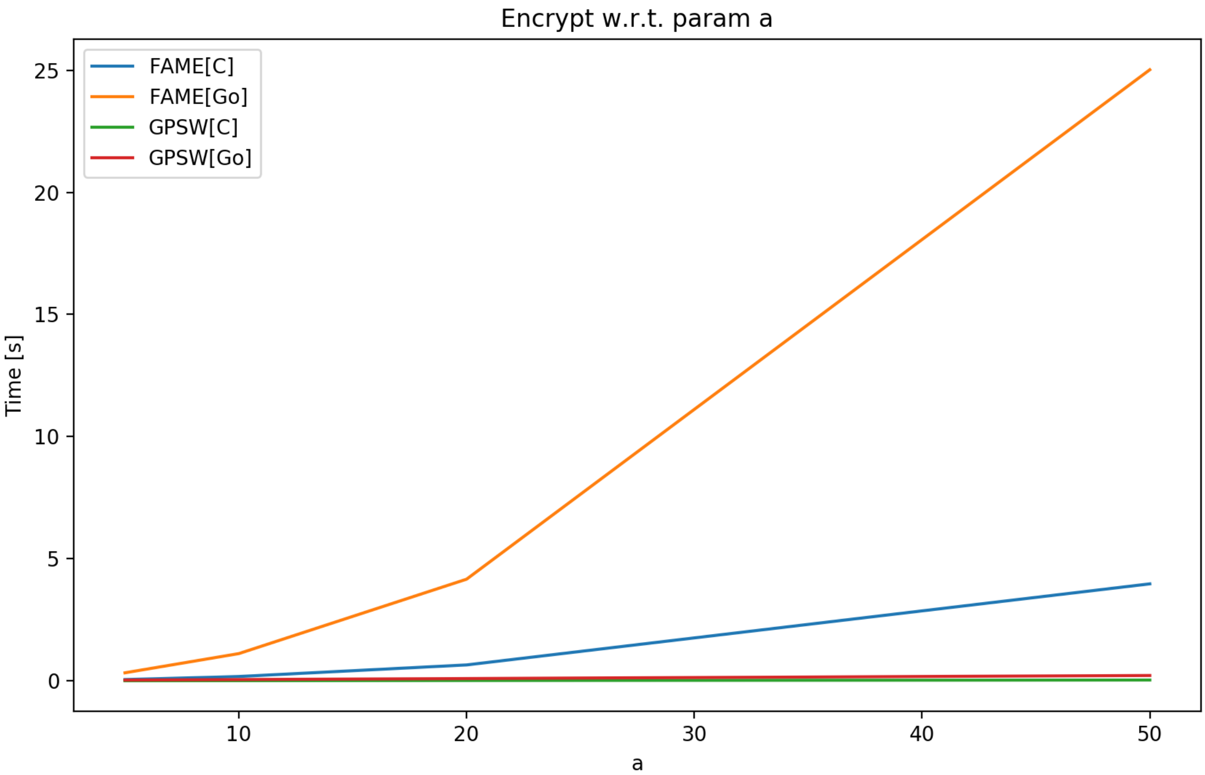

Figure 12: Performance of ABE encrypt operation (in seconds) w.r.t. parameter *a*

b	QUAD. from [8]	SGP from [13]
10	72.714	0.048
20	51.033	0.046
50	51.412	0.046
100	52.215	0.215
200	51.771	0.046
500	59.572	0.079
1000	105.416	0.239

(a) Key generation

b	QUAD. from [8]	SGP from [13]
10	34.474	1.442
20	34.214	1.265
50	34.233	1.298
100	34.109	2.175
200	34.012	1.45
500	34.017	2.283
1000	35.946	3.979

(b) FE key derivation

Table 21: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. bound b

b	QUAD. from [8]	SGP from [13]
10	194.879	17.776
20	194.257	16.31
50	196.118	16.607
100	196.496	16.799
200	197.734	16.279
500	198.483	20.8
1000	206.814	34.448

(a) Encryption

b	QUAD. from [8]	SGP from [13]
10	138.688	402.566
20	146.672	418.916
50	194.723	461.624
100	470.814	636.11
200	874.104	1015.027
500	3746.347	3429.851
1000	4370.975	4200.048

(b) Decryption

Table 22: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. bound b

a	GPSW[Go]	GPSW[C]	FAME[Go]	FAME[C]
5	< 0.0001	< 0.0001	< 0.0001	< 0.0001
10	< 0.0001	< 0.0001	< 0.0001	< 0.0001
20	< 0.0001	< 0.0001	< 0.0001	< 0.0001
50	< 0.0001	< 0.0001	< 0.0001	< 0.0001
100	< 0.0001	< 0.0001	< 0.0001	< 0.0001
200	< 0.0001	< 0.0001	< 0.0001	< 0.0001

Table 23: Performance of the ABE setup (in seconds) w.r.t. parameter a

a	GPSW[Go]	GPSW[C]	FAME[Go]	FAME[C]
5	0.0419	0.0060	0.0559	0.0057
10	0.0622	0.0091	0.0563	0.0057
20	0.1017	0.0156	0.0562	0.0057
50	0.2153	0.0348	0.0567	0.0056
100	0.4071	0.0662	0.0560	0.0057
200	0.7934	0.1290	0.0557	0.0065

Table 24: Performance of the ABE key generation (in seconds) w.r.t. parameter a

a	GPSW[Go]	GPSW[C]	FAME[Go]	FAME[C]
5	0.0078	0.0011	0.1095	0.0152
10	0.0151	0.0023	0.1911	0.0265
20	0.0298	0.0046	0.3561	0.0485
50	0.0751	0.0111	0.8517	0.1146
100	0.1516	0.0225	1.6870	0.2265
200	0.3222	0.0002	3.4590	0.4498

Table 25: Performance of the ABE key derivation (in seconds) w.r.t. parameter a

a	GPSW[Go]	GPSW[C]	FAME[Go]	FAME[C]
5	0.0302	0.0043	0.3285	0.0516
10	0.0518	0.0076	1.1146	0.1766
20	0.0933	0.0139	4.1610	0.6507
50	0.2185	0.0324	25.0193	3.9700
100	0.4276	0.0631	100.1778	15.6191
200	0.8596	0.1254	413.8411	45.0077

Table 26: Performance of the ABE encryption (in seconds) w.r.t. parameter a

a	GPSW[Go]	GPSW[C]	FAME[Go]	FAME[C]
5	0.0599	0.0068	0.0718	0.0092
10	0.1195	0.0137	0.0721	0.0105
20	0.2404	0.0273	0.0723	0.0130
50	0.5998	0.0687	0.0772	0.0225
100	1.2217	0.1489	0.0916	0.0476
200	2.6465	0.0013	0.1897	0.0084

Table 27: Performance of the ABE decryption (in seconds) w.r.t. parameter a

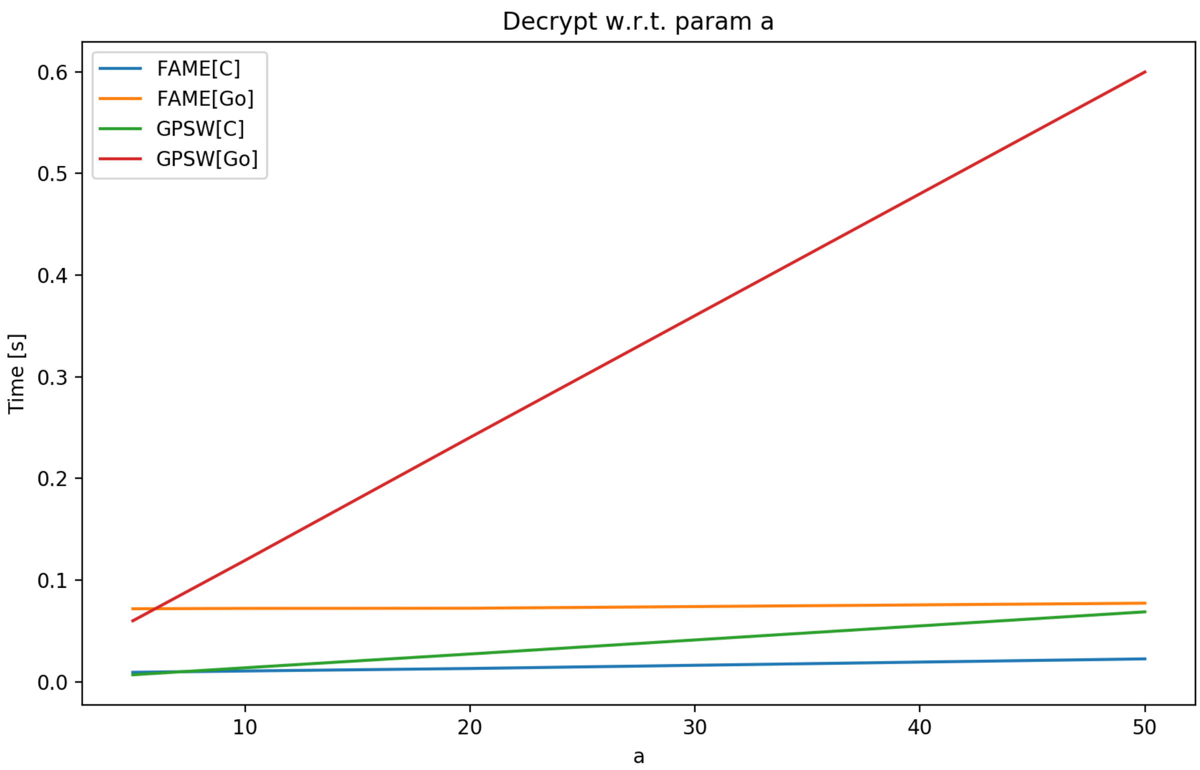

Figure 13: Performance of the ABE decryption (in seconds) w.r.t. parameter a

3 Performance Optimizations

In this section, we summarize the efforts made to improve the performance of the libraries GoFE and CiFEr.

3.1 Precomputation

The majority of the schemes implemented in GoFE and CiFEr depend on the hardness of computing the discrete logarithm in a group (DDH assumption). The group in which this assumption is believed to hold and which was chosen for the implementation is a subgroup of order q in the modular arithmetics group $\mathbb{Z}_p^*$, where $p = 2q + 1$ is a safe prime. One of the main goals of GoFE and CiFEr is to provide FE functionality that is as flexible and versatile as possible. For this reason, the group $\mathbb{Z}_p^*$ in the implementation is not fixed. On the contrary, a user can generate (setup procedure) his own safe prime p that is appropriate for his choice of parameters. A search for a safe prime can be time consuming even if it is done only once at the deployment stage.

Nevertheless, the security of the schemes is not compromised if the same group is used multiple times and certain values are computed in advance, assuming that the values were not chosen specifically to allow backdoors. For this reason, we enabled the initialization of the groups from the precomputed values. This includes the precomputation of the safe prime numbers and generators of the subgroups in advance. This allows the schemes to be used off-the-shelf and set up in less than a millisecond. For users not trusting that the values were precomputed in a random way or wishing to use the scheme with parameters that were not precomputed, the standard setup procedure is still available.

3.2 Elliptic Curve Cryptography

As noted before, the majority of schemes in GoFE and CiFEr are based on the computations in the modular arithmetic subgroup $\mathbb{Z}_p^*$. While some schemes are bound to be used with this group (for example Paillier scheme), others are flexible enough to be implemented with some other group in which DDH assumption is believed to hold.

Groups of elliptic curves are a common replacement for $\mathbb{Z}_p^*$ groups since the numbers that are used are smaller. For this reason, certain operations can be computed faster in elliptic curves groups than in $\mathbb{Z}_p^*$. To improve the performance of the schemes we have implemented the basic adaptively secure DDH inner-product scheme also with operations in the elliptic curve group. The chosen group is a popular P256 group which has an optimized implementation in the standard Golang crypto library. Note that this group should not be confused with pairing groups which are also elliptic curve groups but used for different purposes.

In what follows, we compare the performance of the DDH inner-product scheme implemented with $\mathbb{Z}_p^*$ (2048-bit security and with the precomputed values as described in the previous section) and with the P256 elliptic curve group where various input parameters are used. Recall that the scheme depends on the bound of absolute values of inputs b and the length of input vectors ℓ.

3.2.1 Benchmarking based on parameter ℓ

In Tables 28 and 29, we compare the performance of both schemes with fixed $b = 1000$ and increasing ℓ.

ℓ	DDH in $\mathbb{Z}_p$	DDH in EC
1	11.198	0.19
5	55.395	0.989
10	113.264	2.901
20	231.274	4.103
50	592.737	12.093
100	1152.886	19.285
200	2296.972	40.711

(a) Key generation

ℓ	DDH in $\mathbb{Z}_p$	DDH in EC
1	0.002	0.001
5	0.005	0.003
10	0.007	0.006
20	0.017	0.008
50	0.033	0.02
100	0.068	0.072
200	0.178	0.085

(b) FE key derivation

Table 28: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length ℓ

ℓ	DDH in $\mathbb{Z}_p$	DDH in EC
1	17.08	0.399
5	41.59	1.168
10	73.19	2.013
20	133.242	3.752
50	307.932	9.272
100	593.933	18.611
200	1165.522	35.84

(a) Encryption

ℓ	DDH in $\mathbb{Z}_p$	DDH in EC
1	30.321	47.626
5	44.712	127.668
10	50.333	119.796
20	84.418	239.139
50	80.025	225.917
100	143.987	432.45
200	199.285	425.067

(b) Decryption

Table 29: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. vector length ℓ

3.2.2 Benchmarking based on parameter b

In Tables 30 and 31 we compare the performance of both schemes with fixed $\ell = 10$ and increasing bound b. All results are obtained as averages of repeatedly running the algorithms on random inputs.

3.2.3 Interpretation

Using an EC group excels in key generation procedure and encryption since both procedures include the computation of g^x in a group for a random x, where x is much smaller in the EC group. Other operations are slower which is reflected in decryption procedure taking even more time in EC. Hence, the EC implementation is preferred only in some cases, for example when the inputs bound are small (or the decryption values can be guaranteed small) or if encryption is done on a computationally less powerful device like a cell phone.

b	DDH in $\mathbb{Z}_p$	DDH in EC
10	118.903	3.385
100	113.359	1.84
1000	113.264	2.901
10000	126.262	1.848

(a) Key generation

b	DDH in $\mathbb{Z}_p$	DDH in EC
10	0.007	0.008
100	0.007	0.004
1000	0.007	0.006
10000	0.007	0.004

(b) FE key derivation

Table 30: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. bound b

b	DDH in $\mathbb{Z}_p$	DDH in EC
10	69.992	2.351
100	72.405	1.995
1000	73.19	2.013
10000	69.257	1.986

(a) Encryption

b	DDH in $\mathbb{Z}_p$	DDH in EC
10	13.801	3.17
100	25.036	16.294
1000	50.333	119.796
10000	309.666	1014.068

(b) Decryption

Table 31: Performance of algorithms (in milliseconds) in inner-product schemes w.r.t. bound b

3.3 Gaussian Sampling

Schemes based on LWE and ring-LWE assumptions and Paillier type schemes include sampling values from the so-called discrete Gaussian distribution. While there exist many algorithms and optimized implementations of discrete Gaussian samplers, FE algorithms need to use sampling from Gaussian distribution with relatively big variance. For this reason, we have developed an implementation of a discrete Gaussian sampler built for this task.

We have implemented and integrated into GoFE and CiFEr the new sampler from [14]. It is not limited by sampling only small values and works in constant time to provide side-channel security. We report here on the results of benchmarking it. In Table 32, one can see average times needed to sample a 1000-dimensional vector with the Gaussian sampler where the times depend on the variance of the distribution. Note that the sampler supports sampling with $\sigma = k\sqrt{1/(2\ln(2))}$ for an integer k. One can observe that sampling bigger numbers only slightly worsen the performance. The main reason for this small difference is that the sampler has to deal with greater integers, for example in the case $k = 2^{1024}$ the sampled values have approximately 1024 bits.

$k = \sigma/\sqrt{1/(2\ln(2))}$	Disc. Gauss. sampling
2^1	6.6090
2^2	6.7910
2^4	6.6270
2^8	6.6640
2^{16}	6.6390
2^{32}	6.9110
2^{64}	7.2660
2^{128}	8.0090
2^{256}	9.0260
2^{512}	8.5270
2^{1024}	10.2790

Table 32: Performance of sampling (in milliseconds) a 1000 dimensional vector with discrete Gaussian distribution with various σ

4 Real-world Use Cases Performance

In this section, we briefly present two functional encryption use cases (more detailed presentation can be found in Deliverable D6.3) and discuss their performance. The first one demonstrates the privacy-enhanced health analysis, the second one presents applying neural networks on encrypted images. We compare the performance of the FE approach with the homomorphic encryption (HE) approach. Note that the other two showcases presented in Deliverable D6.3 (privacy-friendly generation of the traffic heatmap and documents access control in hospitals) cannot be implemented by HE.

Performance evaluation of the two scenarios has been discussed to greater detail in a paper accepted at ESORICS [1].

4.1 Privacy-Friendly Prediction of Cardiovascular Diseases

The prediction service is built on the Framingham risk score algorithms. The Framingham heart study 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 Cardiovascular Disease (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. Using FE, the risk score can be computed using only the encrypted values of the input parameters. The user specifies the parameters, these are locally encrypted and sent to the prediction component. The service computes the 30-year risk and returns it to the user.

In [5], a report on the implementation of the 10-year CVD risk score using HE has been done. While this approach has a clear advantage of a prediction service not knowing the risk score, it is also far less efficient than the approach with FE. In a setup that 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 [5] requires more than 30 seconds since computing the Taylor series of 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 the result in a matter of milliseconds.

Furthermore, there is a significant communication overhead in the 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.

The HE approach could be sped up by computing the encryption of only the inner-products (as in the 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 the prediction service knowns the predicted value.

4.2 Neural Networks on Encrypted MNIST Dataset

In the previous section, we described how to implement privacy-friendly predictive services by using efficient FE schemes for inner-products. Using linear functions 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 those 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 in which handwritten digits need to be recognized. A linear classifier on the MNIST dataset is reported to have 92% accuracy, while more complex classifiers achieve over 99% accuracy. GoFE and CiFEr include a scheme [13] for quadratic multivariate polynomials which enables the computation of quadratic polynomials on encrypted vectors. This enables richer machine learning models and even basic versions of neural networks. Using FE, we implemented an accurate neural network classifier for the MNIST dataset. This means that an entity holding a functional 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).

For this use case, the GoFE library and a widely-used machine learning library Tensor-Flow [2] are used. MNIST dataset consists of 60 000 images of handwritten digits. Each image is a 28×28 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 a 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 97% accuracy, a result that is also reported in [13]. The decryption of one image (applying the trained model on the encrypted image) takes under 2 seconds.

Similarly, CryptoNets [9], 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 using the 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 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.

Also, as in the Framingham risk score algorithm, the HE approach is advantageous in the sense that the prediction component does not know the results (only the ciphertext of it). However, when the execution time is prioritized over the accuracy, the FE approach can enable viable machine learning models that can be applied on the encrypted data.

5 Conclusions

In this deliverable, we showed that the performance of the FENTEC libraries is sufficient for real-world use cases. The most time-consuming operations are the ones that are executed only once – at the deployment phase. The operations, like functional key derivation, encryption, and decryption that need to be executed frequently are fast and do not introduce bottlenecks in the deployed systems. Users of the libraries can choose from a variety of functional encryption schemes – ranging from single input inner-product, multi input inner-product and decentralized inner-product schemes, to more complex quadratic and attribute-based encryption schemes. Each of the schemes can be instantiated using different underlying cryptographic primitives that provide the same functionality but excel at different operations. Thus, users can choose the underlying primitives to optimize the performance of the GoFE and CiFEr libraries for their use cases.

References

[1] https://esorics2019.uni.lu/.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[3] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In *IACR International Workshop on Public Key Cryptography*, pages 128–157. Springer, 2019.

[4] Shashank Agrawal and Melissa Chase. FAME: fast attribute-based message encryption. *IACR Cryptology ePrint Archive*, 2017:807, 2017.

[5] Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.

[6] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. *IACR Cryptology ePrint Archive*, 2017:989, 2017.

[7] Pratish Datta, Tatsuaki Okamoto, and Junichi Tomida. Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In *IACR International Workshop on Public Key Cryptography*, pages 245–277. Springer, 2018.

[8] Romain Gay. A new paradigm for public-key functional encryption for degree-2 polynomials. In *IACR International Conference on Public-Key Cryptography*, pages 95–120. Springer, 2020.

[9] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

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

[11] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J Wu. Function-hiding inner product encryption is practical. In *International Conference on Security and Cryptography for Networks*, pages 544–562. Springer, 2018.

[12] Tilen Marc, Miha Stopar, Jan Hartman, Manca Bizjak, and Jolanda Modic. Privacy-enhanced machine learning with functional encryption. In *European Symposium on Research in Computer Security*, pages 3–21. Springer, 2019.

[13] Edouard Dufour Sans, Romain Gay, and David Pointcheval. Reading in the dark: Classifying encrypted digits with functional encryption. *IACR Cryptology ePrint Archive*, 2018:206, 2018.

[14] Raymond K Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 2019.