Towards Efficient Cloud Data Processing: A Comprehensive Guide to CKKS Parameter Selection

Modjtaba Gharibyar[1,*][a], Clemens Krüger[2,*][b] and Dominik Schoop[2][c]

[1]*Research Group for Cryptography and Security, KASTEL Security Research Labs, Germany*

[2]*Department of Computer Science, Esslingen University of Applied Sciences, Germany*

Keywords: Cloud Computing, Security and Privacy, Homomorphic Encryption, CKKS, Parameter Optimization.

Abstract: Cloud computing offers scalability, cost efficiency, and the ability to process large data volumes. However, security and privacy concerns deter many organizations from migrating sensitive data to the cloud. Traditional encryption protects data at rest and in transit but requires decryption for processing, exposing plaintext to cloud providers or attackers. Fully homomorphic encryption (FHE) addresses this issue by enabling computations directly on encrypted data. Among available FHE schemes, CKKS stands out for its relatively good performance but requires careful parameter tuning to balance security, precision, memory use, and runtime efficiency. This paper explores CKKS's practical application by analyzing the impact of parameter configurations on these aspects, demonstrated through prototypical statistical computations. It also provides key criteria for selecting and optimizing parameters to meet desired security and performance levels. The findings simplify CKKS parameter management for non-experts, offering practical guidance for user-friendly implementation.

1 INTRODUCTION

Cloud-based Software as a Service (SaaS) is a cornerstone of modern technology, valued for its scalability, cost efficiency, and ability to process large data volumes. SaaS allows Data Owners to outsource computations to cloud providers, leveraging their substantial processing power.

Despite these advantages, SaaS raises significant privacy concerns. Although secure transmission protocols like TLS protect data in transit, plaintext processing at the cloud backend leaves sensitive information vulnerable to attackers.

Fully homomorphic encryption (FHE) offers a solution by enabling computations directly on encrypted data, maintaining confidentiality throughout the process. Among FHE schemes, CKKS (Cheon et al., 2017) is particularly suited for approximate computations, balancing performance with a degree of inaccuracy. We chose CKKS for this paper, because it is among the most promising schemes for privacy-preserving data analysis and machine learning, due to its comparatively high performance and ability to work on fixed-point numbers. However, its practical use demands careful parametrization to manage trade-offs between security, precision, memory use, and runtime, which can be hard to do properly, especially for beginners.

This study evaluates the critical parameters of CKKS using the OpenFHE library (Badawi et al., 2022). Exemplary statistical calculations, i. e. standard deviation and multi-stage multiplications, were performed to demonstrate the impact of parameter choices on performance and precision.

By addressing these complexities, we aim to provide user-friendly guidelines to simplify CKKS parameter configuration, making FHE accessible to non-experts while ensuring optimal performance and security.

It should be noted that there has been some effort in creating compilers which abstract the FHE and automatically selects parameters (Chowdhary et al., 2021). Nevertheless, we believe it is important for FHE developers to have an understanding of how the parameters interact with each other.

The paper is organized as follows. In Section 2.1, we provide some brief mathematical foundations underlying the CKKS scheme. Section 2.2 details the parameters available in the library OpenFHE. Section 3 outlines the experimental setup and quantitatively defines the objectives. In Section 4, we interpret the results regarding runtime, memory usage, preci-

[a] https://orcid.org/0009-0006-8340-6031
[b] https://orcid.org/0009-0006-2588-0069
[c] https://orcid.org/0009-0006-9971-3677
[*] The authors Gharibyar and Krüger contributed equally to this paper.

Gharibyar, M., Krüger, C. and Schoop, D.
Towards Efficient Cloud Data Processing: A Comprehensive Guide to CKKS Parameter Selection.
DOI: 10.5220/0013144800003899
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 11th International Conference on Information Systems Security and Privacy (ICISSP 2025) - Volume 2*, pages 502-509
ISBN: 978-989-758-735-1; ISSN: 2184-4356
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

sion, and security level, and recommend the optimal parametrization for our functions. Finally, Section 5 summarizes our research results.

2 PRELIMINARIES

In this section, we introduce the fundamental concepts and structures essential for understanding the subsequent analysis and implementation of CKKS. We provide an overview of the polynomial ring structures, key generation processes, and homomorphic operations that form the core of the CKKS scheme.

2.1 CKKS

Let N be a power of two. The ring $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ consists of polynomials with integer coefficients and represents the ring of integers of the $2N$-th cyclotomic field. N is referred to as the ring dimension. The ring $\mathcal{R}_Q := \mathcal{R}/Q\mathcal{R}$ has coefficients reduced modulo Q.

We use a modulus that supports the number theoretic transform (NTT), where $\mathcal{R}_{Q_i} := \mathcal{R}/Q_i\mathcal{R}$ represents the ring with coefficients reduced modulo $Q_i = q_0 q_1 \cdots q_i$, with $i \in \mathbb{N}_0$. The largest modulus, $Q_L = q_0 q_1 \cdots q_L$, corresponds to L multiplicative levels (the number of consecutive multiplications that can be performed; see Section 2.2.3 for more details).

Secret keys s are elements of $\mathcal{R}$ with balanced, ternary coefficients, where $s \in \{0, \pm 1\}^N \subset \mathcal{R}$. The uniform distribution over ternary polynomials is denoted by χ_s. Further, χ_{DG} denotes the discrete Gaussian distribution over $\mathcal{R}$, with coefficients sampled independently with a standard deviation of 3.19, which is the standard choice according to the FHE security standards (Halevi and Shoup, 2020; Albrecht et al., 2019). The CKKS scheme encodes messages using a scaling factor and a packing method over the real numbers: $\mathsf{m'} = \mathsf{Encode}(\mathsf{m})$, where Encode first scales up the message m by a scaling factor Δ, computes the inverse discrete Fourier transform, and then rounds to integers modulo Q_L. This packing method allows homomorphic SIMD (Single Instruction Multiple Data) operations, including addition, multiplication with rescaling (similar to floating-point multiplication), and rotation of the vector of scalars packed in the plaintexts/ciphertexts.

The OpenFHE library implements an optimized variant of the Residue Number System (RNS), also known as the double Chinese Remainder Theorem (CRT). Additionally, the authors slightly changed their implementation compared to the original CKKS, to mitigate key recovery attacks (OpenFHE, 2024). The key operations that can be performed on ciphertexts in the CKKS scheme include:

- Ciphertext-constant operations (CAdd, CSub, CMult): Perform addition, subtraction, and multiplication with constants.
- Ciphertext-ciphertext operations ($\mathsf{Add}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$, $\mathsf{Sub}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$, $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$): Perform addition, subtraction and multiplication with relinearization using the evaluation key evk.
- $\mathsf{Sum}_{\mathsf{rk}}(\mathsf{ct})$: Sum all slots in ct iteratively via rotations using the rotating key $\mathsf{rk}^{(\sigma)}$.
- $\mathsf{Rotate}_{\mathsf{rk}(\sigma)}$: Rotate ct by σ, incorporating relinearization.

2.2 Cryptographic Parameters

In Section 1, we highlighted the inherent trade-off in CKKS between security level, precision, memory use, and runtime. Specifically, the choice of security and precision parameters directly influences runtime and memory use because these parameters determine the dimension $\mathcal{R}$, which reflects the number of polynomial coefficients that must be processed and stored, thereby affecting runtime, ciphertext size, and key size.

In this section, we identify the specific parameters offered by the library OpenFHE that are relevant to CKKS. We categorize these parameters based on the previously discussed factors—security, precision, memory use, and runtime—that contribute to the trade-off in the CKKS scheme.

2.2.1 Cryptographic Parameters for Security

Security Level. In OpenFHE, the security level is determined by the parameter `SecurityLevel`, with options for 128, 192, and 256 bits, applicable to both `Quantum` (resistant to quantum attacks) and `Classic` (resistant to classical attacks) scenarios. These values correspond to the computational resources required to solve the LWE problem at the level of a block cipher. To ensure these security levels, OpenFHE adheres to the "Security Guidelines for Implementing Homomorphic Encryption" (Bossuat et al., 2024).

The guidelines specify explicit values for $\log_2(q)$, where $q = PQ$ represents the LWE modulus in the CKKS scheme. This value defines the maximum bit length required to achieve a specific ring dimension N at a given security level.

It should be noted that the $\log_2(q)$ values are estimates based on the Lattice Estimator (Albrecht et al., 2015), which estimates the costs of various attack methods for a given parameter set.

2.2.2 Cryptographic Parameters for Precision

The `ScalingTechnique` parameter is crucial for precision in the CKKS scheme. In OpenFHE, this parameter is implemented in two RNS variants: static and dynamic. These variants control the rescaling process and directly affect the precision of computations.

Scaling Method. There are static and dynamic scaling methods. The static RNS variant uses the prime power 2^p as the scaling factor across all levels L, where each RNS modulus $q_i \approx 2^p$ is associated with a specific multiplicative level (Ahmad Al Badawi et al., 2022; Blatt et al., 2020; Cheon et al., 2019). In OpenFHE, the static method is selected using the `FIXEDAUTO` scaling technique, where rescaling is performed automatically before each homomorphic multiplication, except for the first one. Alternatively, rescaling can be performed manually by the user with `FIXEDMANUAL` (Ahmad Al Badawi et al., 2022). The dynamic method uses an adaptive scaling factor Δ for each level, which varies per level (Kim et al., 2022). In OpenFHE, this method is implemented using the `FLEXIBLEAUTO` and `FLEXIBLEAUTOEXT` techniques. `FLEXIBLEAUTOEXT` provides higher precision than `FLEXIBLEAUTO`, but it is slower (Ahmad Al Badawi et al., 2022).

FirstModSize and ScaleModSize. In the cryptographic context of CKKS, the parameters `FirstModSize` and `ScaleModSize` are crucial for precision and must be configured accordingly. `FirstModSize` defines the bit length of the initial modulus q_0, while `ScaleModSize` determines the bit length of the scaling factor Δ. These parameters are always configured in pairs, with `FirstModSize` always being larger than `ScaleModSize`.

To understand the relationship between `FirstModSize` and the resulting precision, let us consider the ciphertext modulus chain as described in Section 2:

$$Q_L = q_0 \cdot q_1 \cdot \ldots \cdot q_L$$

The initial modulus q_0 is the largest element in this chain and defines the numerical range in which encrypted data is represented and processed. Therefore, the larger the bit length of q_0, the larger the numbers that can be accurately represented, which implies higher precision.

The scaling factor Δ is used to convert real numbers into an integer form. The scaling factor Δ is defined as $\Delta = 2^{\texttt{ScaleModSize}}$.

2.2.3 Cryptographic Parameters for Runtime and Memory Use

Ring Dimension. The `RingDimension` is crucial for both runtime and memory use. It is typically expressed as 2^n with $n \in \mathbb{N}$ and $10 \leq n \leq 17$. The choice of ring dimension is primarily determined by the multiplicative depth and `ScaleModSize`, along with the desired `SecurityLevel`.

Batch Size. The batch size refers to the number of data values that can be processed simultaneously within a single ciphertext. Batching leverages the property that polynomials in the ring $\mathcal{R}$ can be divided into multiple slots, with each slot being manipulated independently.

MultiplicativeDepth. The parameter `MultiplicativeDepth` determines the maximal number of sequential multiplications that can be carried out before decryption will fail due to the accumulated error. The user can structure the necessary computation in a binary tree of minimal height to optimize this.

3 EXPERIMENTAL SETUP

We carry out an empirical study of the influence of the previously mentioned parameters of the library OpenFHE on run-time, precision and memory use. Therefore, prototype implementations of a number of formulas are executed on a system with 2 Intel Xeon Gold 5315Y (3.2GHz) and 512GB RAM running Ubuntu 22.04.4. We did not utilize any multi threading other than what the OpenFHE library performs out of the box. A synthetically generated dataset, comprising three sets of 65536 (2^{16}) encrypted entries was generated randomly. Each value is within the range $[0,1]$ with a precision of 12 decimal places. All calculations were performed 10 times, and the averages were taken.

Multiplicative Function. The function calculates the product of the slots of the vectors x, y, and z, and sums them over all n elements:

$$f(x,y,z) = \sum_{i=1}^{n} x_i \cdot y_i \cdot z_i \tag{1}$$

We utilize a multiplicative depth of 4 for this function. Note that we require one level to accommodate for the `FLEXIBLEAUTOEXT` scaling technique. We chose this function as a baseline with a relatively low multiplicative depth, which utilizes only multiplications and summation.

Standard Deviation. The standard deviation σ_x of a vector x measures how much the values of x deviate from their mean $\bar{x}$:

$$\sigma_x = \sqrt{n^{-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2}$$

We therefore need to use an approximation function. The OpenFHE library supports the homomorphic evaluation of Chebyshev polynomials for this purpose, which approximates a function to a certain polynomial degree within the given bounds. To accommodate for the approximation, we utilize a multiplicative depth of 12. This formula was chosen in contrast to the previously introduced Multiplicative Function, due to its much higher complexity. Firstly, it utilizes a wider range of operations, including multiplication, summation, subtraction, squaring and the square root. Secondly, the square root specifically introduces the commonly used approximation, which drastically increases the depth as well as the complexity of the formula.

3.1 Evaluation Framework for the Calculations

In this section, we aim to define and quantify the goals for these parameters before moving on to the specific choices made for our implementation. We will let the OpenFHE library automatically determine the `RingDimension` based on the configured parameters.

Security. For our performance experiment, we vary the `SecurityLevel` parameter by choosing $\kappa = 128$, $\kappa = 192$ and $\kappa = 256$ within the `Classic` setting.

Precision. For our study, we are measuring not only the runtime of different parameter sets, but also the precision of the results. Precision should be assessed by comparing the decrypted result of a ciphertext with the result of an unencrypted reference calculation. We first calculate the relative error between the unencrypted (m) and the homomorphically calculated (m') results for each slot i using $RE = abs(\mathsf{m}^{(i)} - \mathsf{m'}^{(i)})/abs(\mathsf{m}^{(i)})$. We then convert this error into bits by calculating $-\log_2(\mathrm{RE})$. We then take the average of these results.

Runtime. We will measure relative runtime by recording the total execution time in seconds for each computation run. This will encompass the time required for encryption, homomorphic computation, and decryption. We do not measure the time for key generation and setting the crypto context.

Memory Use. Memory use is evaluated by analyzing the size of both the ciphertext and the cryptographic keys, measured in kilobytes. This assessment will consider the storage requirements for encrypted data and the associated evaluation keys.

Parameterization. We test all `ScaleModSizes` between 20 and 59 (greatest possible value for OpenFHE in 64-bit mode). Within this range, additional parameters were selectively adjusted to assess the corresponding `RingDimension` chosen for the given Security Level, as well as the respective measurement results. Our measurements consist of several dimensions of parameters, which are used for all tested algorithms. First of all, we test the three Classic Security Levels 128 bits, 192 bits and 256 bits (represented as 1-3). Secondly, we test three `ScalingTechniques`, namely `FIXEDAUTO`, `FLEXIBLEAUTO`, and `FLEXIBLEAUTOEXT` (represented as 1-3). For the scaling parameters, we systematically test all values for `ScaleModSize` in the range $[20,\ldots,59]$, while leaving the `FirstModSize` as the maximum value of 60. Finally, to determine how the chosen `BatchSize` influences the runtime of the calculation, we perform each measurement in two variants: First, we test full packing by setting the maximum possible `BatchSize`, i. e. half the ring dimension. Secondly, we perform the same measurements with a `BatchSize` of 32, representing sparse packing.

3.2 Polynomial Degrees for Chebyshev Approximation

As previously discussed, the standard deviation requires a square root operation which needs to be approximated. The degree of the chosen Chebyshev polynomials massively influences the results. The higher the chosen degree, the more accurate the results will be, however the number of consumed multiplicative levels as well as the runtime will also increase.

We therefore conducted measurements with several degrees to determine the best trade-off for this study. We tested every polynomial degree in the range [10, 20, 30, ..., 2020]. Each parameter set was tested 5 times and the average computation time and precision was taken. We set the `SecurityLevel` to 1, the `ScalingTechnique` to 2, the `FirstModSize` to 40, the `ScaleModSize` to 39 and the `BatchSize` to 32.

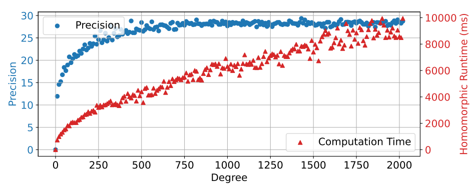

Figure 1: Chebyshev approximation results.

🟥 Ring dimension: 2^{14}
🟩 Ring dimension: 2^{15}
🟦 Ring dimension: 2^{16}
🟪 Ring dimension: 2^{17}

● FIXEDAUTO
× FLEXIBLEAUTO
+ FLEXIBLEAUTOEXT

Figure 2: Legend of result graphs.

We performed several measurements for the approximation of the square root across one ciphertext with varying polynomial degrees and then calculated the average of each of the results. This average can now be compared to the average result of the plaintext reference implementation to get an understanding of the precision of the approximation.

Figure 1 shows the results of these measurements. On the left, the graph plots the precision of the results in bits. On the right, the computation time is shown. It is clearly visible that the precision increases logarithmically until it reaches its limit of about 28 bits at a polynomial degree of around 750. The computation time appears to grow logarithmically as well, although there is no clear limit in the tested range. For our measurements, we chose the polynomial degree of 247 for all further approximations, as it provides a decent level of precision at around 25 bits, which is only about 3 bits below the upper limit, while still having comparatively low computation time.

4 RESULTS

After evaluating the calculations as described in Section 3, we present the results, including the average computation times, the precision of the results, as well as the sizes of the keys and ciphertexts. In the following sections, we will show and discuss the results for the two exemplary functions in detail. .

Unless otherwise specified, the following graphs adhere to the legend shown in Figure 2. The color of each point defines the ring dimension of the measurement, and the marker style defines the utilized scaling technique.

4.1 Multiplicative Function

We discuss the computation time, precision and memory usage of each parameter set for the Multiplicative

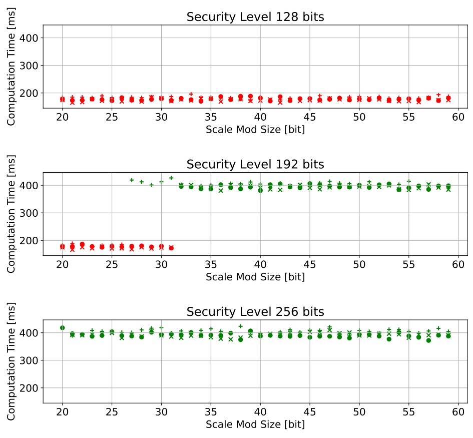

Figure 3: Computation time for the Multiplicative Function.

Function (Equation 1).

Computation Time. Figure 3 shows the computation time for each value of `ScaleModSize`. The graph shows the measurements with full packing only. The differences between full and sparse packing will be discussed later. For a security level of 128 bits, every measurement results in a ring dimension of 2^{14}. For 192 bits, all measurements with `ScaleModSizes` higher than 31 using the scaling techniques `FIXEDAUTO` and `FLEXIBLEAUTO` lead to a higher ring dimension of 2^{15}. Incidentally, these measurements also roughly double the computation time compared to the rest of the results. The scaling technique `FLEXIBLEAUTOEXT` already results in a higher ring dimension starting with `ScaleModSize` 27, due to the fact that it consumes an additional level. For 256 bits, all measurements result in a ring dimension of 2^{15}.

The graph shows that the computation time mainly scales with the chosen ring dimension, whereas the `ScaleModSize` has no noticeable direct influence. However, as a higher `ScaleModSize` eventually increases the ring dimension, there is of course a transitive influence. With a ring dimension of 2^{15} the computation time is roughly double compared to ring dimension 2^{14}.

Comparing the three security levels one can see that 128 bits can be achieved with a ring dimension of 2^{14} throughout all `ScaleModSizes`. With 192 bits, after a certain `ScaleModSize` the ring dimension increases to 2^{15}. At 256 bits all measurements resulted in the ring dimension 2^{15}. Therefore, the security level of 192 bits shares its results with parts of the other security levels. This indicates that the chosen parameters for 128 bits for the measurements until `ScaleModSize` 31 happen to be high enough for both 128 bits as well as 192 bits. The same goes for the

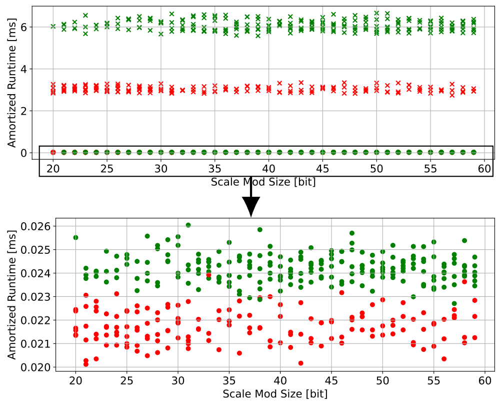

Figure 4: Full vs. sparse packing for the Multiplicative Function.

● Full packing
× BatchSize 32

Figure 5: Legend of packing graphs.

higher `ScaleModSizes` which result in parameters high enough for both 192 bits and 256 bits. Due to the much higher computation times in the respective higher ring dimensions, it is recommended to choose a `ScaleModSize` within the tight range where the precision is optimal but ring dimension stays low.

Figure 4 shows the amortized computation time per slot in the ciphertexts across all measurements for the Multiplicative Function. The different `RingDimensions` are color-coded the same as the previous graphs, whereas the marker style shows the packing type as described in Figure 5.

The first graph shows both packing types in one, whereas the second graph zooms into the results of the full packing only. It is clearly visible that the batch size massively influences the computation time. Generally speaking, higher `BatchSizes` result in higher computation times. However, since the throughput is also increased due to the fact that more numbers are calculated per batch, the amortized computation time is much lower with full packing. Additionally, the `RingDimension` plays a noticeable role here as well, however it is much smaller compared to the packing type. These observations make it clear that if the implemented application is able to utilize higher `BatchSizes`, full packing should always be used. If the application does not get any advantages from high `BatchSizes`, then a more appropriate smaller `BatchSizes` should be chosen to minimize the overall computation time.

Precision. Figure 6 shows the precision in bits (as defined in Section 3) of each computation result across all tested `ScaleModSizes`, specifically those

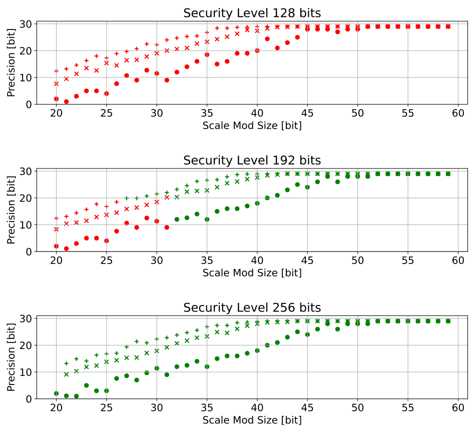

Figure 6: Precision for Multiplicative Function.

with full packing. It is evident that the `ScaleModSize` has a big impact on the precision of the end result. The same goes for the Scaling Technique, where `FLEXIBLEAUTOEXT` leads to the highest precision and `FIXEDAUTO` to the lowest. Neither the security level nor the chosen ring dimension have a noticeable effect on the precision. It can also be seen that there is a clear upper limit to the precision of this function, which is at around 30 bits. This upper limit is reached with a `ScaleModSize` of about 50 in the worst case, for some `Scaling Techniques` even earlier. Therefore there is no reason to choose higher values. For the security level of 192 bits, the ring dimension increases with `ScaleModSizes` larger than 41 for the first two scaling techniques. To get a good trade-off between runtime and precision, it would be best to choose 41, as the precision is already relatively high, whereas the runtime is comparably low due to the lower ring dimension. Since the ring dimension stays the same within the measurements for both 128 and 256 bits and the previous section showed that `ScaleModSize` does not influence performance meaningfully, there is no such trade-off to be made here. For these security levels, the precision can be freely chosen according to the application requirements.

Table 1: Average memory usage for different ring dimensions of Multiplicative Function.

Parameter	Ring Dimensions	
	2^{14}	2^{15}
`CiphertextSize [MB]`	1.002	2.002
`PublicKeySize [MB]`	1.830	3.676
`EvalMultKeySize [MB]`	5.490	11.038
`SecretKeySize [MB]`	0.666	1.338

Memory Usage. Table 1 shows the detailed memory usage for the ciphertext, public key, multiplication key and private key across all measurements for this

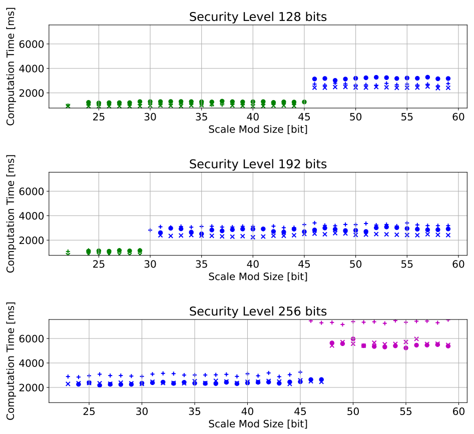

Figure 7: Computation time for the Standard Deviation.

function, separated by ring dimension. The ring dimension is the main contributor to memory size. As can be seen in the table, the memory usage roughly doubles from the lower to the higher ring dimension.

4.2 Standard Deviation

In this section we present the measurement results for the Standard Deviation. We will discuss the computation time, precision and memory usage of each parameter set.

Computation Time. Here we can see considerable differences compared to the previous function. Figure 7 shows the computation times of the Standard Deviation, where the effects of the square root approximation are apparent, as the computation takes much longer. There are two factors here which drive this increase in runtime. First of all, the approximation function itself takes a noticeable amount of time. Secondly, the approximation requires a much higher multiplicative depth for the calculation, which increases the overall runtime of all calculations. To get a high `ScaleModSize` above 47 bits combined with 256 bits of security, the ring dimension jumps to 2^{17}. In general, the computation times are about 10 times higher than those of the Multiplicative Function with comparable ring dimensions. With the highest ring dimension, the computation time goes up to about 7 seconds, compared to a maximum of about 0.5 seconds for the other functions.

Figure 8 shows the relationship between full and sparse packing for the Standard Deviation. Again, the difference between the packing strategies is apparent. However, in this graph the change in runtime for the different `RingDimensions` is a lot more noticeable. Here, in contrast to the two functions previously discussed, the difference between `RingDimensions` is

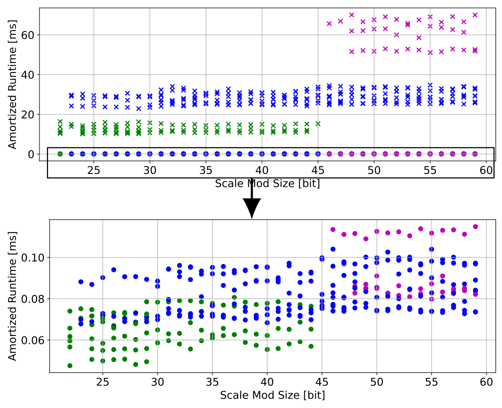

Figure 8: Full vs. sparse packing for the Standard Deviation.

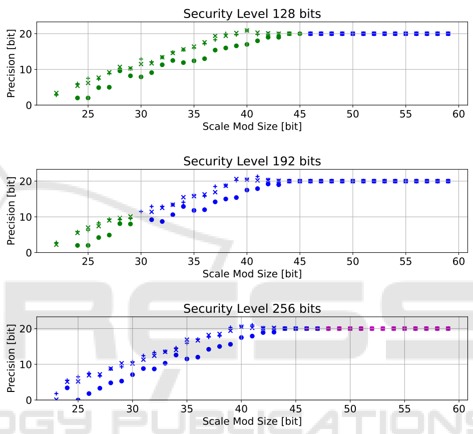

Figure 9: Precision for the Standard Deviation.

even discernible for the full packing. This is due to the fact that the overall runtimes are much higher for the Standard Deviation, which makes these effects stand out more against small measurement noise.

Precision. While the precision curve of the Standard Deviation (Figure 9) follows the same pattern as the previous function, there is a clear difference. In this case, the limit of the precision is at 20 bits, as compared to 30 bits before. This can be attributed to the inclusion of the approximated square root, which reduces the maximum possible precision for this function. This maximum precision is reached at a `ScaleModSize` of 44 throughout all security levels. It can be seen that the highest possible precision can be reached at a security level of 256 bits without having to accept a ring dimension of 2^{17}. The same goes for 128 bits of security, where ring dimension 2^{15} is not needed for optimal results.

Table 2: Average memory usage for different ring dimensions of the Standard Deviation.

	Ring Dimensions		
Parameter [KB]	2^{15}	2^{16}	2^{17}
`CiphertextSize`	5.519	12.663	26.740
`PublicKeySize`	8.224	17.512	36.740
`EvalMultKeySize`	24.980	52.683	110.217
`SecretKeySize`	3.269	6.622	13.371

Memory Usage. The higher multiplicative depth of the Standard Deviation leads to a drastically increased memory usage (Table 2). This can be attributed to the fact that the higher depth also increases the modulus, which in turn affects the memory usage. This shows that while the ring dimension is a major factor when it comes to memory usage, other factors such as the multiplicative depth also influence it.

In the highest ring dimension, which is needed to reach the highest `ScaleModSizes` in the 256 bits security level, the memory usage is more than 10 times higher than the highest ring dimension for the previous function. However, as described above, this ring dimension can be avoided without loss of precision or security.

5 CONCLUSION

In this paper we evaluated the main parameters for configuring the CKKS encryption scheme within the library OpenFHE. The results indicate that both runtime and precision are strongly affected by the computational complexity of the functions being executed.

Our observations provide some guidance on which parameters affect which aspects of the results. In general, the computation time and memory usage are mainly influenced by the ring dimension as well as the multiplicative depth. The precision on the other hand is influenced mostly by the `ScaleModSize`.

The main goal when designing an application using the CKKS scheme is to optimize the parameters. It is crucial to choose parameters which have acceptable precision, but at the same time achieve the lowest possible ring dimension for the chosen Security Level. Especially the measurement results for 192 bits of security across all three functions showed that it is possible to stay in a lower ring dimension while still retaining close to the maximum precision.

REFERENCES

Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca (2022). Openfhe: Open-source fully homomorphic encryption library.

Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., and Vaikuntanathan, V. (2019). Homomorphic encryption standard. Cryptology ePrint Archive, Paper 2019/939.

Albrecht, M. R., Player, R., and Scott, S. (2015). On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203.

Badawi, A. A., Alexandru, A., Bates, J., Bergamaschi, F., Cousins, D. B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Pascoe, C., Polyakov, Y., Quah, I., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., and Zucca, V. (2022). OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915.

Blatt, M., Gusev, A., Polyakov, Y., Rohloff, K., and Vaikuntanathan, V. (2020). Optimized homomorphic encryption solution for secure genome-wide association studies. *BMC Medical Genomics*, 13:1–13.

Bossuat, J.-P., Cammarota, R., Cheon, J. H., Chillotti, I., Curtis, B. R., Dai, W., Gong, H., Hales, E., Kim, D., Kumara, B., Lee, C., Lu, X., Maple, C., Pedrouzo-Ulloa, A., Player, R., Lopez, L. A. R., Song, Y., Yhee, D., and Yildiz, B. (2024). Security guidelines for implementing homomorphic encryption. Cryptology ePrint Archive, Paper 2024/463.

Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. (2019). A full RNS variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada*, pages 347–368. Springer.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China*, pages 409–437. Springer.

Chowdhary, S., Dai, W., Laine, K., and Saarikivi, O. (2021). Eva improved: Compiler and extension library for ckks. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 43–55.

Halevi, S. and Shoup, V. (2020). Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481.

Kim, A., Papadimitriou, A., and Polyakov, Y. (2022). Approximate homomorphic encryption with reduced approximation error. In Galbraith, S. D., editor, *Topics in Cryptology – CT-RSA 2022*, Security and Cryptology, pages 120–144.

OpenFHE (2024). Security notes for homomorphic encryption. https://openfhe-development.readthedocs.io/en/latest/sphinx_rsts/intro/security.html.