Remark on the Security of CKKS Scheme in Practice

Jung Hee Cheon, Seungwan Hong, and Duhyeong Kim

Seoul National University, South Korea

Abstract. Recently, Li and Micciancio (ePrint 2020/1533) have proposed a passive attack on the CKKS approximate homomorphic encryption (HE) scheme, which allows an adversary to query decryption on valid ciphertexts. In this paper, we discuss for which applications such attack is applicable, and introduce an extension of the HEaaN library. In addition, we investigate the mitigation strategies of other HE libraries that support the CKKS scheme including HElib, PALISADE, Lattigo and SEAL.

1 CKKS with IND-CPA Security

The Cheon-Kim-Kim-Song (CKKS) [10] scheme is a homomorphic encryption (HE) scheme which supports approximate computations of real (complex) numbers. As usual HE schemes such as BGV [8] and BFV [7, 11], the CKKS scheme addresses the IND-CPA security of which an adversary can query for the encryption oracle but not for the decryption oracle. Since it is essentially impossible to achieve the IND-CCA2 security in HE and is even hard to construct a practical IND-CCA1-secure HE scheme, the IND-CPA security has been the conventional security notion in these state-of-the-art HE schemes and their libraries.

Indeed, many applications of HE suffice to satisfy the IND-CPA security. In most of the scenarios of HE, a secret key owner provides her private data to others and receives outsourced data manipulation results. For instance, a client may request model-specific inference computation to model provider, such as outsourced matrix computation [13], secure logistic regression [16, 14], prediction phase of neural network [12], secure GWAS computation [15, 6], and training models for decision tree [5, 19].

In these scenarios, attacks using information leakage from decryption values cannot be applied since the adversary cannot access the decryption oracle. Thus the client can use the original decryption algorithm without any modification. However, in a practical perspective, it is quite reasonable to consider the case that decrypted values are shared with some other parties who do not own the secret key, and in this case the security cannot be covered by merely the IND-CPA notion.

2 New Security Notion: IND-CPA+

Recently Li and Micciancio [17] formalized a new refined security notion called IND-CPA+, which allows an adversary for decryption queries contrary to IND-CPA and similarly to IND-CCA. However, the main difference between IND-CPA+ and IND-CCA is whether an adversary is passive or active. More precisely, while an IND-CCA adversary can actively manipulate arbitrary ciphertexts, an IND-CPA+ adversary is allowed for decryption queries only on valid ciphertexts which are obtained by legitimate operations (e.g., encryption and homomorphic computation only through the public interfaces provided by HE libraries as mentioned in [17]).

They remarked that the IND-CPA+ security is equivalent to the IND-CPA security for exact HE schemes such as BGV [8] and BFV [7, 11], but the equivalence does not hold for the CKKS approximate HE scheme. The weakness of the CKKS scheme against IND-CPA+ adversaries mainly comes from the *linearity* of its decryption function to the secret key. To be precise, since the decryption function is of the form $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ctxt}) = \langle \mathsf{ctxt}, \mathsf{sk} \rangle \pmod{q}$, the secret key is fully recovered if the decrypted value and ctxt are given as a pair.

3 Extension of CKKS Against IND-CPA+ Attack in HEaaN

3.1 Our Suggestion : "Decryption" and "Decryption for Sharing"

The main reason for the security issue for approximate encryption is the revelation of decryption noise, which is a linear combination of the secret key. It can be easily prevented if the key owner does not share the decrypted values. However, some applications for approximate homomorphic encryption, such as collaboration with other cryptographic primitives including multi-party computation and differential privacy, have to share decrypted values from each party. In other words, we need to clearly distinguish between those cases, one to use the decryption algorithm and the other to modify it to satisfy stronger security. For the details, consider the following two scenarios.

In our settings, two parties are involved in each scenario: secret key owner and service provider. The secret key owner wants to share her data to service provider and receive manipulated result of them while the information of data is not leaked. For this purpose, she generates a HE secret key and shares the encrypted data. On the other hand, the service provider has its own data analysis model and wants to provide a service that manipulates and returns the client's data. To guarantee that the one who is not authorized to get the secret key cannot get any information from data in manipulation, the data provider can provide two different security levels.

1. *Prohibit client disclosing the results of decrypted messages.* In this scenario, attacks using information leakage from decryption values cannot be applied since the adversary cannot access to the decryption oracle. Thus the secret key owner can use the original decryption algorithm without any modification. However, to be ensured for the safety of her data, it is important that the decryption value should not be disclosed permanently.

2. *Allow secret key owner to share her decrypted messages to others.* In many real-world applications, applying manipulated results may cause us to share them with others. In this case, the secret key owner should conceal the decryption values to ensure the safety of data. Hence, she has to modify the decryption algorithm so that the message does not reveal any information of the secret key. We say this modified decryption algorithm by "decryption for sharing".

In the rest of the section, we focus on the second scenario and introduce the new decryption function for CKKS scheme in HEaaN library [1].

3.2 Updates in HEaaN

In HEaaN, we keep using the original decryption function Dec for the scenarios which are satisfactory with IND-CPA security and provide another decryption function DecForShare considering the IND-CPA+ security that decrypted values are shared with other parties who do not own the secret key.

For DecForShare, we follow one of the potential countermeasures suggested in [17] to add a proper error at the end of the original CKKS decryption process.

– $\mathsf{DecForShare}_{\mathsf{sk}}(\mathsf{ctxt}; B_{\mathsf{ctxt}})$: Sample a Gaussian error $e \leftarrow D_{\mathbb{Z}^n, B_{\mathsf{ctxt}}}$. Then compute $m = \langle \mathsf{ctxt}, \mathsf{sk} \rangle + e \pmod q$, and output $\mathsf{Decode}(m) \in \mathbb{C}^{N/2}$.

When one decrypts a ciphertext ctxt through $\mathsf{DecForShare}$, he is required to put some noise upper bound B_{ctxt} (w.r.t. the maximal norm $\|\cdot\|_\infty$) of ctxt as an input of $\mathsf{DecForShare}$. Then, a Gaussian error of the width B_{ctxt} is added at the end of the decryption procedure so that the extension induces at most 1-bit loss of the precision with high probability.

Remark 1. There is not yet a complete theoretical result of how to choose the tight bound of B_{ctxt}. However, assuming that the underlying CKKS scheme with $\mathsf{DecForShare}$ instead of Dec is *private-key setting* and an adversary can query the decryption only for *fresh ciphertexts*, one may choose much smaller additional noise than the above description with a theoretical guarantee. That is, the IND-CPA security of the modified CKKS scheme is equivalent to this weaker notion of the IND-CPA+ security under the hardness assumption of the multi-hint RLWE problem [9]. It is also shown in [9] that the multi-hint RLWE problem is at least as hard as the standard RLWE problem with a proper error width. It is left as a future work to extend the theoretical result in [9] to the IND-CPA+ security so that the methodology is finally adopted to HEaaN.

4 Mitigation of other HE Libraries for IND-CPA+

After the Li-Micciancio attack [17] on CKKS was disclosed, each HE library presented a mitigation strategy on the CKKS scheme to prevent the IND-CPA+ attack. At a high-level, they share the same idea to attach a proper error at the end of the decryption process to avoid the linearity on the secret key, which is in fact one of the potential countermeasures proposed in [17]. In this section, we investigate how the additional decryption error was applied in each library (e.g., the error size, adding error or rounding, etc.) based on the official comments and documents (e.g., `Security.md`) attached in the library.

HElib. The mitigation strategy of HElib [2] is precisely described in the `CKKS-security.md` document. Basically, as default they set the size of an additional decryption error equal to the noise bound of the ciphertext which is roughly estimated through `ctxt.errorBound` in the library. However, sometimes the ciphertext noise can be estimated over-conservatively, and it might yield a significant precision loss. For this reason, HElib let the application to determine the *required output precision*, and then the size of the additional decryption error is chosen *as large as possible* under this accuracy constraint. Based on the required output precision and the target computation, the application computes the input precision for encrypting plaintext data and take it as one of the inputs of the encryption function `ptxt.encrypt`. Under this setting, the decrypted value of the resulting ciphertext after the target computation gets to satisfy the predetermined output precision.

PALISADE. The PALISADE lattice cryptography library [4] takes an approach that the noise growth in the ciphertext can be traced by looking into the imaginary part if we only use the real part of each plaintext slot as a message. Precisely, the imaginary part of the plaintext is encoded by 0 as default, and those are not allowed to be used as messages in the application. As proceeding several operations among some ciphertexts, the noises in the imaginary parts also increase. In this perspective, PALISADE observes that the imaginary parts after decryption can be exploited to estimate the noise size in real parts. As a result, the size of the additional decryption error is determined by the estimated ciphertext noise and the number of allowed decryption queries. Please refer to `Security.md` in the library for more details.

Lattigo. In Lattigo library [3], they chose a strategy to take a *rounding* after the original decryption process, while the previous libraries added an key-independent error. To be Precise, they provide a new decoding function called `DecodeAndRound(ptxt, b)` which additionally takes an output precision b contrary to the original decoding function `Decode(ptxt)`. For the case that decrypted values need to be disclosed, the `DecodeAndRound` function would be used instead of `Decode`.

SEAL. Currently, a modification for IND-CPA+ security on algorithms or API does not appear in SEAL [18]. Instead, they noted in `SECURITY.md` that the decryption results of SEAL ciphertexts should be treated as private information only available to the secret key owner.

Bibliography

[1] HEaaN. `https://github.com/snucrypto/HEAAN`, 2018. SNUCRYPTO.

[2] HElib v1.3.0. `https://github.com/homenc/HElib`, 2020. IBM.

[3] Lattigo (`dev_indCPA+_mitigation` branch). Online: `https://github.com/ldsec/lattigo/tree/dev_indCPA%2B_mitigation`, dec 2020. EPFL-LDS.

[4] PALISADE v1.10.6. `https://gitlab.com/palisade/palisade-release`, Dec. 2020. PALISADE Project.

[5] A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald. Privacy-preserving decision tree training and prediction against malicious server. *IACR Cryptol. ePrint Arch.*, 2019:1282, 2019.

[6] M. Blatt, A. Gusev, Y. Polyakov, and S. Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020.

[7] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.

[9] J. H. Cheon, W. Cho, and D. Kim. Note on IND-CPA+ Security of CKKS. 2020.

[10] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[11] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[12] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[13] X. Jiang, M. Kim, K. Lauter, and Y. Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222, 2018.

[14] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):83, 2018.

[15] D. Kim, Y. Son, D. Kim, A. Kim, S. Hong, and J. H. Cheon. Privacy-preserving approximate gwas computation based on homomorphic encryption. *BMC Medical Genomics*, 13(7):1–12, 2020.

[16] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2):e19, 2018.

[17] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. Cryptology ePrint Archive, Report 2020/1533, 2020. `https://eprint.iacr.org/2020/1533`.

[18] Microsoft SEAL (release 3.0). `http://sealcrypto.org`, Oct. 2018. Microsoft Research, Redmond, WA.

[19] X. Xiao, T. Wu, Y. Chen, and X. Fan. Privacy-preserved approximate classification based on homomorphic encryption. *Mathematical and Computational Applications*, 24(4):92, 2019.