Understanding the Error Sensitivity of Privacy-Aware Computing

Matías Mazzanti[1], Esteban Mocskos[1], Augusto Vega[2], Pradip Bose[2]
[1]*University of Buenos Aires (Argentina),* [2]*IBM T. J. Watson Research Center (NY, USA)*

***Abstract*—Homomorphic Encryption (HE) enables secure computation on encrypted data without decryption, allowing a great opportunity for privacy-preserving computation. In particular, domains such as healthcare, finance, and government, where data privacy and security are of utmost importance, can benefit from HE by enabling third-party computation and services on sensitive data. In other words, HE constitutes the "Holy Grail" of cryptography: data remains encrypted *all the time*, being protected while in use.**

HE's security guarantees rely on noise added to data to make relatively simple problems computationally intractable. This error-centric intrinsic HE mechanism generates new challenges related to the fault tolerance and robustness of HE itself: hardware- and software-induced errors during HE operation can easily evade traditional error detection and correction mechanisms, resulting in silent data corruption (SDC).

In this work, we motivate a thorough discussion regarding the sensitivity of HE applications to bit faults and provide a detailed error characterization study of CKKS (Cheon-Kim-Kim-Song). This is one of the most popular HE schemes due to its fixed-point arithmetic support for AI and machine learning applications. We also delve into the impact of the residue number system (RNS) and the number theoretic transform (NTT), two widely adopted HE optimization techniques, on CKKS' error sensitivity. To the best of our knowledge, this is the first work that looks into the robustness and error sensitivity of homomorphic encryption and, as such, it can pave the way for critical future work in this area.

I. Introduction and Background

Cloud computing has profoundly changed the way businesses and individuals use, process, and manage their data [1]. Despite the benefits of this approach, its adoption exacerbates some existing problems and generates new ones related to the security and privacy of user data [2]. In particular, delivering data to a third party to be processed opens up different possibilities of compromising them, both by improper access or by a decision of the provider to give access to the data without authorization. One of the solutions to this problem resides in the use of data encryption schemes, which allow the data to only be interpreted by the holder of the key that allows decryption [3], [4]. However, most of the cryptography schemes do not allow computing directly on the encrypted data; it is necessary to decrypt before processing them. In this way, an external cloud computing provider can access the unencrypted data and, therefore, be able to compromise them. *Homomorphic Encryption* (HE) solves this challenge by allowing **computation on encrypted data** [5]. Although HE schemes have high computational and memory requirements, which have limited so far their widespread adoption, the great interest in developing new schemes and the recent development of aggressive optimizations (at both algorithm and hardware levels) has opened the door to its use in real-world applications. Figure 1 presents an overview of a typical HE setting that involves a client that makes use of third-party cloud services.

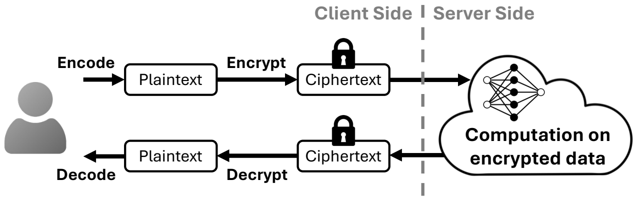


Fig. 1: Homomorphic encryption used in a typical business application: users send their data encrypted to a third-party service provider, where data is processed in its encrypted form.

Multiple HE schemes exist today, with different characteristics and supported capabilities. In this work, we focus on CKKS (Cheon-Kim-Kim-Song) [6], [7], a popular HE scheme for AI and machine learning applications due to its fixed-point arithmetic support. Fundamentally, schemes like CKKS base their hardness on a simple idea: add small errors ("noise") to data to make relatively simple problems computationally intractable, an approach known as *Learning With Errors* (LWE) [8]. In other words, HE operations (like additions and multiplications) take place within a "noisy" domain.

As HE operates with noisy (or "consciously erroneous") data, a consequent question arises: *how can we distinguish between HE's deliberately introduced error and error resulting from faulty hardware or software?* The answer, although not straightforward, motivates us to conduct a thorough study of the sensitivity of CKKS to bit faults ("flips"), resulting in a detailed error characterization study of this scheme.

A. Silent Data Corruption

Errors across CKKS stages (encoding, encryption, decryption, and decoding) can result in two scenarios: HE operation "breaks"[1] and the error is detected, or HE operation does not break and the error propagates across stages and ends up on *silent data corruption* (SDC). The latter is the dangerous case and, as it has been widely studied and reported, hardware- and software-induced SDC happens, even in today's cutting edge

[1] The FHE library used in this work (OpenFHE) detects the data alteration and finishes its execution with an assertion error.

systems and large-scale datacenters [9]. The nasty aspect about hardware and software errors in HE applications emerges from the very same error-centric intrinsic operation of HE schemes. Once in the HE domain, data becomes noisy by construction and, if additional error occurs due to faulty hardware or software, such error camouflages within the HE error and becomes very hard to detect. Figure 2 presents a cartoonish illustration of this idea, where original data (plaintext) is "encrypted" by adding random HE error (noise)[2].

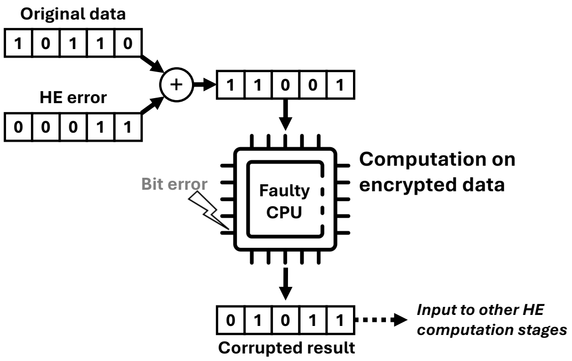


Fig. 2: Illustrative scenario of a SDC case induced by a faulty CPU. The corrupted result incorporates both the HE error and the faulty hardware error.

II. Error Resilience Analysis

This section presents preliminary error sensitivity results for CKKS. As illustrated in Figure 1, the *encoding* stage transforms user's input data into a *plaintext*, a polynomial of degree N that we will call $p(X)$. This plaintext is then encrypted into a *ciphertext*, a pair of polynomials of degree N each, that we will call $c = (c_0(X), c_1(X))$. We adopt a single-bit error model. As such, each bit of every polynomial coefficient (in both the plaintext and the ciphertext) is flipped in sequence for the set of single-bit-flip fault injection experiments. After each bit error injection, we execute the entire HE pipeline, and compare the recovered data after decoding (last stage) against the original data. This methodology is depicted in Figure 3, where two input elements (original message) are encoded into a 4-element polynomial (plaintext) and encrypted into two 4-element polynomials (ciphertext). A coefficient bit is flipped at a time before decryption and decoding. The recovered message is compared against the original one using the L_2 norm. We execute all runs on an Intel i7-11700 CPU with 32 GB RAM and Arch Linux 257.5-1, using the CKKS implementation from the OpenFHE library [10], which includes native RNS and NTT support and 64-bit coefficient representation.

A first observation is that the error behavior is similar when bit errors occur in coefficients of $p(X)$ (the plaintext) and in coefficients of $c_0(X)$, one of the two polynomials in c (the ciphertext). For this reason, and due to space constraints, in this paper we focus our campaign on errors in $c = (c_0(X), c_1(X))$.

[2] In practice, encryption involves additional steps not shown in the figure.

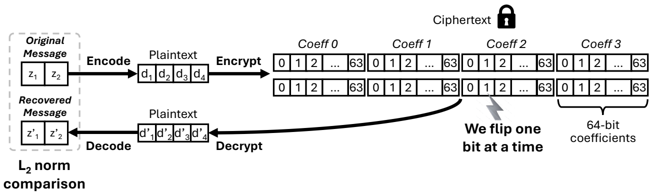


Fig. 3: Error injection methodology.

This initial study emphasizes the occurrence of bit errors specifically during the encoding/encryption (and corresponding decoding/decryption) stages. It does not address errors that may arise during computations performed on the encrypted data, which will be the subject of future research.

A. Error Position Impact

This section examines the impact of a single bit error introduced in the ciphertext $c = (c_0(X), c_1(X))$. We consider a simple scenario where both polynomials $c_0(X)$ and $c_1(X)$ consist of four 64-bit coefficients each. Figure 4 illustrates the L_2 norm error of the output after decoding, in comparison to the original input. A somewhat expected observation is that the magnitude of the error increases with the importance of the altered bit within each coefficient. In particular, modifications to the first 50 bits of each coefficient result in negligible effects on the recovered output. Additionally, we observe that $c_1(X)$ exhibits more pronounced error peaks than $c_0(X)$. This phenomenon can be understood by examining the CKKS decryption process (Equation 1), where polynomial $c_1(X)$ is multiplied by the secret key s (another polynomial), leading to the dispersion of the error across the coefficients.

$$m' = [c_0 + c_1 \times s]_Q \tag{1}$$

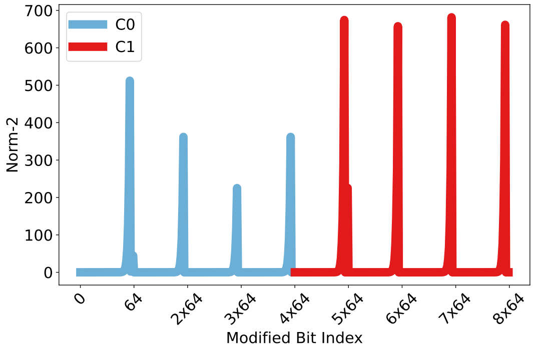


Fig. 4: L_2 norm error of a single-bit flip in the ciphertext. The X-axis shows the position of the modified bit. Polynomial degree $N = 4$.

B. Scale Factor (Δ) Impact

CKKS employs various configuration parameters to guarantee a certain security level (e.g., 128 bits), such as the polynomial degree N, the ciphertext coefficient modulus q, and the scale factor Δ. The scale factor is essential for adjusting the input data to maintain its precision as much as possible throughout the HE stages and, as discussed in this section, it influences error sensitivity. Figure 5 shows the L_2 norm error of the recovered output using different scale factors: 2^{20}, 2^{40}, and 2^{50}. As observed, an increase in Δ leads to enhanced error resilience, characterized by a reduction in errors. The multiplication by the scale factor Δ effectively "shifts right" the coefficient, thereby augmenting the quantity of bits that remain unaffected by errors.

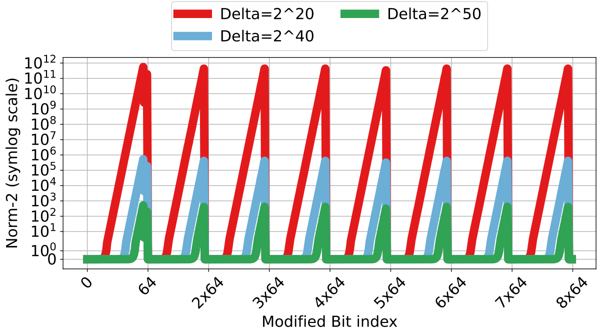


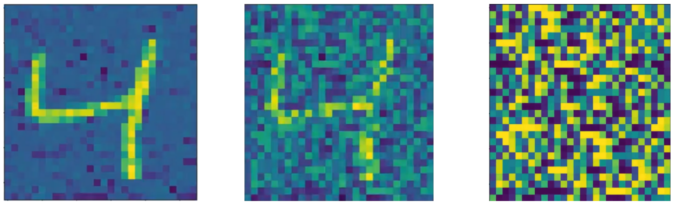
Fig. 5: L_2 norm error of a single-bit flip in the ciphertext using different scale factors Δ. The X-axis shows the position of the modified bit. Polynomial degree $N = 4$.

In practical applications, the scale factor Δ must be carefully adjusted for effective CKKS implementations. The choice of this factor represents a compromise between enhanced computation precision (larger Δ) and reduced computational complexity (smaller Δ). However, when incorporating error resilience into the analysis, such compromise needs to be revisited.

C. RNS and NTT Optimizations Impact

Schemes like CKKS rely on two widely adopted optimizations to make HE problems tractable on available systems: the residue number system (RNS) and the number theoretic transform (NTT). RNS splits the huge polynomial coefficients used by HE schemes into smaller ones that fit common 64-bit processor words; while NTT enables efficient multiplication of high-degree polynomials. It is not the focus of this work to delve into the details of RNS and NTT (additional details can be found in [7]). However, these optimization techniques have different effects on CKKS' error sensitivity. Figure 6 illustrates a scenario in which an image from the MNIST dataset (Figure 6(a)) is processed through the HE pipeline. The effect of a bit error introduced during the encoding phase, without the application of RNS and NTT, is depicted in Figure 6(b), where a fairly accurate representation of the original image is retrieved. Conversely, when a bit error occurs during the utilization of RNS and NTT, the resulting image is entirely distorted, as demonstrated in Figure 6(c).

RNS case: RNS uses Equation 2 to reconstruct a polynomial coefficient from its smaller RNS remainders r_k. A bit error in r_k is amplified when multiplied by the large $\left[\left(\frac{1}{Q_k}\right) \bmod q_k\right] Q_k$ factor, impairing the reconstruction of the original coefficient.



(a) Original (b) No RNS/NTT (c) RNS+NTT

Fig. 6: Impact of a single bit-flip in CKKS encoding.

$$p = \left(\sum_{k=1}^{L} r_k \left[\left(\frac{1}{Q_k} \right) \bmod q_k \right] Q_k \right) \bmod Q \qquad (2)$$

NTT case: The NTT, which is a variant of the Discrete Fourier Transform (DFT), can be implemented using Cooley-Tukey butterfly computations (Figure 7). As a result, a bit error in any of the NTT input elements (a or b in the figure) will inherently spread across *all* the output elements (A and B).

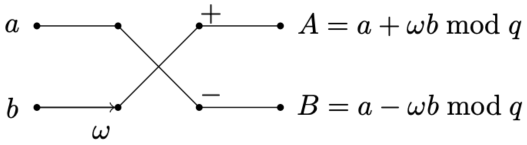


Fig. 7: Cooley-Tukey butterfly.

III. Conclusion

This work provides a first-cut analysis of error characterization in CKKS, a popular homomorphic encryption (HE) scheme due to its support of fixed-point arithmetic in AI and machine learning applications. The study examines the sensitivity of errors based on their occurrence location, the scale factor Δ, and the implementation of residue number system (RNS) and number theoretic transform (NTT) techniques.

The findings reveal trade-offs between robustness, performance, and security that may be overlooked when robustness is not considered. For instance, while smaller scale factors are advantageous for minimizing HE complexity and enhancing the noise budget for operations, larger scale factors can enhance bit error resilience. Additionally, RNS and NTT techniques are acknowledged as significant optimizations in today's CKKS applications; however, their use may lead to severe consequences in the presence of bit errors.

The intrinsic error-centric nature of HE exacerbates this issue, as errors arising from defective hardware or software can easily blend with the inherent errors of the encryption, making detection difficult. Consequently, silent data corruption is anticipated to become a prevalent challenge in environments susceptible to faults in homomorphic encryption. Consequently, we anticipate that this research will lay the groundwork for significant future investigations in this domain.

References

[1] K. Konstantinos, M. Persefoni, F. Evangelia, M. Christos, and N. Mara, "Cloud computing and economic growth," in *Proceedings of the 19th Panhellenic Conference on Informatics*, 2015, pp. 209–214.

[2] J. Sen, "Security and privacy issues in cloud computing," in *Cloud technology: concepts, methodologies, tools, and applications*. IGI global, 2015, pp. 1585–1630.

[3] H. Williams, "A modification of the RSA public-key encryption procedure (corresp.)," *IEEE Transactions on Information Theory*, vol. 26, no. 6, pp. 726–729, 1980.

[4] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[5] X. Yi, R. Paulet, E. Bertino, X. Yi, R. Paulet, and E. Bertino, *Homomorphic encryption*. Springer, 2014.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "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, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[7] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.

[8] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05, 2005, p. 84–93. [Online]. Available: https://doi.org/10.1145/1060590.1060603

[9] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

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