UNIVERSITY OF CALIFORNIA

Los Angeles

High Precision Bootstrapping of Approximate Homomorphic Encryption

A dissertation submitted in partial satisfaction

of the requirements for the degree Doctor of Philosophy

in Computer Science

by

Nathan Manohar

2021

ABSTRACT OF THE DISSERTATION

High Precision Bootstrapping of Approximate Homomorphic Encryption

by

Nathan Manohar

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2021

Professor Amit Sahai, Chair

The CKKS homomorphic encryption scheme is a homomorphic encryption scheme that supports approximate arithmetic over real/complex numbers. Due to its ability to natively compute over real numbers, the CKKS homomorphic encryption scheme is considerably more efficient than other schemes for many real world applications that naturally lend themselves to computation over real numbers. Such applications include, for example, privacy-preserving machine learning and secure genome analysis.

In the CKKS homomorphic encryption scheme, ciphertexts have an associated level, which is reduced as homomorphic computation is performed. Eventually, a ciphertext is at the lowest level, and no further homomorphic computation can be performed. In order to evaluate high depth circuits, it is necessary to bootstrap a ciphertext using a procedure called bootstrapping, which takes a ciphertext at the lowest level and increases its level so that additional homomorphic computation is possible. Unfortunately, the bootstrapping procedure for CKKS has a large associated error, and, prior to this work, it was not possible to perform high precision computations in CKKS since bootstrapping would reduce the precision of the plaintext. Obtaining high precision bootstrapping of CKKS is particularly important since many applications of CKKS require high precision computation.

In this dissertation, we show how to obtain high precision bootstrapping of CKKS. The main challenge is to find low-degree polynomial approximations of the mod function in small intervals around multiples of the modulus. We show the above by first showing how to approximate the mod function in ϵ-sized intervals around multiples of the modulus using a sine series, where our sine series of order n has error $O(\epsilon^{2n+1})$. This, after a Taylor series approximation of the sine function, results in a low-degree polynomial approximation of the mod function with small coefficients that can be used to approximate the mod function to arbitrary precision, resulting in practical high precision bootstrapping of the CKKS homomorphic encryption scheme. We validate our approach by an implementation and obtain 100 bit precision bootstrapping as well as improvements over prior work even at lower precision.

The contents of this dissertation are based on a joint work with Charanjit S. Jutla.

The dissertation of Nathan Manohar is approved.

University of California, Los Angeles

2021

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Amit Sahai, for his constant support throughout my entire PhD. Amit is truly an inspiration; not only is he a fantastic researcher, he is also an amazing advisor. Despite being so many years removed from his own graduate studies, he is able to seamlessly put himself in the shoes of his grad students and understand their challenges. One of his greatest qualities as an advisor is his ability to inspire confidence in his students. When starting out, conducting research can feel like trying to drink from a fire hose. You are inundated with so much information that it can feel impossible to learn everything. Amit's ability to shepherd students through this flood of information is unparalleled. He has a great talent for taking challenging problems and breaking them down into smaller, achievable goals. During my time at UCLA, not only have I learned a lot from Amit about tackling problems and mentoring students that I hope to carry with me for my entire career, but I have also made a lifelong friend.

I would also like to especially thank my long-term collaborator and mentor, Charanjit S. Jutla. I first met Charanjit during the summer of 2020, when I interned at the IBM T.J. Watson Research Center. Despite the internship being fully remote, we were able to work together effortlessly. I have learned a tremendous amount from working with Charanjit and have greatly enjoyed our discussions about research and life. The contents of this dissertation are based on a work that resulted from this collaboration.

I am grateful to Nicholas Genise for his mentorship and collaboration during my internship at SRI International. I had a lot of fun discussing ideas with Nick, making my internship a memorable experience. I am very fortunate to have Aayush Jain as both a collaborator and a good friend. His willingness to answer my questions during the early stages of my PhD was invaluable. I would also like to thank Alexis Korb and Paul Lou for being great labmates and friends. I will fondly remember the countless hours we spent hanging out and exploring LA. I am also grateful to Saikrishna Badrinarayanan for being a great collaborator and friend.

I would also like to thank all my co-authors: Prabhanjan Ananth, Saikrishna Badrinarayanan, Dan Boneh, Rishab Goyal, Aayush Jain, Abhishek Jain, Charanjit S. Jutla, Sam Kim, Dmitry Kogan, Alexis Korb, Peter Manohar, Rajit Manohar, Amit Sahai, Brent Waters, and David J. Wu.

Many thanks to Victor Shoup and Karim Eldefrawy for their mentorship during my internships and Yuval Ishai, Alex Lombardi, and Fermi Ma for many enlightening discussions during their visits to UCLA. I would also like to thank Raghu Meka, Rafail Ostrovsky, and Alexander Sherstov for serving as members of my dissertation committee and for teaching excellent courses. I have learned a lot from them during my time at UCLA.

I am also thankful to many people who were influential in my decision to study cryptography. As an undergrad, Salil Vadhan's many courses (in particular, Introduction to Cryptography) gave me a great appreciation for cryptography and theoretical computer science. Serving as a TA for Boaz Barak's cryptography course also grew my passion for the subject. Prabhanjan Ananth, Akshay Degwekar, and Prashant Vasudevan were very supportive during a summer I spent at the Simons Institute as an undergrad and gave me insight into what life would be like as a PhD student. During the year I spent at Stanford, Dan Boneh introduced me to more applied cryptography and impressed me with his ability to identify excellent research questions, a trait I hope to be able emulate during my career. I also had the pleasure of working with Moses Charikar and Mary Wootters, who both helped broaden my horizons and learn about research in other areas of theoretical computer science. David J. Wu also graciously took the time to answer my many questions and discuss interesting research directions.

I would also like to thank my labmates and fellow students Dakshita, Saikrishna, Aayush, Rex, Alexis, Paul, Riddhi, Alain, Ashutosh, Pei, Eli, Kevin, Andrey, Hadley, Shuyang, and Poorva for making my time at UCLA a thoroughly enjoyable experience. I am also very grateful for my friends Nick G., Nick R., Tom, David D., Remy, Bryce, Jacqueline, David G., and Saba that have kept in touch and supported me throughout my PhD.

I would also like to thank my cat, Mary, for her support during the writing of this dissertation. Despite her best efforts, she is not, to the best of my knowledge, responsible for any of the typographical errors that may be present in this work.

Finally, none of this would have been possible without the love and support of my family. My brother, Peter, has always supported me and pushed me to be the best I can be, both as a researcher and a person. My parents instilled in me a love for science and mathematics at a young age and have always encouraged and inspired me to explore my passions. This dissertation is dedicated to them.

VITA

2016	A.B. in Computer Science and Mathematics, Harvard University.
2016	S.M. in Computer Science, Harvard University.
2016-2017	Graduate Student Researcher, Stanford University.
2020	Research Intern at IBM T.J. Watson Research Center.
2021	Research Intern at SRI International.

SELECTED PUBLICATIONS

Secure MPC: Laziness Leads to GOD

Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, Amit Sahai

ASIACRYPT 2020

Amplifying the Security of Functional Encryption, Unconditionally

Aayush Jain, Alexis Korb, Nathan Manohar, Amit Sahai

CRYPTO 2020

Self-Processing Private Sensor Data via Garbled Encryption

Nathan Manohar, Abhishek Jain, Amit Sahai

PETS 2020

Combiners for Functional Encryption, Unconditionally

Aayush Jain, Nathan Manohar, Amit Sahai

EUROCRYPT 2020

From FE Combiners to Secure MPC and Back

Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, Amit Sahai

TCC 2019

Watermarking Public-Key Cryptographic Primitives

Rishab Goyal, Sam Kim, Nathan Manohar, Brent Waters, David J. Wu

CRYPTO 2019

T/Key: Second-Factor Authentication From Secure Hash Chains

Dmitry Kogan, Nathan Manohar, Dan Boneh

CCS 2017

Chapter 1

Introduction

In its most primitive form, cryptography deals with methods of communicating secret messages between parties in a manner that hides the contents of these messages from other external parties. Such methods are referred to as *encryption schemes* and come in two types: private (or symmetric)-key encryption schemes and public-key encryption schemes. Private-key encryption schemes are the simplest form of encryption schemes and in these schemes, there is a shared secret key sk that can be used to encrypt and decrypt messages. In public-key encryption schemes, the key consists of two parts: a public key pk and a secret key sk. Encryption only requires the public key pk and, thus, anybody is capable of encrypting messages in a public-key encryption scheme. While encrypting data succeeds in hiding the data from external parties, it comes at the cost of eliminating functionality. For example, if a client wants to store data on some external untrusted server, they can use an encryption scheme to store their data in encrypted form on the server. However, if the client later wants to learn something about the data (such as search for a particular entry), the client is unable to do so easily because the data stored on the server is encrypted. This basic challenge has led to modern cryptography exploring methods of computing on encrypted data. Encryption schemes that are capable of computing on encrypted data are referred to as *homomorphic encryption schemes*.

1.1 Homomorphic Encryption

A homomorphic encryption scheme is an encryption scheme that additionally possesses an evaluation procedure. This evaluation procedure takes as input a function f and a ciphertext ct that encrypts some message m and returns a ciphertext ct' that encrypts $f(m)$. To make such a notion non-trivial (and useful), we additionally require that the size of the ciphertext ct' does not grow with the size of f. A homomorphic encryption scheme that is capable of evaluating all efficiently computable functions f is called a *fully homomorphic encryption scheme*.

The problem of constructing fully homomorphic encryption (FHE) has a rich history, and, it turns out, for a long time, it was not clear if it was even possible to construct FHE. In 1978, Rivest, Adleman, and Dertouzos [RAD78] formalized the theoretical notion of FHE and noted how incredibly useful it would be if it could be constructed. While it was known how to construct somewhat homomorphic encryption schemes (RSA encryption [RSA78] is multiplicatively homomorphic and El-Gamal encryption [ElG84] can be made either additively or multiplicatively homomorphic), at that time, it was unknown if constructing FHE was even possible. For about three decades, this was the state of affairs until the world of cryptography changed in 2009 with Craig Gentry's breakthrough result [Gen09], which gave the first candidate construction of FHE that supported computing any general functionality. This candidate was revolutionary because it was the first feasibility result, but at the same time, it was very far from practical. However, over the last decade or so, cryptography researchers have a put a lot of effort into improving the performance of FHE constructions and now, we have several FHE schemes and implementation libraries that can be downloaded and ran on a commodity laptop. Table 1.1 gives an overview of the state of the art FHE schemes. These FHE schemes support different circuit types and plaintext types and, thus, the FHE scheme one should use is determined by the application. However, these schemes all share several common characteristics. For one, the security of all these FHE schemes relies on the hardness of the learning with errors problem (LWE) [Reg09] or its ring variant ring-LWE [LPR10]. Due to the reliance on this computational hardness assumption, ciphertexts in these FHE schemes all have an associated "error" that grows during homomorphic computation. Decryption operates by first recovering a noisy value and then "rounding" away the error to obtain the message in the clear. This "error" enforces a computational budget. After this computational budget has been exhausted, further homomorphic computation cannot be performed. Following Gentry's blueprint [Gen09], these FHE schemes possess a bootstrapping procedure, which "refreshes" the computational budget of a ciphertext so that additional homomorphic computation can be performed. By repeatedly alternating between computing homomorphically and bootstrapping, these FHE schemes support unbounded computation. In this dissertation, we will focus on improving the bootstrapping procedure of the CKKS homomorphic encryption scheme.

Table 1.1: Overview of FHE Schemes

Scheme	Circuit Type	Plaintext Type
BGV [BGV12]	Arithmetic	Mod p
BFV [Bra12, FV12]	Arithmetic	Mod p
GSW [GSW13]	Boolean	Bits
FHEW [DM15]	Boolean	Bits
TFHE [CGGI20]	Boolean	Bits
CKKS[†] [CKKS17]	Arithmetic	Real/Complex

[†] Only for approximate arithmetic

1.2 CKKS Overview

The work of [CKKS17] presented a new homomorphic encryption (HE) scheme for approximate arithmetic (called the CKKS-HE scheme) over real/complex numbers. Due to its ability to natively approximately evaluate arithmetic circuits, the CKKS-HE scheme is considerably more efficient than other schemes for many real world applications that naturally lend themselves to computation over real numbers. One of the key insights of [CKKS17] is to treat the ciphertext "error" as part of the approximate arithmetic error, and, thus, no additional mechanism is required to round away the ciphertext error after decryption. The CKKS-HE scheme has found many applications, among them privacy-preserving machine learning and secure genome analysis (see [KSK$^+$18, MHS$^+$20, BHHH19, KSW$^+$18, SPTP$^+$20, KHB$^+$20] for some examples).

However, the initial CKKS-HE scheme was only capable of evaluating low depth circuits since it lacked a bootstrapping procedure to "refresh" the computational budget of a ciphertext to enable further homomorphic computation. In CKKS, ciphertexts have an associated level and homomorphic multiplication consumes a level. Thus, if a fresh ciphertext was at level L, then one can only evaluate arithmetic circuits with multiplicative depth $L-1$ until one ends up with a ciphertext at level 1, on which no additional homomorphic computation can be performed. Therefore, the initial CKKS-HE scheme is a *leveled* homomorphic encryption scheme, since there is a bound on the depth of circuits it can evaluate, and the parameters of the scheme scale with this depth. The consequence of this was that many real world applications (computations with multiplicative depth greater than $\approx$ 20–25) were practically infeasible due to the computational inefficiency caused by the parameter growth.

1.2.1 Bootstrapping of CKKS Homomorphic Encryption

This was remedied when [CHK$^+$18a] introduced the first bootstrapping procedure for the CKKS-HE scheme which enabled, for the first time, arbitrary depth computations in CKKS. An observant reader may note that supporting arbitrary depth computations seems contradictory to the fact that CKKS is only for approximate arithmetic. In particular, as more computation is performed, the precision of the plaintext value decreases until, eventually, the encrypted message is nothing but noise. While this is true, bootstrapping for CKKS is essential since, as discussed above, it enables evaluating larger depth circuits without greatly increasing the parameter sizes, making evaluating such circuits practically feasible. One critical point to note is that since CKKS is only for approximate arithmetic, the bootstrapping procedure for CKKS differs from that of other FHE schemes. In particular, bootstrapping in other FHE schemes follows Gentry's blueprint [Gen09] and involves evaluating the decryption circuit homomorphically on an encryption of the secret key in order to reduce the ciphertext error to enable further homomorphic computation. However, since decryption in CKKS is only approximate, the standard bootstrapping procedure would not succeed in reducing the ciphertext error. Instead, the goal of CKKS bootstrapping is only to refresh the ciphertext level (from 1 to some higher level L') to enable further homomorphic computation, and the ciphertext error actually *increases* as a result of CKKS bootstrapping. Unfortunately, the error associated with the bootstrapping procedure in [CHK$^+$18a] is so large that only the most significant 20–25 bits of the message are preserved. The consequence of this is that it was not possible to perform high precision computation in CKKS since as soon as bootstrapping was required, all but the most significant bits of the message would be destroyed. This naturally leads to the question that is the focus of this dissertation

How can we limit the error growth associated with the bootstrapping procedure so that high precision computations can be performed?

The reason obtaining high precision bootstrapping for CKKS is important is that one of the main applications of CKKS is privacy-preserving machine learning. However, many ML algorithms require high precision computation in order to converge. This may be especially true during the learning phase of neural networks, which involves back propagation and integer division by private integers. Additional nonlinear steps involve pooling functions, threshold functions, etc. Moreover, due to their high depth, computing these ML algorithms homomorphically without bootstrapping is infeasible. Thus, for privacy-preserving ML applications, high precision bootstrapping is required.

The State of the Art Prior to This Work

To understand the challenges associated with achieving high precision bootstrapping in CKKS, we will first give an overview of the state of the art prior to this work. All bootstrapping procedures for CKKS (including this work) follow the general bootstrapping template introduced in [CHK$^+$18a]. As mentioned previously, ciphertexts in CKKS have an associated level from 1 to L. The level ℓ of the ciphertext determines the ciphertext modulus q_ℓ, which can be thought of as q^ℓ for some fixed base modulus q. Recall that the goal of bootstrapping is to take a ciphertext ct at level 1 (with respect to modulus q) and transform it into a new ciphertext ct' at some higher level ℓ to enable further homomorphic computation. For bootstrapping to be useful, we necessarily require that ct' encrypts approximately the same message as ct. Bootstrapping for CKKS involves viewing a ciphertext ct with a small modulus q as a ciphertext with respect to the largest modulus q_L and then homomorphically computing coefficient rounding modulo q to obtain a new ciphertext ct' that encrypts approximately the same message as ct with respect to a larger modulus q_ℓ, enabling further homomorphic computation. Thus, a challenge here is to compute the mod function homomorphically, which is not easily representable via an arithmetic circuit. In fact, the mod function modulo q on the interval $[-Kq, Kq]$ for some integer K is not even a continuous function. However, [CHK$^+$18a] made the clever observation that in the CKKS-HE scheme, we have an upper bound m on the size of the message, which can be made much smaller than q. In this situation, we actually only need to be able to compute the mod function on points in $[-Kq, Kq]$ that are a distance at most m from a multiple of q. In this case, the mod function is periodic with period q and is linear on each of the small intervals around a multiple of q. Figure 1.1 shows the mod function along with the intervals for approximation.

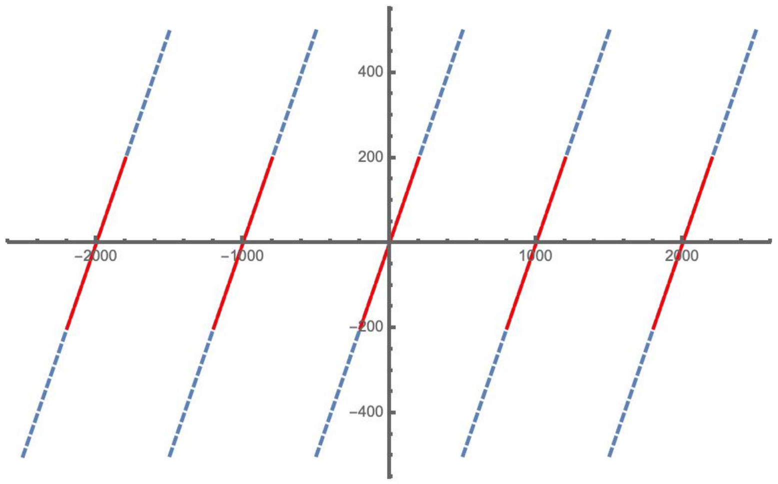
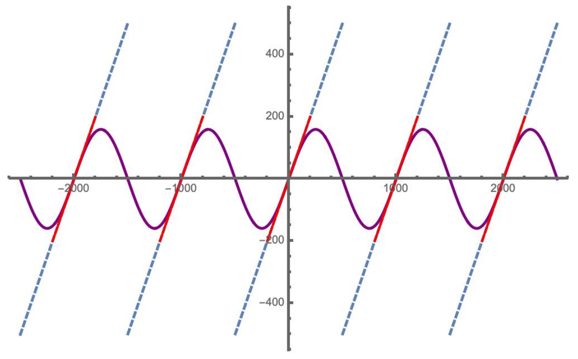


Figure 1.1: The mod function with modulus $q = 1000$. The solid red lines represent the intervals on which we need to approximate.

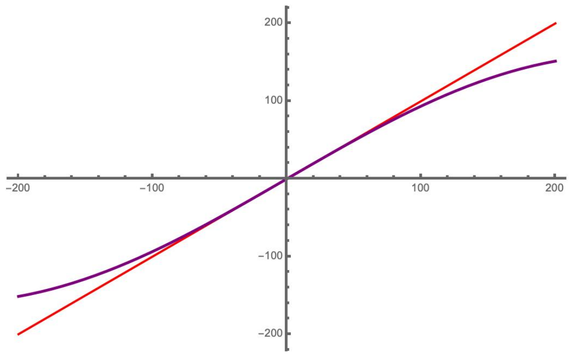
The work of [CHK$^+$18a] further observed that the mod function $[t]_q$ on these intervals can be approximated via a scaled sine function $S(t) = \frac{q}{2\pi}\sin\left(\frac{2\pi t}{q}\right)$. This approximation introduces an inherent error that depends on the message upper bound m. Let ϵ denote the ratio $\frac{m}{q}$. Then, it can be shown that

$$|[t]_q - S(t)| \leq \frac{2\pi^2}{3} q\epsilon^3.$$

If ϵ is small enough, then this error can be sufficiently small for use in bootstrapping provided that $S(t)$ can be well-approximated by a low degree polynomial. The work of [CHK$^+$18a] along with several followup works [CCS19, HK20] proceeded to provide methods of approximating this scaled sine function (or scaled cosine function in the case of [HK20]) by a low-degree polynomial, which can then be plugged into the bootstrapping procedure of [CHK$^+$18a]. However, due to the inherent error between the mod function $[t]_q$ and the scaled sine function $S(t)$, this approach has a "fundamental error" that will occur regardless of how $S(t)$ is approximated. Figure 1.2a shows the scaled sine approximation of the mod function and Figure 1.2b shows this approximation zoomed in on the interval around the origin.



(a)



(b)

Figure 1.2: Scaled sine approximation of the mod function.

One of the problems with this is that in order for the error to be $O(1)$ (and, therefore, not destroy the message), m must be $O(q^{2/3})$. This means that we must begin bootstrapping while the size of the encrypted message is considerably smaller than q, which is a source of inefficiency in the bootstrapping procedure, particularly in applications that require high precision. An even greater problem is that when homomorphically computing the mod function, we must treat $qI + m$ for some integer I as the input, which we refer to as the bootstrapping plaintext. The issue with this is that if q is significantly larger than m, then since the number of modulus bits "consumed" by each homomorphic multiplication of the mod function is the size of the bootstrapping plaintext, these homomorphic multiplications will consume significantly more modulus bits than normal homomorphic operations. Thus, it is *inefficient* to obtain high precision bootstrapping by simply increasing q to decrease ϵ. Instead, in order to obtain high precision bootstrapping, it is beneficial to obtain good polynomial approximations to the mod function for fixed ϵ. An additional challenge to obtaining high precision bootstrapping is that the approximation to the mod function must be representable by a *low-degree* polynomial. If the degree of the polynomial is too high, evaluating it homomorphically may consume almost all of the ciphertext modulus, leaving the ciphertext after bootstrapping incapable of performing many homomorphic operations. Compounding this challenge is the fact that the coefficients of the low-degree polynomial approximation to the mod function must additionally be small. This is because if the coefficients are large, when evaluating the polynomial, the basis polynomials must be computed to higher precision to ensure the stability of the computation, since errors introduced by approximate arithmetic are amplified by large coefficients.

Recently, the works of [LLL^{+}21] and [JM20] were able to bypass the "fundamental error" in the approximation of the mod function by a scaled sine function to obtain higher-precision bootstrapping. [LLL^{+}21] attempts to avoid the scaled sine function by finding the optimal minimax polynomial of a fixed degree that approximates the mod function via algorithmic search. They use a variant of the Remez algorithm [Rem34] to obtain an approximation to the optimal minimax polynomial of a given degree that approximates the modular reduction function on the union of intervals containing points close to multiples of q. Unfortunately, as observed by [LLL^{+}21], the size of the coefficients of these polynomials are too large to enable high precision bootstrapping. They then show that by using a composition of sine/cosine and the inverse sine function and using the Remez algorithm to algorithmically search for good polynomial approximations to these functions, one can obtain higher-precision bootstrapping, but their bootstrapping method has only been shown to obtain 40 bit message precision in the latest version of their work. [JM20] avoids the “fundamental error” by finding direct polynomial approximations of the mod function on small intervals around the modulus via a new technique called modular Lagrange interpolation. The coefficients of these polynomials were small enough to enable high precision bootstrapping. However, the coefficients were still large enough that in order to evaluate the polynomial approximations, one would need to operate at a higher precision than the bootstrapping plaintext. Ultimately, this fact corresponded to the bootstrapping procedure losing additional levels, since the computations during bootstrapping were operating at a higher precision. The authors are able to obtain 67 bit precision bootstrapping in the latest version of their work.

1.3 This Work

In this work, we show how to obtain arbitrary precision bootstrapping via a different method from that of [JM20] and more in line with the original sine function approach of [CHK$^+$18a]. Instead of approximating the mod function directly, we first approximate the mod function by a sine series and then approximate the sine function by its Taylor series (more precisely, the Taylor series of e^{ix}). This is then followed by a series of squarings to approximate the other terms in the sine series. We show that the sine series converges to the mod function in small intervals around the modulus. In particular, our sine series of order n has error $O(\epsilon^{2n+1})$ for approximating the mod function in ϵ-sized intervals around multiples of the modulus. Figure 1.3 shows our sine series approximation of the mod function of order 4. Observe that the red approximation regions are completely covered by our approximation.

Thus, we avoid the fundamental error of the scaled sine approach and are able to obtain

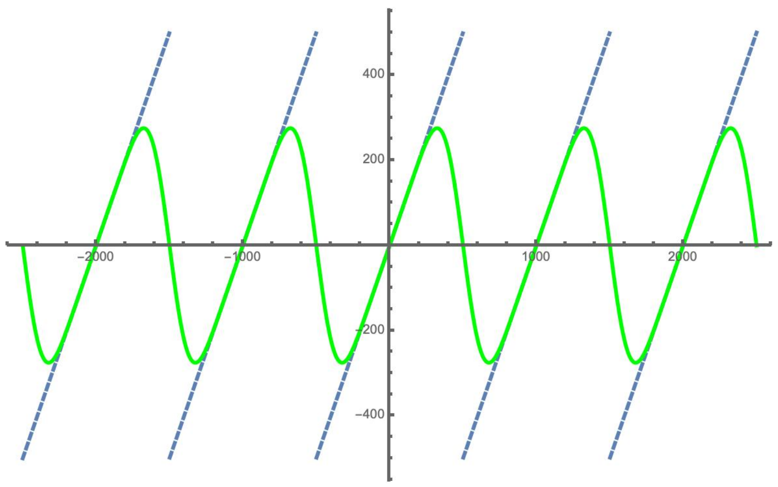


Figure 1.3: Our sine series approximation of the mod function of order 4.

an approximation with arbitrarily small error in the desired intervals. Furthermore, the coefficients of the sine series are small (in fact, they have norm < 2). This, combined with the fact that the Taylor series expansion of $\sin x$ has small coefficients, leads to a polynomial approximation of the mod function with small coefficients. Due to these small coefficients, the whole polynomial can be computed at a precision only slightly larger than $(-2n-1)\log\epsilon$, the precision of the approximation being sought.

We validate our approach by an implementation and obtain 100 bit precision bootstrapping as well as improvements over prior work even at lower precision.

1.3.1 Problem Overview

Here, we provide a brief overview of the challenges of approximating the mod function for use in CKKS-HE bootstrapping. We provide a thorough overview of the bootstrapping procedure in Chapter 2. Recall, the goal of CKKS-HE bootstrapping is to take a ciphertext ct at the lowest level and bring it up to the highest level so that homomorphic computation can continue. In other words, we wish to obtain a ciphertext ct' such that

$$\langle \mathsf{ct}, \mathsf{sk} \rangle \bmod q \approx \langle \mathsf{ct}', \mathsf{sk} \rangle \bmod q_\ell,$$

where q is the lowest level modulus and q_ℓ represents a higher level modulus. Since errors accumulated during homomorphic computation are not eliminated by decryption in CKKS-HE, the goal is not to reduce the error in the ciphertext, but, rather, to increase the modulus so that more computations can be performed. If one simply views the ciphertext ct as operating at the highest level q_L, then it follows that $\langle \mathsf{ct}, \mathsf{sk} \rangle \bmod q_L = qI + m$. The magnitude of I can be upper bounded and $m \ll q$ and, thus, the challenge then becomes to compute mod q on small intervals near multiples of q (we defer additional complications such as computing on slots vs. coefficients to Chapter 2). Since CKKS-HE can compute homomorphic additions and multiplications, we need a polynomial approximation to the mod function. However, there are three crucial criteria that are relevant to the bootstrapping application.

- **Error:** The error of the approximation contributes additional error to the message m, which, if large, will cause a loss in plaintext precision.

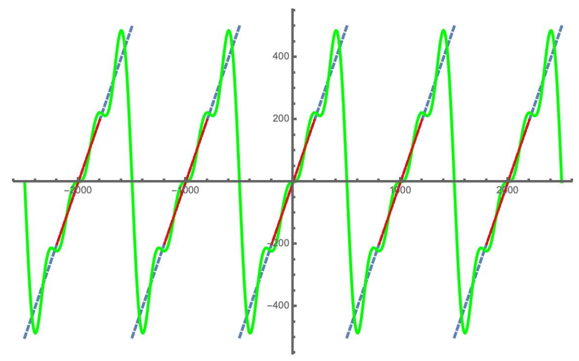
- **Degree:** The degree of the polynomial approximation determines the multiplicative depth required to evaluate it. A larger multiplicative depth corresponds to losing more modulus levels and, thus, if too large, the polynomial will not be able to be evaluated homomorphically.

- **Coefficient Magnitude:** The size of the coefficients of the polynomial approximation determine the "evaluation precision" at which one must operate during bootstrapping. Larger coefficients correspond to a larger "evaluation precision" in order to maintain numerical stability, which, in turn, corresponds to losing more modulus bits per level.

Thus, it is critical that we obtain good low-degree polynomial approximations to the mod function in small intervals around multiples of the modulus that additionally have small coefficients. Moreover, as discussed previously, it is important the ratio $m/q = \epsilon$ is not too small, since then the size of the bootstrapping plaintext $qI + m$ will be significantly larger than m, and homomorphically evaluating the approximation to the mod function will consume a large number of modulus bits. Thus, one can think of ϵ as fixed to be, say 2^{-10}.

1.3.2 Sine Series

As mentioned previously, several prior approaches to CKKS-HE bootstrapping approximated the mod function via a scaled sine function. For simplicity, we will ignore the scaling for the moment and try to obtain a good approximation to the mod 2π function. Thus, prior works used $\sin x$ as an approximation of this function and noted that, for $|x| < \epsilon$, the error of approximation is $O(\epsilon^3)$. It is well-known that the Fourier series of the mod function (or sawtooth function) converges everywhere except the discontinuities. Unfortunately, the rate of convergence is too slow, and the Fourier series does not give a good approximation when the number of terms is small. Figure 1.4 demonstrates this, with Figure 1.4a showing the Fourier series approximation of the mod function of order 4 and Figure 1.4b showing this same approximation zoomed in on the interval around the origin, with the scaled sine approximation plotted in purple for comparison.


(a)

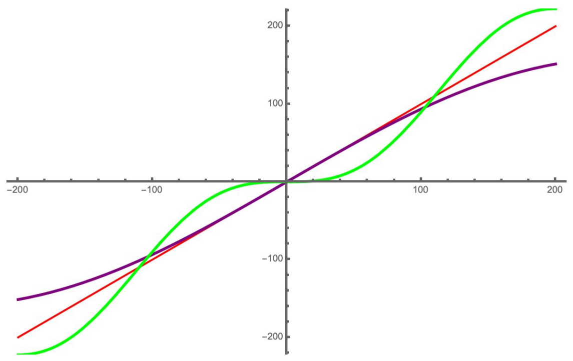

(b)

Figure 1.4: Fourier series approximation of the mod function of order 4.

Instead, we will approximate the mod function by a different sine series such that it converges to the mod function near multiples of the modulus very quickly. As a warmup, suppose we added a $\sin 2x$ term to our approximation of the mod function. If we can determine coefficients β_1 and β_2 such that the Taylor series expansion of $\beta_1 \sin x + \beta_2 \sin 2x$ is $x + x^5 p(x)$ for some polynomial $p(x)$, then for $|x| < \epsilon$, the error of approximation will be $O(\epsilon^5)$, an improvement on $\sin x$. Thus, looking at the x and x^3 terms in the Taylor series expansions of $\sin x$ and $\sin 2x$, we wish to determine β_1, β_2 such that $\beta_1 + 2\beta_2 = 1$ (so that the coefficient of x is 1) and $\beta_1 + 2^3\beta_2 = 0$ (so that the coefficient of x^3 is 0). This can be solved to yield $\beta_1 = 4/3, \beta_2 = -1/6$. This intuition can then be extended to give an n-term sine series with error $O(\epsilon^{2n+1})$. We will show that the β_i's are small and, thus, the resulting low-degree polynomial approximation has small coefficients. Moreover, we will show that the constants hiding in the big-O notation are reasonable, and the dependence on n is minor.

On Approximating Arcsine

An alternative way to view our result is that having computed the periodic function $\sin x$, our sine series allows us to compute arcsin (of $\sin x$) using an angle-multiplication computation. In other words, since we showed above that $x = 4/3 \sin x - 1/6 \sin 2x + O(x^5)$ (for small x, and hence small $\sin x$), then equivalently $\arcsin y = 4/3\ y - 1/6\ d(y) + O(y^5)$, where d is a function such that $d(\sin x) = \sin 2x$. However, $d(\sin x)$ is not a simple polynomial function of $\sin x$ (as opposed to the easy double-angle formula for $\cos x$), and this way of computing $\arcsin y$ cannot use a simple polynomial of y. While good polynomial approximations of $\arcsin y$ might exist (for small y), there seems no simple methodology to obtain this. Instead, [LLL$^+$21] use the Remez algorithm to obtain a best fit low degree polynomial approximation of arcsin. This algorithmic approach has the drawback that while the polynomial degree maybe small, the coefficients of the polynomial output by Remez algorithm can be of arbitrary size. Fortunately, [LLL$^+$21] report that the coefficients are small enough to obtain 40-bit precision bootstrapping, although it is not clear if this holds in general.

Our approach is different, as we utilize the potential of CKKS-HE to compute on complex numbers. Thus, instead of first computing $\sin x$ and then its arcsin, we first compute the periodic function e^{ix} (using its Taylor series approximation) and then compute its logarithm. Thus, given that $x = 4/3 \sin x - 1/6 \sin 2x + O(x^5)$, we also get that $x = \mathrm{Im}(4/3 e^{ix} - 1/6 e^{2ix}) + O(x^5)$ (for small x). Most importantly, it is a polynomial in its argument (i.e. e^{ix}) with small coefficients. Thus, this allows for an easy homomorphic computation.

1.3.3 Organization

In Chapter 2, we describe the CKKS-HE scheme and its bootstrapping procedure. In Chapter 3, we formalize the above intuition and prove explicit error bounds for the sine series approximation of the mod function. In Chapter 4, we implement bootstrapping using our sine series approximation and give performance metrics and comparisons with prior approaches.

Chapter 2

The CKKS Homomorphic Encryption Scheme

In this chapter, we describe the CKKS homomorphic encryption scheme [CKKS17] and its bootstrapping procedure introduced in [CHK$^+$18a].

2.1 Defining Approximate Homomorphic Encryption

A public-key approximate homomorphic encryption scheme $\mathsf{AHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ consists of four probabilistic polynomial-time (PPT) algorithms. Let $\mathcal{M} = \mathcal{M}_{\lambda\lambda\in\mathbb{N}}$ denote the message space of the scheme with an associated norm $||\cdot||$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ denote the circuit family of the scheme with outputs in $\mathcal{M}$. Let E denote an error function that takes as input the security parameter 1^λ and a circuit $C \in \mathcal{C}$ (or $\perp$) and outputs an upper bound on the approximate homomorphic computation error. The algorithms of AHE have the following syntax.

- $\mathsf{KeyGen}(1^\lambda)$: On input the security parameter 1^λ, KeyGen outputs the public key, secret key, and evaluation key $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$.

- $\mathsf{Enc}(\mathsf{pk}, m)$: On input the public key pk and a message $m \in \mathcal{M}$, Enc outputs a ciphertext ct.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: On input the secret key sk and a ciphertext ct, Dec outputs a message $m' \in \mathcal{M}$.

- $\mathsf{Eval}(\mathsf{evk}, C, \vec{\mathsf{ct}})$: On input the evaluation key evk, a circuit $C \in \mathcal{C}$ with ℓ inputs, and vector of length ℓ of ciphertexts $\vec{\mathsf{ct}}$, Eval outputs a ciphertext ct'.

We require an approximate homomorphic encryption scheme to satisfy the following properties.

Definition 2.1.1 (Approximate Correctness)**.** For all $\lambda \in \mathbb{N}$, for all $m \in \mathcal{M}_\lambda$,

$$\Pr\left[\begin{array}{c} (\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ ||\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) - m|| \leq E(1^\lambda, \perp) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

and for all $\lambda \in \mathbb{N}$, for all circuits $C \in \mathcal{C}_\lambda$ with ℓ inputs, for all $m_1, m_2, \ldots, m_\ell \in \mathcal{M}_\lambda$,

$$\Pr\left[\begin{array}{c} (\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i) \text{ for } i \in [\ell] \\ \mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{evk}, C(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_\ell)) \\ ||\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') - C(m_1, m_2, \ldots, m_\ell)|| \leq E(1^\lambda, C) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Definition 2.1.2 (Compactness)**.** There is a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, for all circuits $C \in \mathcal{C}_\lambda$ with ℓ inputs, for all $m_1, m_2, \ldots, m_\ell \in \mathcal{M}_\lambda$,

$$\Pr\left[\begin{array}{c} (\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i) \text{ for } i \in [\ell] \\ \mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{evk}, C(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_\ell)) \\ |\mathsf{ct}'| \leq \mathsf{poly}(\lambda) \end{array}\right] = 1.$$

Definition 2.1.3 ($\mathsf{IND\text{-}CPA}$ Security)**.** For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, the advantage of $\mathcal{A}$ is

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND-CPA}} = \left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where for each $b \in \{0, 1\}$ and $\lambda \in \mathbb{N}$, the experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, b)$ is defined as follows:

1. Generate $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and send $(\mathsf{pk}, \mathsf{evk})$ to $\mathcal{A}$.

2. $\mathcal{A}$ outputs a pair of messages $m_0, m_1 \in \mathcal{M}_\lambda$.

3. Run $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ and send ct to $\mathcal{A}$.

4. $\mathcal{A}$ outputs a bit b', and the output of the experiment is set to b'.

For certain applications, the notion of IND-CPA security may be insufficient since it does not provide any guarantee if the results of decryption (which are only approximately correct) are given to the adversary. We examine this issue in more detail when we discuss the security of the CKKS homomorphic encryption scheme later in this chapter.

2.2 CKKS Preliminaries

Throughout, let $\mathbb{Z}_q$ denote the integers modulo q in balanced representation (in $[-q/2, q/2)$). Let N be a power of 2 and $\Phi_{2N}(X) = X^N + 1$ be the $2N$th cyclotomic polynomial of degree N. Let $\mathcal{R} = \mathbb{Z}[X]/\Phi_{2N}(X)$. For an integer q, let $\mathcal{R}_q = \mathbb{Z}_q[X]/\Phi_{2N}(X)$. We refer to N as the ring dimension of $\mathcal{R}$. The $2N$th roots of unity are $e^{\frac{2\pi i}{2N}k}$ for $k = 1, 2, \ldots, 2N$, and the $\phi(2N) = N$ primitive $2N$th roots of unity are $e^{\frac{2\pi i}{2N}k}$ for odd integers k between 1 and $2N$. The N primitive $2N$th roots of unity are the roots of the $2N$th cyclotomic polynomial $\Phi_{2N}(X)$. Let $\zeta = e^{\frac{2\pi i}{2N}}$. The Galois group of $\mathbb{Q}[\zeta]/\mathbb{Q}$ is given by the automorphisms σ_k that map ζ to ζ^k for odd integers k between 1 and $2N$. This Galois group is isomorphic to $\mathbb{Z}_{2N}^* \cong \mathbb{Z}_{N/2} \times \mathbb{Z}_2$.

The security of the CKKS homomorphic encryption scheme is based on the ring-LWE (RLWE) assumption [LPR10]. We will use a version of the RLWE assumption defined for cyclotomic rings of degree N for N a power of 2 and secret keys with low Hamming weight.

Let $\mathcal{R}$ and $\mathcal{R}_q$ be defined as above. Fix $s \in \mathcal{R}$. For a real $\sigma > 0$, let χ denote the distribution over $\mathcal{R}$ obtained by sampling each coefficient independently from the discrete Gaussian distribution with variance σ^2. Let $\mathcal{D}_{s,\chi}$ denote the distribution over $\mathcal{R}_q^2$ obtained by sampling $a \leftarrow \mathcal{R}_q$ uniformly at random, sampling $e \leftarrow \chi$, and outputting $(a, b = a * s + e \bmod q)$. Let $\mathcal{S}$ denote a distribution over $\mathcal{R}$.

Definition 2.2.1 (Ring-LWE Assumption)**.** No probabilistic polynomial-time adversary $\mathcal{A}$, given polynomially many independent samples, can distinguish with non-negligible advantage between the distribution $\mathcal{D}_{s,\chi}$ with $s \leftarrow \mathcal{S}$ and $\mathcal{U}_{\mathcal{R}_q^2}$, the uniform distribution over $\mathcal{R}_q^2$.

In the above definition, we have left $\mathcal{S}$, the secret distribution, unspecified. Theoretical results [LPR10, LPR13] justify setting $\mathcal{S}$ to be the same as the error distribution χ. However, for improved efficiency, CKKS employs a ternary secret distribution, where the coefficients of $s \in \mathcal{R}$ are all in $\{-1, 0, 1\}$. We will utilize two different secret distributions. The first, $\mathcal{HWT}(h)$, for a positive integer h, is the uniform distribution over ternary secrets with Hamming weight h. That is, $\mathcal{HWT}(h)$ is the distribution over $\mathcal{R}$ obtained by sampling uniformly over the set of vectors in $\{-1, 0, 1\}^N$ with Hamming weight h and then viewing this vector as the coefficient vector of an element of $\mathcal{R}$. The second distribution is simply the distribution over ternary secrets where each coefficient is sampled independently from the same probability distribution over $\{-1, 0, 1\}$. The particular distribution we will use will have 0 occur with probability 1/2 and -1 and 1 occur each with probability 1/4.

Although ring-LWE with sparse secrets lacks a reduction to worst-case lattice problems, the security of ring-LWE with sparse secrets has been studied, and the parameters N, q, σ, h must be delicately chosen to obtain the desired security level λ. Please refer to [CP19, Alb17, APS15] for security analysis and appropriate parameter choices.

2.3 Message Space Encoding

The message space of CKKS will be ring polynomials $m(X) \in \mathcal{R}$ with sufficiently small norm. In order to support real/complex arithmetic, it is necessary to map such numbers into the polynomial ring $\mathcal{R}$. To begin, the reverse is possible using the canonical embedding $\sigma : \mathcal{R} \to \mathbb{C}^N$ defined by $\sigma(m(X)) = [m(\zeta), m(\zeta^3), m(\zeta^5), \ldots, m(\zeta^{2N-1})]$, where ζ is a primitive $2N$th root of unity. That is, σ is defined to be the resulting complex vector obtained by evaluating $m(X)$ at all the primitive $2N$th roots of unity. Since the primitive $2N$th roots of unity come in conjugate pairs, these evaluations also come in conjugate pairs since $m(\zeta^j) = \overline{m(\overline{\zeta^j})}$. This follows since

$$\begin{aligned} m(\zeta^j) &= \sum_{k=0}^{N-1} m_k \zeta^{kj} = \sum_{k=0}^{N-1} m_k (\overline{\zeta^{-kj}}) \\ &= \sum_{k=0}^{N-1} \overline{m_k (\zeta^{-kj})} = \overline{m(\overline{\zeta^j})}, \end{aligned}$$

where the equality in the second line follows from the fact that the m_k's are integers.

Thus, overloading notation, one can consider the restricted canonical embedding $\sigma : \mathcal{R} \to \mathbb{C}^{N/2}$ that only evaluates the ring polynomial at one of the primitive roots in each conjugate pair. For easy interaction with the Galois group, these primitive roots are chosen to be $\zeta, \zeta^5, \zeta^{5^2}, \ldots, \zeta^{5^{N/2-1}}$. Extending σ to also operate on polynomials with real coefficients, it is also possible to define an isometric ring isomorphism between $\mathcal{S} = \mathbb{R}[X]/\Phi_M(X)$ and

$\mathbb{C}^{N/2}$, where for an element $m(X) \in \mathcal{S}$, it has the canonical embedding norm $||m||_\infty^{\mathsf{can}} = ||\sigma(m)||_\infty$. This means that polynomial addition and multiplication in $\mathcal{S}$ is mapped to coordinate-wise addition and multiplication in $\mathbb{C}^{N/2}$ and, moreover, norms of elements are preserved. This is crucial because, as we will see, it enables packing $N/2$ complex numbers into a single CKKS ciphertext and performing the same operations on each entry in the complex vector independently. These different entries in the complex vector are referred to as the *plaintext slots.* Moreover, preserving norms is essential because CKKS is only for approximate arithmetic, and the fact that norms are preserved ensures that small errors in the coefficients of $m(X)$ will map to small errors in the plaintext slots.

Since the message space is the ring $\mathcal{R}$, it is necessary to first round a vector of complex numbers $\mathbf{z}$ in $\mathbb{C}^{N/2}$ to the image of $\sigma(\mathcal{R})$ before inverting. There are various methods to round to the image of $\sigma(R)$ with small rounding error, which we will not discuss here. In order to maintain a larger number of bits of precision, one can first multiply $\mathbf{z}$ by a scaling factor $\Delta \geq 1$ before rounding to the image of $\sigma(\mathcal{R})$ and mapping to $\mathcal{R}$. A plaintext ring polynomial $m(X)$ is decoded to $\mathbf{z}$ by simply applying σ and dividing by the scale factor Δ.

2.4 Approximate HE Construction

We are now ready to describe the construction of the CKKS homomorphic encryption scheme. The message space of the scheme is polynomials m in $\mathcal{R}$ with $||m||_\infty^{\mathsf{can}} < q/2$ for a base modulus q. As discussed above, one can map a vector in $\mathbb{C}^{N/2}$ of fixed precision into $\mathcal{R}$. A ciphertext ct encrypting a message $m \in \mathcal{R}$ is an element of $\mathcal{R}_{q_\ell}^2$ for some $\ell \in \{0, \ldots, L\}$. ℓ refers to the "level" of the ciphertext. In [CKKS17], $q_\ell = p^\ell * q$ for integers p and q. However,

q_ℓ can be set in other ways (such as via an RNS basis [CHK$^+$18b]). It can be convenient to have p and q be powers of 2 so that every q_ℓ is a power of 2. In this case, instead of referring to a ciphertext by its level, we can refer to it by the number of ciphertext modulus bits ($\log_2 q_\ell$). The advantage of this is that instead of reducing a ciphertext modulus by an entire level at a time, we can reduce the ciphertext modulus by a fixed number of bits. This allows for more fine-grained managing of ciphertexts, which can enable additional homomorphic computation prior to exhausting a ciphertext's computational budget.

The decryption structure is $\langle \mathsf{ct}, \mathsf{sk} \rangle \bmod q_\ell = m + e$ for some small error $e \in \mathcal{R}$. Observe that there is no way to remove e and some of the least significant bits of m are unrecoverable. A fresh ciphertext is generated at the highest level L. Homomorphic operations increase the magnitude of the error and the message and one must apply the rescaling procedure or modular reduction to bring a ciphertext to a lower level to continue homomorphic computation. Eventually, a ciphertext is at the lowest level (an element of $\mathcal{R}_q^2$), and no further operations can be performed. The scheme's algorithms are given below. Implicit in the description is the requirement to keep ciphertexts tagged with their current level ℓ.

- $\mathsf{KeyGen}(1^\lambda)$: On input the security parameter 1^λ, choose integers L, p, q, P, h, a power of two N, and a real number σ so that the ring-LWE assumption with parameters $N, P * q_L, \sigma, h$ holds with desired security level λ. Sample $s \leftarrow \mathcal{HWT}(h)$, $a \leftarrow \mathcal{R}_{q_L}$, and $e \leftarrow \chi$. Set $b = -a * s + e \bmod q_L$. Set pk as $(b, a) \in \mathcal{R}_{q_L}^2$ and set sk as $(1, s) \in \mathcal{R}^2$. Sample $a' \leftarrow \mathcal{R}_{P * q_L}$ and $e' \leftarrow \chi$. Set $b' = -a' * s + e' + P * s^2 \bmod P * q_L$. Set the evaluation key evk as $(b', a') \in \mathcal{R}_{P * q_L}^2$. Output $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$.

- $\mathsf{Enc}(\mathsf{pk}, m)$: On input the public key pk and a message $m \in \mathcal{R}$, sample $v \in \mathcal{R}$ by sampling each coefficient independently from the distribution over $\{-1, 0, 1\}$ that is 0 with probability $1/2$ and -1, 1 with probability $1/4$. Sample $e_0, e_1 \leftarrow \chi$. Output $v * \mathsf{pk} + (m + e_0, e_1) \bmod q_L$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: On input the secret key sk and a ciphertext ct, parse ct as (b, a) and output $b + a * s \bmod q_\ell$, where ℓ is the level of ct.

- $\mathsf{AddConst}(\mathsf{ct}, c)$: On input a ciphertext ct and a constant $c \in \mathcal{R}$, output $\mathsf{ct} + (c, 0) \bmod q_\ell$, where ℓ is the level of ct.

- $\mathsf{MultConst}(\mathsf{ct}, c)$: On input a ciphertext ct and a constant $c \in \mathcal{R}$, output $c * \mathsf{ct} \bmod q_\ell$, where ℓ is the level of ct.

- $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$: On input ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ at the same level ℓ, output $\mathsf{ct}_1 + \mathsf{ct}_2 \bmod q_\ell$.

- $\mathsf{Mult}(\mathsf{evk}, \mathsf{ct}_1, \mathsf{ct}_2)$: On input the evaluation key evk and ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ at the same level ℓ, parse ct_1 as (b_1, a_1) and ct_2 as (b_2, a_2). Set $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \bmod q_\ell$. Output $(d_0, d_1) + \lfloor P^{-1} * d_2 * \mathsf{evk} \rceil \bmod q_\ell$.

- $\mathsf{Rescale}(\mathsf{ct})$: On input a ciphertext ct at level ℓ, output $\lfloor \mathsf{ct}/p \rceil \bmod q_{\ell-1}$.

- $\mathsf{ModDown}(\mathsf{ct})$: On input a ciphertext ct at level ℓ, output $\mathsf{ct} \bmod q_{\ell-1}$.

Using CKKS

In order to perform computation using CKKS, it is necessary to keep ciphertexts tagged with additional information and to carefully manage ciphertexts. Ciphertexts should always be stored with their level ℓ, an upper bound M on the plaintext magnitude, an upper bound B on the error magnitude, and the scaling factor Δ for decoding. The upper bound M on the plaintext magnitude is used to ensure that no homomorphic computation is performed that could cause $||m||_\infty^{\mathsf{can}} > q_\ell/2$ for a ciphertext at level ℓ, which would cause overflow and prevent correct decryption. The upper bound B on the error magnitude is used to keep track of the number of accurate plaintext bits during approximate arithmetic.

The Add and Mult algorithms only work when both input ciphertexts are at the same level ℓ. Thus, if they are at different levels, it is necessary to bring the one at the higher level down to the lower level before utilizing the desired algorithm. Either the $\mathsf{Rescale}$ or the $\mathsf{ModDown}$ algorithm can be used to reduce the ciphertext level. $\mathsf{ModDown}$ simply reduces the ciphertext level without altering the plaintext or its associated error. $\mathsf{Rescale}$, on the other hand, not only reduces the ciphertext level, but also scales down the plaintext by approximately p and the associated error as well. Thus, when $\mathsf{Rescale}$ is used, the scale factor Δ should be adjusted accordingly. One must always take the scale factor of ciphertexts into account when performing homomorphic operations. A simple way to manage ciphertexts is to use a fixed scale factor Δ and always apply $\mathsf{Rescale}$ after Mult, which will bring the scale factor back to Δ from Δ^2. $\mathsf{ModDown}$ is utilized for other instances when ciphertext level reduction is required since it does not affect the scaling factor. In this manner, the number of levels consumed by a computation is its multiplicative depth.

Correctness and Compactness It follows immediately from the construction that for $m \in \mathcal{R}$, for $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$,

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m + (ev + e_0 + e_1 s) \bmod q_L.$$

Thus, for $||m||_\infty^{\mathsf{can}} \ll q_L$, decryption is approximately correct with error $e' = ev + e_0 + e_1 s$, which is small since v and s have small entries and Hamming weight with high probability.

Correctness of $\mathsf{AddConst}$ and $\mathsf{MultConst}$ are immediate by inspection. Correctness of Add follows from the fact that decryption is linear, so if $\langle \mathsf{ct}_1, \mathsf{sk} \rangle \bmod q_\ell \approx m_1$ and $\langle \mathsf{ct}_2, \mathsf{sk} \rangle \bmod q_\ell \approx m_2$, then

$$\langle \mathsf{ct}_1 + \mathsf{ct}_2, \mathsf{sk} \rangle \bmod q_\ell = \langle \mathsf{ct}_1, \mathsf{sk} \rangle + \langle \mathsf{ct}_2, \mathsf{sk} \rangle \bmod q_\ell \approx m_1 + m_2.$$

Correctness of Mult follows from the fact that the multiplied ciphertext decrypts to

$$d_0 + d_1 s + \lfloor P^{-1} * d_2 * (-a' * s + e' + P * s^2) \rceil + \lfloor P^{-1} * d_2 * a' * s \rceil \bmod q_\ell$$

$$\approx b_1 b_2 + (a_1 b_2 + a_2 b_1) s + a_1 a_2 s^2 + \lfloor P^{-1} * a_1 a_2 * e' \rceil \bmod q_\ell$$

$$\approx m_1 m_2$$

since

$$\begin{aligned} m_1 m_2 &\approx \langle \mathsf{ct}_1, \mathsf{sk} \rangle * \langle \mathsf{ct}_2, \mathsf{sk} \rangle \bmod q_\ell \\ &= (b_1 + a_1 s) * (b_2 + a_2 s) \bmod q_\ell \\ &= b_1 b_2 + (a_1 b_2 + a_2 b_1) s + a_1 a_2 s^2 \end{aligned}$$

and $\lfloor P^{-1} * a_1 a_2 * e' \rceil$ is small for sufficiently large P.

Correctness of Rescale and ModDown are immediate, noting that ModDown is only a valid operation if the underlying plaintext m satisfies $||m||_\infty^{\mathsf{can}} \ll q_{\ell-1}$. By appropriately applying these procedures, it is possible to evaluate bounded-depth arithmetic circuits, and correctness follows. Compactness also follows from noting that the evaluation procedures do not increase the size of the ciphertext, which is always two ring elements in $\mathcal{R}_{q_\ell}^2$, where ℓ is the level of the ciphertext. In fact, ciphertexts actually decrease in size as homomorphic computation is performed since reducing the level of a ciphertext reduces the ciphertext modulus.

Security

Assuming the hardness of ring-LWE with ternary secrets (Def. 2.2.1), the CKKS homomorphic encryption scheme satisfies the standard definition of $\mathsf{IND\text{-}CPA}$ security (Def. 2.1.3). We note that we can only prove security when the adversary is not given access to the evaluation key evk and will have to rely on a circular security assumption for security when the adversary has access to evk.

We will prove $\mathsf{IND\text{-}CPA}$ security (against an adversary not given access to evk) via a hybrid argument. Consider the following sequence of hybrids.

- $\mathsf{Hyb}_1(1^\lambda, b)$: This hybrid is the same as $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, b)$ except that pk is sampled uniformly at random from $\mathcal{R}_{q_L}^2$.

- $\mathsf{Hyb}_2(1^\lambda, b)$: This hybrid is the same as Hyb_1 except that $\mathsf{Enc}(\mathsf{pk}, m_b)$ is computed by sampling two uniformly random elements u_0, u_1 from $\mathcal{R}_{q_L}^2$ and outputting $\mathsf{ct} = (u_0 + m_b, u_1) \bmod q_L$.

- $\mathsf{Hyb}_3(1^\lambda, b)$: This hybrid is the same as Hyb_2 except that $\mathsf{Enc}(\mathsf{pk}, m_b)$ is computed by sampling two uniformly random elements u_0, u_1 from $\mathcal{R}_{q_L}^2$ and outputting $\mathsf{ct} = (u_0, u_1)$.

Lemma 2.4.1. $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, 0) \approx_c \mathsf{Hyb}_1(1^\lambda, b)$.

Proof. This follows immediately from the ring-LWE assumption with sparse secrets where s is the ring-LWE secret.

Lemma 2.4.2. $\mathsf{Hyb}_1(1^\lambda, b) \approx_c \mathsf{Hyb}_2(1^\lambda, b)$.

Proof. This follows immediately from the ring-LWE assumption with sparse secrets where v is the ring-LWE secret.

Lemma 2.4.3. $\mathsf{Hyb}_2(1^\lambda, b) \approx_s \mathsf{Hyb}_3(1^\lambda, b)$.

Proof. This is immediate since the distribution of $u_0 + m_b$ is uniform.

Observe that $\mathsf{Hyb}_3(1^\lambda, b)$ is independent of b, so it follows that $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, 0) \approx_c \mathsf{Expt}_{\mathcal{A}}^{\mathsf{IND-CPA}}(1^\lambda, 1)$ as desired.

Security When Publishing Decryption Results

Since CKKS is for approximate arithmetic, the notion of IND-CPA security may be insufficient for various applications. This is because if the approximate computation results are published, they may reveal information about the underlying ring-LWE error, compromising security. In fact, [LM21] recently showed attacks against CKKS if the decryption results are published. This can be addressed by modifying the decryption algorithm to, say, add additional noise or round the decrypted value. We refer the interested reader to [LM21] for more details.

Error Growth

As previously discussed, CKKS is only for approximate arithmetic and, therefore, there are associated errors with the various homomorphic operations. It is crucial to have an understanding of the error growth of various operations, so that the error bounds associated with ciphertexts can be updated appropriately and the precision of a decrypted value is known. We will cite the error analysis given in [CKKS17] and refer the interested reader there for further details. The error growth can be bounded based on the following constants that we will subsequently define: B_{clean}, B_{scale}, and $B_{mult}(\ell)$. A fresh ciphertext has an error upper bound of B_{clean}. $\mathsf{AddConst}$ does not affect the error bound of a ciphertext, while $\mathsf{MultConst}$ applied to a ciphertext with error bound B outputs a ciphertext with error bound $||c||_\infty^{\mathsf{can}} * B$, where c is the constant multiplied. Add simply adds the error bounds B_1 and B_2 of ct_1 and ct_2, respectively, to give the error bound $B_1 + B_2$ of the added ciphertext. Mult takes ciphertexts ct_1 and ct_2 at level ℓ, with message and error bounds M_1, B_1 and M_2, B_2, respectively, and outputs a ciphertext with error bound $M_1B_2 + M_2B_1 + B_1B_2 + B_{\mathsf{mult}}(\ell)$. $\mathsf{Rescale}$ takes a ciphertext with error bound B and outputs a ciphertext with error bound $B/p + B_{\mathsf{scale}}$. $\mathsf{ModDown}$ does not affect the error bound B of a ciphertext.

The constants B_{clean}, B_{scale}, and $B_{mult}(\ell)$ are given as follows in Lemmas 1–3 in [CKKS17] and are upper bounds with high probability:

$$B_{\mathsf{clean}} = 8\sqrt{2}\sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}$$

$$B_{\mathsf{scale}} = \sqrt{N/3} * (3 + 8\sqrt{h})$$

$$B_{\mathsf{mult}}(\ell) = P^{-1} * q_\ell * (8\sigma N/\sqrt{3}) + B_{\mathsf{scale}}$$

Rotating and Conjugating Plaintext Slots

The procedures described above operate independently on each of the plaintext slots. This immediately gives parallelism since up to $N/2$ complex numbers can be encoded in a single ring polynomial $m \in \mathcal{R}$. However, using the Galois group action, it is also possible to rotate the plaintext slots to enable interaction between them and conjugate the plaintext slots. Recall that the Galois group $\mathcal{G}$ of $\mathbb{Q}[\zeta]/\mathbb{Q}$ is given by the automorphisms f_k that map ζ to ζ^k for odd integers k between 1 and $2N$ and is isomorphic to $\mathbb{Z}_{2N}^*$. $\mathcal{G}$ is generated by the automorphisms f_5 and $f_{-1} = f_{2N-1}$. Recall that the plaintext encoding is defined by the mapping σ, where $\sigma(m) = [m(\zeta), m(\zeta^5), m(\zeta^{5^2}), \ldots, m(\zeta^{5^{N/2-1}})]$. Applying the automorphism f_5 to m gives $m'(X) = m(X^5)$ such that

$$\sigma(m') = [m(\zeta^5), m(\zeta^{5^2}), \ldots, m(\zeta^{5^{N/2-1}}), m(\zeta)],$$

a rotation of the plaintext slots of m. Similarly, applying the automorphism f_{-1} to m gives $m'(X) = m(X^{-1})$ such that

$$\begin{aligned}\sigma(m') &= [m(\zeta^{-1}), m(\zeta^{-5}), m(\zeta^{-5^2}), \ldots, m(\zeta^{-5^{N/2-1}})]\\ &= [m(\overline{\zeta}), m(\overline{\zeta^5}), m(\overline{\zeta^{5^2}}), \ldots, m(\overline{\zeta^{5^{N/2-1}}})]\\ &= [\overline{m(\zeta)}, \overline{m(\zeta^5)}, \overline{m(\zeta^{5^2})}, \ldots, \overline{m(\zeta^{5^{N/2-1}})}],\end{aligned}$$

a conjugation of the plaintext slots of m.

It is also possible to apply the automorphisms to ciphertexts in order to rotate/conjugate the encrypted plaintext slots. For example, if we have a ciphertext $\mathsf{ct} = (b, a)$ that encrypts $m \in \mathcal{R}$, then applying an automorphism f to ct gives the ciphertext $\mathsf{ct}' = (f(b), f(a))$. Since f is an automorphism, it follows that ct' decrypts to $f(m)$ under the secret key $\mathsf{sk}' = (1, f(s))$. Therefore, it is necessary to apply the key-switching procedure to ct' to transform it into a ciphertext that decrypts to $f(m)$ under the original secret key $\mathsf{sk} = (1, s)$. This can be done by having KeyGen additionally publish an automorphism key autk for f that is analogous to evk except it encrypts $f(s)$ instead of s^2. Key-switching, as is implicit in the description of Mult, can then be applied to ct' using autk. One can simply give out automorphism keys for f_5 and f_{-1} to enable performing any automorphism in $\mathcal{G}$, but, in practice, one can give out additional automorphism keys to allow the automorphisms to be performed more efficiently at the cost of increasing the size of the extra key material required.

2.5 Bootstrapping of Approximate HE

[CHK+18a] introduced the first bootstrapping procedure for the CKKS-HE scheme. Subsequent works [CCS19, HHC19, HK20, BMTPH21] improved various aspects of bootstrapping, but the overall procedure remains the same. The goal is to take a ciphertext at the lowest level and bring it up to a higher level so that homomorphic computation can continue. Thus, given a ciphertext ct at the lowest level, we want to obtain another ciphertext ct' such that

$$\langle \mathsf{ct}, \mathsf{sk} \rangle \bmod q \approx \langle \mathsf{ct}', \mathsf{sk} \rangle \bmod q_\ell$$

for some $\ell > 1$. For simplicity in the following, we will include the starting decryption error in the message m. That is, we will assume that $\langle \mathsf{ct}, \mathsf{sk} \rangle \bmod q = m$.

Bootstrapping is done via the following sequence of steps:

1. **Modulus Raising:** By simply considering ct as a ciphertext at the highest level, it follows that $\langle \mathsf{ct}, \mathsf{sk} \rangle \bmod q_L = qI + m$ for some $I \in \mathcal{R}$.

2. **Coefficients to Slots:** We need to perform the modular reduction on the polynomial coefficients of $t = qI + m$. However, recall that homomorphic computations evaluate coordinate-wise on the plaintext "slots," not the polynomial coefficients. Thus, we need to transform our ciphertext so that the polynomial coefficients are in the "slots." This can be done by evaluating a linear transformation homomorphically. We elaborate below.

3. **Compute the Mod Function:** We need a procedure to compute/approximate the mod function homomorphically. This is a significant challenge since we can only compute arithmetic operations homomorphically. Improving this step is the focus of this dissertation, and we describe our approach via a sine series approximation of the mod function in Chapter 3.

4. **Slots to Coefficients:** Finally, we need to undo the coefficients to slots step. This can be done by homomorphically evaluating the inverse of the previous linear transform.

Observe that if we can approximate the mod function, then the above procedure will give us a ct' at some higher level ℓ that decrypts to $m + e$ for some small error e. Since we are dealing with approximate arithmetic, this error from bootstrapping can be absorbed into the other errors that occur during approximate arithmetic and homomorphic evaluation. We can upper bound $|I| < K$ for some integer K so that we only need to approximate the mod function on the interval $[-Kq - m, Kq + m]$, where we have overloaded notation to make m an upper bound on the size of the message. The integer K is determined based on the Hamming weight h of the secret s. Under the assumption that the coefficients of a ciphertext are distributed uniformly over $\mathbb{Z}_q$, we can obtain an $O(\sqrt{h})$ bound for K.

Coefficients to Slots and Slots to Coefficients

Let $\zeta_j = \zeta^{5^j}$ for $0 \leq j < N/2$. Recall that a ring polynomial $m(X) \in \mathcal{R}$ encodes the complex vector $\mathbf{z}^T = [m(\zeta), m(\zeta^5), m(\zeta^{5^2}), \ldots, m(\zeta^{5^{N/2-1}})]$. We can express this relationship via the following linear transform.

$$\begin{bmatrix} 1 & \zeta_0 & \zeta_0^2 & \cdots & \zeta_0^{N-1} \\ 1 & \zeta_1 & \zeta_1^2 & \cdots & \zeta_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \zeta_{N/2-1}^2 & \cdots & \zeta_{N/2-1}^{N-1} \end{bmatrix} \cdot \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{N-1} \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{N/2-1} \end{bmatrix}$$

Defining the leftmost matrix as U and the coefficient vector of $m(X)$ as $\mathbf{m}$, it follows that $U \cdot \mathbf{m} = \mathbf{z}$ and $\overline{U} \cdot \mathbf{m} = \overline{\mathbf{z}}$. Define $W = \begin{bmatrix} U \\ \overline{U} \end{bmatrix}$. It follows that $W^{-1} = \frac{1}{N}\overline{W}^T$ since

$X^N - 1 = (X-1)(1 + X + X^2 + \ldots + X^{N-1})$, which implies that $1 + X + X^2 + \ldots + X^{N-1}$ evaluates to 0 at every Nth root of unity except for 1 where it evaluates to N. Thus, $\mathbf{m} = W^{-1} \cdot \begin{bmatrix} \mathbf{z} \\ \overline{\mathbf{z}} \end{bmatrix}$. Thus, if we want to take a ciphertext encrypting $m(X)$, which encodes the complex vector $\mathbf{z}$, and obtain ciphertexts encrypting a message polynomials that encode the complex vectors $\mathbf{m}_0 = [m_0, m_1, \ldots, m_{N/2-1}]^T$ and $\mathbf{m}_1 = [m_{N/2}, m_{N/2+1}, \ldots, m_{N-1}]^T$, it suffices to homomorphically evaluate the linear transform W^{-1} on ciphertexts encrypting the complex vectors $\mathbf{z}$ and $\overline{\mathbf{z}}$. Observe that we require two ciphertexts to hold all the coefficients of $m(X)$ since there are N coefficients, but only $N/2$ plaintext slots. We can easily obtain a ciphertext encrypting $\overline{\mathbf{z}}$ from one encrypting $\mathbf{z}$ by applying the automorphism for conjugation. From this, we can obtain ciphertexts encrypting $\mathbf{m}_0$ and $\mathbf{m}_1$ by performing homomorphic matrix multiplications and additions. Observe that matrix multiplication can be evaluated homomorphically using a combination of homomorphic multiplications and rotations. For the inverse operation (the Slots to Coefficients step), observe that defining $U = \begin{bmatrix} U_0 & U_1 \end{bmatrix}$ for square matrices U_0, U_1 gives $\mathbf{z} = U_0 \cdot \mathbf{m}_0 + U_1 \cdot \mathbf{m}_1$. Thus, we can obtain a ciphertext encrypting $\mathbf{z}$ from ones encrypting $\mathbf{m}_0$ and $\mathbf{m}_1$ by evaluating the previous equation homomorphically.

It is often the case that we do not need to utilize all $N/2$ plaintext slots. If, instead, we only use n slots, it is possible to encode these n slots in a subring of $\mathcal{R}$ and adapt the above approach so that the number of homomorphic operations required depends on n instead of N. Please refer to [CHK$^+$18a] for additional details. Furthermore, while the above illustrates the method of performing the Coefficients to Slots and Slots to Coefficients steps presented in [CHK$^+$18a], there have been subsequent works [CCS19, BMTPH21] that have improved the efficiency of these steps.

Chapter 3

Sine Series Approximation of the Mod Function

In this chapter, we will show the following theorem and corollaries, giving a sine series approximation to the mod function in small intervals around the modulus that can be used for CKKS-HE bootstrapping.

Theorem 3.0.1. *For every $n \geq 1$, there exists a sequence of rational numbers $\beta_1, ...\beta_n$ such that for every ϵ, $0 < \epsilon < 2/\sqrt{n}$, for every $|x| < \epsilon$,*

$$\left| x - \sum_{k=1}^{n} \beta_k \sin(kx) \right| < e^2 * (n+1) * (\epsilon/2)^{2n+1}.$$

Using the periodicity of the sine function, we immediately arrive at the following corollary.

Corollary 3.0.1. *For every $n \geq 1$, there exists a sequence of rational numbers $\beta_1, ...\beta_n$ such* *that for every* ϵ, $0 < \epsilon < 2/\sqrt{n}$, *for every integer* m, *for every* x *such that* $|x - 2m\pi| < \epsilon$,

$$\left|(x \; mod \, 2\pi) - \sum_{k=1}^{n} \beta_k \sin(kx)\right| < e^2 * (n+1) * (\epsilon/2)^{2n+1}.$$

A further simple manipulation leads to the following scaled version of the corollary.

Corollary 3.0.2. *For every* $n \geq 1$, *there exists a sequence of rational numbers* $\beta_1, ...\beta_n$ *such that for every* ϵ, $0 < \epsilon < \frac{1}{\pi\sqrt{n}}$, *for every integer* $q \geq 1$, *for every integer* m, *for every* x *such that* $|x - m * q| < \epsilon * q$,

$$\left|(x \; mod \, q) - \frac{q}{2\pi} * \sum_{k=1}^{n} \beta_k \sin(2\pi k * x/q)\right| < \frac{e^2 * q}{2\pi} * (n+1) * (\epsilon\pi)^{2n+1}.$$

3.1 Determining the β_i's

To prove Theorem 3.0.1, for each n, we will determine the rational numbers $\{\beta_i\}_{i\in[n]}$. In particular, these are *not* the same as the Fourier coefficients of the sawtooth function, as we are focused on x that is potentially much smaller than the period of the sawtooth function. Recall that we wish to determine $\{\beta_i\}_{i\in[n]}$ such that the resulting sine series has a Taylor series expansion of the form $x + x^{2n+1}p(x)$ for some polynomial $p(x)$. In particular, there are no terms of degree $< 2n+1$ (except for x). These constraints give a system of equations that can be solved to determine the β_i's.

We begin by formalizing this intuition. For every $n > 0$, for every sequence of n distinct integers $\mathbf{a} = (a_1, ..., a_n)$, let $V^{(n)}(\mathbf{a})$ denote the Vandermonde matrix of $\mathbf{a}$, i.e. it is the $n \times n$ matrix with the (i, j)-th element a_i^{j-1} (for $i, j \in [1..n]$). Define $S^{(n)}(\mathbf{a})$ to be the $n \times n$ matrix with the (i,j)-th element a_i^{2j-1}, i.e. each row is the odd powers of the elements of $\mathbf{a}$. Note that the first column of this matrix is just $\mathbf{a}$. Also, define a related matrix $\hat{S}^{(n)}(\mathbf{a})$ to be the $n \times n$ matrix which is same as $S^{(n)}(\mathbf{a})$ except that the first column (i.e. $\mathbf{a}$) is replaced by $(2n+1)$-th powers of $\mathbf{a}$. In other words, the $(i,1)$-th element of this matrix is $a_i^{(2n+1)}$.

Let $\vec{\beta} = (\beta_1, \beta_2, \ldots, \beta_n)$ be an n-vector of rational numbers. For the sine series approximation, we would like to determine $\vec{\beta}$ so that the transpose of the matrix $S^{(n)}(\mathbf{a})$ multiplied by $\vec{\beta}$ is a vector with all entries zero except the first, which is one. Since β_i refers to the coefficient of the $\sin(a_i x)$ term in the sine series, the above requirement ensures that when we Taylor expand each sine term in the sine series about the origin (or a multiple of 2π) and sum the terms, the resulting polynomial will be $x + x^{2n+1}p(x)$ for some polynomial $p(x)$. Thus, the $x^3, x^5, \ldots, x^{2n-1}$ terms in the Taylor series expansions of the $\sin(ix)$'s cancel out. We note that since our sine series will include $\sin x, \sin 2x, \sin 3x, \ldots$ terms, we will later instantiate $\mathbf{a}$ with $(1, 2, \ldots, n)$. The required condition is drawn below.

$$\begin{pmatrix} a_1 & a_2 & \ldots & a_n \\ a_1^3 & a_2^3 & \ldots & a_n^3 \\ & & \vdots & \\ a_1^{2n-1} & a_2^{2n-1} & \ldots & a_n^{2n-1} \end{pmatrix} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{3.1}$$

Let d_i denote the $(i,1)$-th minor of $S^{(n)}(\mathbf{a})$. In other words, the list $\{d_i\}_i$ is the list of minors of the first column of $S^{(n)}(\mathbf{a})$.

Lemma 3.1.1.

$$\beta_i = (-1)^{i+1} * \frac{d_i}{\det(S^{(n)}(\mathbf{a}))}.$$

Proof. From the above equation, $\vec{\beta}$ is just the first column of the inverse of $(S^{(n)}(\mathbf{a}))^T$. Note that the $(i,1)$-th element of the inverse of the transpose of $S^{(n)}(\mathbf{a})$ is $(-1)^{i+1} * d_i$ divided by the determinant of $S^{(n)}(\mathbf{a})$. □

We now give an explicit formula for the determinant of $S^{(n)}(\mathbf{a})$. We will also give an explicit formula for the determinant of $\hat{S}^{(n)}(\mathbf{a})$, which will be of use later. We will use the well-known fact that the determinant of the Vandermonde matrix is given by the following formula.

$$\det(V^{(n)}(\mathbf{a})) = \prod_{i=1}^{n} \prod_{1 \le j < i} (a_i - a_j).$$

Lemma 3.1.2. *The determinant of the matrix $S^{(n)}(\mathbf{a})$ is*

$$\left(\prod_{i=1}^{n} a_i\right) * \prod_{i=1}^{n} \prod_{1 \le j < i} (a_i^2 - a_j^2).$$

The determinant of the matrix $\hat{S}^{(n)}(\mathbf{a})$ is

$$(-1)^{n-1} * \det(S^{(n)}(\mathbf{a})) * \prod_{i=1}^{n} a_i^2.$$

Proof. We will first focus on the matrix $S^{(n)}(\mathbf{a})$. For computing the determinant, for each row i, we get a contribution of a factor a_i towards the determinant, and the remaining matrix is then just a Vandermonde matrix with all powers of a_i^2. Thus,

$$\det(S^{(n)}(\mathbf{a})) = \left(\prod_{i=1}^{n} a_i\right) * \det(V^{(n)}(\mathbf{a}')),$$

where $\mathbf{a}' = (a_1^2, \ldots, a_n^2)$. The result then follows from the well-known determinant of Vandermonde matrices.

As for the claim for the matrix $\hat{S}^{(n)}(\mathbf{a})$, first consider a modified matrix that is obtained by moving the first column to the last. Since this can be accomplished by $(n-1)$ column exchanges, the determinant of the modified matrix is $(-1)^{n-1}$ times the determinant of $\hat{S}^{(n)}(\mathbf{a})$. Furthermore, the determinant of the modified matrix is easily related to determinant of $S^{(n)}(\mathbf{a})$ by noting that i-th row in the modified matrix is a_i^2 times the i-th row in $S^{(n)}(\mathbf{a})$.

□

We observe from the formula for the determinant of $S^{(n)}(\mathbf{a})$ that if the sequence of integers $\mathbf{a}$ are in increasing order and lower bounded by one, then the determinant of $S^{(n)}(\mathbf{a})$ is positive. We now show the following lemma, characterizing the β_i's.

Lemma 3.1.3. *For the matrix $S^{(n)}(\mathbf{a})$ with $\mathbf{a}$ set to the sequence of integers from one to n,*

$$\beta_1 = \frac{2n}{n+1} < 2$$

and, for $i \geq 2$

$$|\beta_i| \ < \ 1.$$

Moreover, the β_i's alternate in sign and decrease in magnitude as i increases. That is,

$$|\beta_{i+1}| < |\beta_i|$$

for all $i \in [n]$, $\beta_{2j+1} > 0$, and $\beta_{2j} < 0$.

Proof. We will show this using the formula for β_i from Lemma 3.1.1. By definition,

$$d_i = \det \begin{pmatrix} a_1^3 & a_1^5 & \dots & a_1^{2n-1} \\ a_2^3 & a_2^5 & \dots & a_2^{2n-1} \\ & & \vdots & \\ a_{i-1}^3 & a_{i-1}^5 & \dots & a_{i-1}^{2n-1} \\ a_{i+1}^3 & a_{i+1}^5 & \dots & a_{i+1}^{2n-1} \\ & & \vdots & \\ a_n^3 & a_n^5 & \dots & a_n^{2n-1} \end{pmatrix}$$

Thus,

$$d_i = \left(\prod_{j=1, j\neq i}^{n} a_j^2 \right) * \det(S^{(n-1)}(\mathbf{a}')),$$

where $\mathbf{a}'$ is $\mathbf{a}$ with a_i removed. Thus,

$$\beta_i = (-1)^{i+1} * \frac{\left(\prod_{j=1, j\neq i}^{n} a_j^2 \right)}{a_i * \left(\prod_{j=1}^{i-1} (a_i^2 - a_j^2) \right) * \left(\prod_{j=i+1}^{n} (a_j^2 - a_i^2) \right)}.$$

We observe that every term in the above expression is positive except for $(-1)^{i+1}$ and, thus, the β_i's alternate sign with $\beta_{2j+1} > 0$ and $\beta_{2j} < 0$. It follows that

$$\beta_1 = \frac{2(n!)^2}{(n+1)!(n-1)!} = \frac{2n}{n+1} < 2.$$

Moreover, for $i \geq 2$,

$$|\beta_i| = \frac{1}{i} * \frac{2(n!)^2}{(2n)!} * \binom{2n}{n+i}$$

Observe that $|\beta_{i+1}| < |\beta_i|$. Moreover, since $\binom{2n}{n+i} < \binom{2n}{n}$ for $i \geq 2$, it follows that

$$|\beta_i| < \frac{2}{i} \leq 1$$

for $i \geq 2$. □

3.2 Bounding the Error: A First Attempt

Having characterized the β_i's, we now turn our focus to bounding the error between $f(x) = \sum_{k=1}^{n} \beta_k \sin(kx)$ and x for $|x| < \epsilon$. We note that $f(x)$ is an analytic function since it is the sum of analytic functions and, therefore, its Taylor series converges to $f(x)$. Thus, taking the Taylor series expansion of $f(x)$ around 0,

$$f(x) = x + \sum_{m=2n+1}^{\infty} \frac{f^{(m)}(0)}{m!} x^m.$$

We can bound $|x - f(x)|$ for $|x| < \epsilon$ using the Lagrange remainder term of the $2n$-th Taylor polynomial of $f(x)$. Thus,

$$|x - f(x)| = \left| \frac{f^{(2n+1)}(\xi)}{(2n+1)!} x^{2n+1} \right|$$

for some real number ξ between 0 and x. We have that

$$f^{(2n+1)}(x) = \pm \sum_{k=1}^{n} \beta_k k^{2n+1} \cos(kx).$$

Upper bounding $f^{(2n+1)}(\xi)$ gives

$$|x - f(x)| < \sum_{k=1}^{n} |\beta_k| k^{2n+1} \frac{|x^{2n+1}|}{(2n+1)!}.$$

By Lemma 3.1.3, $\beta_k < 2/k$, which gives

$$|x - f(x)| < |x^{2n+1}| * \frac{2}{(2n+1)!} * \sum_{k=1}^{n} k^{2n}.$$

This then gives an upper bound of $\epsilon^{2n+1} * \frac{2*n*n^{2n}}{(2n+1)!}$, and no better than $\epsilon^{2n+1} * \frac{2*n^{2n}}{(2n+1)!} \approx (\epsilon/2)^{2n+1} * e^{2n}/(\sqrt{\pi(n+1)} * n)$ However, we will now show that a more sophisticated, yet elementary, approach that improves upon this bound by approximately a factor of e^{2n}, essentially giving us an upper bound of $(\epsilon/2)^{2n+1}$.

3.3 A Better Bound via the Alternating Series Test

To obtain a better error bound, we will show that the Taylor series expansion of our sine series satisfies Leibniz's alternating series test. This will enable us to bound the error of the sine series $f(x)$ from the mod function by the $(2n+1)-$th term in the Taylor series expansion (the first nonzero term after x). We can write the Taylor series expansion of $f(x)$

as $x - \sum_{m=n+1}^{\infty}(-1)^m * b_m$, where

$$b_m = \sum_{j=1}^{n} \beta_j * \frac{(jx)^{2m-1}}{(2m-1)!}. \tag{3.2}$$

To bound the error, we will show, for any x in the domain of approximation, that the series $\sum_{m=n+1}^{\infty}(-1)^m * b_m$ satisfies the alternating series test. The alternating series test requires that the b_m satisfy the following three conditions.

1. $\lim_{m\to\infty} b_m = 0$

2. All b_m are positive (or all b_m are negative)

3. $|b_m| \geq |b_{m+1}|$ for all natural numbers $m \geq n+1$.

Theorem 3.3.1. *Alternating Series Test [Leibniz]. If the series above satisfies the alternating series test then $\sum_{m=n+1}^{\infty}(-1)^m * b_m$ converges. Moreover, for all $k \geq 0$,*

$$\left| \sum_{m=n+1}^{\infty} (-1)^m * b_m - \sum_{m=n+1}^{n+1+k-1} (-1)^m * b_m \right| \leq |b_{n+1+k}|.$$

We will show the following lemma.

Lemma 3.3.1. *(**Main Lemma**) For every $|x| < 2/\sqrt{n}$, the above series given by b_m satisfies the Leibniz alternating series test.*

A Naive Proof Attempt

We briefly explain why the following naive approach to proving this lemma fails. For simplicity, assume that n is odd, so that β_n is positive and β_{n-1} is negative by Lemma 3.1.3.

Then, the naive approach would be to prove that

$$\beta_n * \frac{(n*x)^{2m-1}}{(2m-1)!} + \beta_{n-1} * \frac{((n-1)*x)^{2m-1}}{(2m-1)!}$$

(and similarly paired other terms) decreases as m increases, starting from $m = n+1$. Since powers of $n*x$ are larger than powers of $(n-1)*x$, this would eventually be true for some $m > n+1$. However, since $|\beta_n| < |\beta_{n-1}|$ and β_{n-1} is negative (see Lemma 3.1.3), this is not necessarily true at $m = n+1$. In fact, calculations show that this indeed fails for a few terms beyond $m = n+1$. Thus, a more advanced approach is required to prove that the Leibniz test holds starting at $m = n+1$. We will show that the test holds for $|x| < 2/\sqrt{n}$.

Preparing for the Proof

We prove Lemma 3.3.1 in the next subsection, but first we show several additional lemmas which will assist us in the proof of Lemma 3.3.1.

Define $V^{(n,k)}(\mathbf{a})$ to be an $n \times n$ matrix, which is same as the Vandermonde matrix $V^{(n)}(\mathbf{a})$ except the last column is replaced by the $(n-1+k)$-th powers (instead of the $(n-1)$-th powers).

Let $h_k(\mathbf{a})$ be the *complete homogeneous symmetric polynomial* of degree k in $\mathbf{a}$ given by

$$h_k(\mathbf{a}) = \sum_{1 \le i_1 \le \ldots \le i_k \le n} a_{i_1} * \cdots * a_{i_k}.$$

The base polynomial $h_0(\mathbf{a})$ is taken to be one. Note that the polynomials $h_k(\mathbf{a})$ differ from the elementary symmetric polynomials $e_k(\mathbf{a})$, since in the latter the summation is taken over

$1 \leq i_1 < ... < i_k \leq n$. The following lemma is a consequence of the well-known *generating series* of the complete homogeneous symmetric polynomials, but we give a simple proof for completeness.

Lemma 3.3.2. *For any $k \geq 0$, any $\mathbf{a}$ of length $n > 0$, and an independent formal variable t,*

$$\sum_{j=0}^{k} h_j(\mathbf{a})t^j \ = \ \prod_{i=1}^{n}\sum_{j=0}^{k}(ta_i)^j \ \mathit{mod} \ \ t^{k+1}.$$

Proof. We prove this lemma by induction over n. The base case for $n = 1$ follows as $h_j(a) = a^j$ for every j in $[0..k]$. Suppose the lemma holds for $n-1$. Then, let $\mathbf{a}'$ be the truncation of $\mathbf{a}$ to its first $n-1$ components. We have, modulo t^{k+1},

$$\begin{aligned}
\prod_{i=1}^{n}\sum_{j=0}^{k}(ta_i)^j &= \sum_{z=0}^{k}(ta_n)^z \ * \ \prod_{i=1}^{n-1}\sum_{j=0}^{k}(ta_i)^j \\
&= \sum_{z=0}^{k} t^z a_n^z \ * \ \sum_{j=0}^{k} h_j(\mathbf{a}')t^j \\
&= \sum_{j=0}^{k}\sum_{z=0}^{k} a_n^z * h_j(\mathbf{a}')t^{j+z} \\
&= \sum_{z=0}^{k}\sum_{j=0}^{k} a_n^z * h_j(\mathbf{a}')t^{j+z} \\
&= \sum_{z=0}^{k}\sum_{j=0}^{k-z} a_n^z * h_j(\mathbf{a}')t^{j+z} \\
&= \sum_{z=0}^{k}\sum_{j'=z}^{k} a_n^z * h_{j'-z}(\mathbf{a}')t^{j'} \\
&= \sum_{z=0}^{k}\sum_{k\geq j';\, j'\geq z} a_n^z * h_{j'-z}(\mathbf{a}')t^{j'} \\
&= \sum_{z\leq k;\, j'\leq k;\, z\geq 0;\, z\leq j'} a_n^z * h_{j'-z}(\mathbf{a}')t^{j'} \\
&= \sum_{j'=0}^{k}\sum_{z=0}^{j'} a_n^z * h_{j'-z}(\mathbf{a}')t^{j'} \\
&= \sum_{j'=0}^{k} h_{j'}(\mathbf{a})t^{j'}
\end{aligned}$$

□

Lemma 3.3.3. *For $k \geq 1$, the determinant of the matrix $V^{(n,k)}(\mathbf{a})$ is*

$$\det(V^{(n)}(\mathbf{a})) \, * \, h_k(\mathbf{a})$$

Proof. Fix any $k \geq 1$. Consider an $n \times n$ matrix M which is same as $V^{(n,k)}(\mathbf{a})$ except that the last row is powers of an indeterminate x. In other words the last row is $(x^0, x^1, ..., x^{n-2}, x^{n-1+k})$. Let $\mathbf{a}'$ stand for the $(n-1)$ length truncation of $\mathbf{a}$. Treating the elements of $\mathbf{a}'$ as scalars, the determinant of the matrix M is a polynomial in x of degree $n-1+k$. Call this polynomial $f(x)$. Since the determinant of a matrix with two equal (or even scaled by a constant) rows is zero, the polynomial $f(x)$ has roots $\mathbf{a}'$. Thus,

$$f(x) = g(x) * \prod_{i=1}^{n-1} (x - a_i), \tag{3.3}$$

where $g(x)$ is a polynomial (to be determined) of degree k . However, $f(x)$, the degree $n-1+k$ polynomial, has zero coefficients for all monomials x^j with j in $[n-1..n-1+k-1]$. If we introduce a new formal variable $t = 1/x$, then the above equation (3.3) can be written as

$$\tilde{f}(t) = \tilde{g}(t) * \prod_{i=1}^{n-1} (1 - ta_i). \tag{3.4}$$

where $\tilde{f}$ (resp. $\tilde{g}$) is the polynomial f (resp. g) with coefficients reversed. Note, all the zero coefficients of $f(x)$ described above imply that coefficient of monomial t^j in $\tilde{f}(t)$ is zero for every j in $[1..k]$, and the constant term in $\tilde{f}(t)$ is f_{n-1+k}, where f_{n-1+k} denotes the coefficient of x^{n-1+k} in $f(x)$. Thus, $\tilde{f}(t) = f_{n-1+k} \bmod t^{k+1}$. Considering equation (3.4) modulo t^{k+1}, we get

$$f_{n-1+k} \ * \ \prod_{i=1}^{n-1}(1-ta_i)^{-1} = \tilde{g}(t) \bmod \ t^{k+1}. \tag{3.5}$$

The above equation is well-formed as inverse of $(1-ta_i)$ modulo t^{k+1} is well-defined. Indeed, it is easy to check that $(1-ta_i) * \sum_{j=0}^{k}(ta_i)^j$ is 1 mod t^{k+1}. Hence, we also get,

$$f_{n-1+k} \ * \ \prod_{i=1}^{n-1}\sum_{j=0}^{k}(ta_i)^j = \tilde{g}(t) \bmod \ t^{k+1}. \tag{3.6}$$

Since $g(x)$ is of degree k, $\tilde{g}(t)$ has degree at most k as well. Denote by $\tilde{g}_j$ the coefficient of t^j in $\tilde{g}_j$, which is same as g_{k-j}. Then, by comparing coefficients of t^j on both sides, by Lemma 3.3.2 we get that for each $j \in [0..k]$,

$$g_{k-j} = \tilde{g}_j = f_{n-1+k} \ * \ h_j(\mathbf{a}').$$

Thus, having determined $g(x)$, we also have $f(x)$ by (3.3). Letting $x = a_n$, then we get

$$\begin{aligned}
\det\ (V^{(n,k)}(\mathbf{a})) &= f(a_n) \\
&= \prod_{i=1}^{n-1}(a_n - a_i) \ * \ g(a_n) \\
&= \prod_{i=1}^{n-1}(a_n - a_i) \ * \ f_{n-1+k} \ * \ \sum_{j=0}^{k} a_n^{k-j} h_j(\mathbf{a}') \\
&= \prod_{i=1}^{n-1}(a_n - a_i) \ * \ f_{n-1+k} \ * \ h_k(\mathbf{a}) \\
&= \det(V^{(n)}(\mathbf{a})) * h_k(\mathbf{a}),
\end{aligned}$$

where the last equality follows by noting that the top coefficient of $f(x)$, i.e. f_{n-1+k} is the (n,n)-minor of $V^{(n,k)}(\mathbf{a})$, which is same as the (n,n)-minor of Vandermonde matrix $V^{(n)}(\mathbf{a})$, which, in turn, is $(-1)^{n+n} * \det\ V^{(n-1)}(\mathbf{a}')$. □

Lemma 3.3.4. *For* $\mathbf{a} = (1^2, 2^2, 3^2, \ldots, n^2)$*, for all* $k \geq 0$*,*

$$\frac{h_{k+1}(\mathbf{a})}{h_k(\mathbf{a})} \leq n^3.$$

Proof. First note that $h_{k+1}(\mathbf{a}) = \sum_{i=1}^{n} a_i * h_k(\mathbf{a}_{(i)})$, where $\mathbf{a}_{(i)}$ is $\mathbf{a}$ restricted to first i entries. Since a_i are monotonically increasing, it follows that $h_{k+1}(\mathbf{a}) \leq n * a_n * h_k(\mathbf{a})$, from which the claim follows. □

Lemma 3.3.5. *For the matrix* $S^{(n)}(\mathbf{a})$ *with* $\mathbf{a}$ *set to the sequence of integers from one to* n*, let* β_i *be given by the formula in Lemma 3.1.1. Then,*

$$\sum_{i=1}^{n} \beta_i * i^{2n+1} = (-1)^{n-1} * (n!)^2.$$

Proof. With $\mathbf{a}$ set to the sequence of integers from one to n, $\sum_{i=1}^{n} \beta_i * i^{2n+1}$ is the inner product of the first column of $\hat{S}^{(n)}(\mathbf{a})$ and $\vec{\beta}$. In the following, the i-th column of a matrix M will be denoted by M_i, and the (i,j)-th entry of M will be denoted by $M_{i,j}$. Thus, using

Lemma 3.1.1, we have

$$\begin{aligned}
\sum_{i=1}^{n} \beta_i * i^{2n+1} &= \vec{\beta}^\top \cdot (\hat{S}^{(n)}(\mathbf{a}))_1 \\
&= \frac{1}{\det(S^{(n)}(\mathbf{a}))} * \sum_{i=1}^{n} (-1)^{i+1} d_i * (\hat{S}^{(n)}(\mathbf{a}))_{i,1} \\
&= \frac{\det(\hat{S}^{(n)}(\mathbf{a}))}{\det(S^{(n)}(\mathbf{a}))} \\
&= (-1)^{n-1} * \prod_{i=1}^{n} a_i^2 \\
&= (-1)^{n-1} * (n!)^2,
\end{aligned}$$

where we have used Lemma 3.1.2 in the second-to-last equality. □

3.3.1 Alternating Series Test (Proof of the Main Lemma)

Having shown Lemmas 3.3.3, 3.3.4, and 3.3.5, we are now ready to prove the main lemma (Lemma 3.3.1).

Proof. (of Lemma 3.3.1) In this proof, we will fix $\mathbf{a}$ to be the sequence of integers from 1 to n. Note, each b_m can be written as $b_m = c_m * \frac{x^{2m-1}}{(2m-1)!}$, where $c_m = \sum_{j=1}^{n} \beta_j * j^{2m-1}$. We now prove the three properties required of b_m so that the series $\sum_{m=n+1}^{\infty} (-1)^m * b_m$ satisfies the alternating series test.

1. We show that b_m goes to zero, as m goes to infinity. Since n is fixed and all β_i are bounded by Lemma 3.1.3, we just need to show that for every x in the domain of approximation, for every $j \in [n]$, $\frac{(jx)^{2m-1}}{(2m-1)!}$ goes to zero as m goes to infinity. Since the

domain of approximation is bounded, $|x|$ itself is bounded. Since, $k! \geq e(k/e)^k$, the above is upper bounded by $e^{-1} * (jx * e/(2m-1))^{2m-1}$, which goes to zero as m goes to infinity.

2. To show that all b_m are positive (or all are negative), it suffices to show that all c_m are positive (or all c_m are negative). As a warmup, we first focus on c_{n+1} (i.e. m set to $n+1$). By Lemma 3.3.5, this quantity is simply $(-1)^{(n-1)} * (n!)^2$ and hence is positive if n is odd, and negative when n is even.

   Let $\hat{S}^{(n,k)}(\mathbf{a})$ be the matrix that is the same as $\hat{S}^{(n)}(\mathbf{a})$ except that the first column is replaced by the $(2n-1+2k)$ powers of $\mathbf{a}$. Thus, $\hat{S}^{(n,1)}(\mathbf{a})$ is same as $\hat{S}^{(n)}(\mathbf{a})$. As in the proof of Lemma 3.3.5,

$$\begin{aligned} c_{n+k} &= \sum_{i=1}^{n} \beta_i * i^{2n-1+2k} \\ &= \vec{\beta}^\top \cdot (\hat{S}^{(n,k)}(\mathbf{a}))_1 \\ &= \frac{1}{\det(S^{(n)}(\mathbf{a}))} * \sum_{i=1}^{n} (-1)^{i+1} d_i * (\hat{S}^{(n,k)}(\mathbf{a}))_{i,1} \\ &= \frac{\det(\hat{S}^{(n,k)}(\mathbf{a}))}{\det(S^{(n)}(\mathbf{a}))} \end{aligned}$$

   To give an expression for $\det(\hat{S}^{(n,k)}(\mathbf{a}))$, we will use Lemma 3.3.3. To use this lemma, we first relate $\hat{S}^{(n,k)}(\mathbf{a})$ to $V^{n,k}(\mathbf{a})$. Recall, the first column of $\hat{S}^{(n,k)}(\mathbf{a})$ is $(2n-1+2k)$ powers of $\mathbf{a}$. Also, for other columns, the (i,j)-th entry is a_i^{2j-1} $(2 \leq j \leq n)$. Since $k \geq 1$, each entry in the i-th row has at least one power of a_i, and hence the determinant of $\hat{S}^{(n,k)}(\mathbf{a})$ is $\prod_{i=1}^{n} a_i$ times the determinant of a new matrix M, which has as its first column $(2n+2(k-1))$ powers of $\mathbf{a}$, and all other columns as $2(j-1)$-th powers of $\mathbf{a}$ $(2 \le j \le n)$. Let $\mathbf{a}^{(2)}$ be the sequence $\mathbf{a}$, but with each entry squared. Then this matrix M is same as the matrix $V^{n,k-1}(\mathbf{a}^{(2)})$ but with the first and last column exchanged. Thus, using Lemma 3.3.3, it follows that $\det(\hat{S}^{(n,k)}(\mathbf{a}))$ is

$$(-1)^{n-1} * h_{k-1}(\mathbf{a}^{(2)}) * \prod_{i=1}^{n} \prod_{1 \le j < i} (a_i^2 - a_j^2) \; * \; \prod_{i=1}^{n} a_i^3,$$

From Lemma 3.1.2, we also have that the determinant of $S^{(n)}(\mathbf{a})$ is

$$\left(\prod_{i=1}^{n} a_i \right) * \prod_{i=1}^{n} \prod_{1 \le j < i} (a_i^2 - a_j^2).$$

Recalling that a_i is just i, we thus have that for $k \ge 1$, all c_{n+k} are positive if n is odd, and all c_{n+k} are negative if n is even.

3. We now show that $|b_m| \ge |b_{m+1}|$ for all $m \ge n+1$. We have,

$$\begin{aligned}
\frac{|b_{m+1}|}{|b_m|} &= \frac{(-1)^{n-1} * h_{m+1-(n+1)}(\mathbf{a}^{(2)}) * \prod_{i=1}^{n} \prod_{1 \le j < i} (a_i^2 - a_j^2) \; * \; \prod_{i=1}^{n} a_i^3 \; * \; \frac{x^{2m+1}}{(2m+1)!}}{(-1)^{n-1} * h_{m-(n+1)}(\mathbf{a}^{(2)}) * \prod_{i=1}^{n} \prod_{1 \le j < i} (a_i^2 - a_j^2) \; * \; \prod_{i=1}^{n} a_i^3 \; * \; \frac{x^{2m-1}}{(2m-1)!}} \\
&= \frac{h_{m+1-(n+1)}(\mathbf{a}^{(2)}) \; * \; \frac{x^{2m+1}}{(2m+1)!}}{h_{m-(n+1)}(\mathbf{a}^{(2)}) \; * \; \frac{x^{2m-1}}{(2m-1)!}} \\
&= \frac{h_{m+1-(n+1)}(\mathbf{a}^{(2)})}{h_{m-(n+1)}(\mathbf{a}^{(2)})} \; * \; \frac{x^2}{2m(2m+1)} \\
&\le n^3 \; * \; \frac{x^2}{2m(2m+1)} \text{ (by Lemma 3.3.4)} \\
&\le 1 \text{ (for } |x| < 2/\sqrt{n}).
\end{aligned}$$

$\square$

We are now ready to prove Theorem 3.0.1.

Proof. (of Theorem 3.0.1) Let β_k, for $k \in [1..n]$, be defined as in equation (3.1) with $\mathbf{a}$ set to the sequence of numbers from 1 to n. From the Taylor series expansion of the sine series, which converges since the sine series is analytic, it follows that

$$\sum_{k=1}^{n} \beta_k \sin(kx) = x - \sum_{m=n+1}^{\infty} (-1)^m * b_m,$$

where b_m are defined in equation (3.2), i.e. $b_m = \sum_{k=1}^{n} \beta_k * \frac{(kx)^{2m-1}}{(2m-1)!}$. Thus, by Lemma 3.3.1 and Leibniz's alternating series test (Theorem 3.3.1), we have for $|x| < 2/\sqrt{n}$,

$$\begin{aligned}\left| x - \sum_{k=1}^{n} \beta_k \sin(kx) \right| &\leq |b_{n+1}| \\ &= \left| \sum_{k=1}^{n} \beta_k * \frac{(kx)^{2n+1}}{(2n+1)!} \right| \\ &= \frac{|x^{2n+1}|}{(2n+1)!} * \left| \sum_{k=1}^{n} \beta_k * k^{2n+1} \right| \\ &= \frac{(n!)^2}{(2n+1)!} * |x^{2n+1}|,\end{aligned}$$

where we used Lemma 3.3.5 in the last equality.

Restricting $|x| < \epsilon$, Theorem 3.0.1 follows from the fact that

$$\begin{aligned}
\frac{(n!)^2}{(2n+1)!}\epsilon^{2n+1} &< \frac{((n+1)/e)^{2n+2}e^2}{((2n+1)/e)^{2n+1}}\epsilon^{2n+1} \\
&= e * (n+1) * \left(\frac{n+1}{2n+1}\right)^{2n+1} * \epsilon^{2n+1} \\
&= e * (n+1) * \left(\frac{2n+2}{2n+1}\right)^{2n+1} * \left(\frac{\epsilon}{2}\right)^{2n+1} \\
&< e^2 * (n+1) * \left(\frac{\epsilon}{2}\right)^{2n+1},
\end{aligned}$$

where we have used the fact that

$$\left(\frac{n}{e}\right)^n < n! < \left(\frac{n+1}{e}\right)^{n+1} e$$

for all $n \geq 1$ and that $(1+1/n)^n < e$ for all $n \geq 1$. □

3.4 Evaluating the Sine Series Approximation of the Mod Function

In order to use the sine series approximation of the mod function given by Corollary 3.0.2 for bootstrapping, we must approximate the sine series by a low-degree polynomial, since the CKKS-HE scheme cannot compute sine directly. In this section, using our sine series approximation of the mod function and the well-known Taylor series expansion of the sine function, we will give explicit low-degree polynomial approximations of the mod function on small intervals around multiples of the modulus to (almost) arbitrary precision. The resulting polynomials have small coefficients, as the Taylor series of the sine function has small coefficients, and the sine series itself has small coefficients by Lemma 3.1.3. Recall that small coefficients are beneficial in contrast to large coefficients, as in the latter case one is forced to compute the different power monomials to much higher precision in order to obtain an accurate polynomial evaluation. This, in turn, causes the computational precision that we must operate at during bootstrapping to be higher, which causes each "level" to consume more bits of the modulus. We next explain how we evaluate the sine series and then determine the degree and evaluation precision required for the Taylor series approximation of sine.

Evaluating the Sine Series

To evaluate the sine series, we first compute a Taylor series approximation of e^{ix} (recall that CKKS-HE allows us to compute over complex numbers). We can obtain an approximation to $\sin x$ by extracting the imaginary part. The other higher order $\sin kx$ terms can be obtained conveniently by computing e^{ikx} from e^{ix} and extracting the imaginary part. As for computing the Taylor series approximation of the sine function, note that the domain of approximation is small intervals around ℓq, where $\ell \in [-K..K]$ and q is the modulus. The bound K comes from the bound on the Hamming-weight of the secret key and is typically 12 to 32. If our input is $X = x + \ell q$ for some small offset x and $\ell \in [-K..K]$, our goal is to compute $e^{i(2\pi(x+\ell q)/q)}$. This then requires a Taylor series that has powers of $2\pi(x+\ell q)/q$, which can be more than one. Earlier works noted that one can instead first compute $e^{i(2\pi(x+\ell q)/(q2^r))}$ using a Taylor series expansion (for some $r > 0$) and then compute $e^{i(2\pi(x+\ell q)/q)}$ using r squarings.

Determining the Degree of the Taylor Series Approximation

Next, we must determine the degree to which we compute the Taylor series expansion of $e^{2\pi i(x+\ell q)/(q2^r)}$. The Taylor series expansion is

$$\sum_{m=0}^{\infty}(2\pi i(x+\ell q)/(q2^r))^m/m!.$$

We now determine for which range of values of $(x+\ell q)$ the above restricted to the sine terms, i.e. the imaginary terms or odd powers of x, satisfies the alternating series test (so that the partial series error can be bound by the absolute value of the next missing term). Thus, we need to determine the conditions under which

$$\begin{aligned} 1 &> \frac{(2\pi|(x+\ell q)|/(q2^r))^{(2m+1)}/(2m+1)!}{(2\pi|(x+\ell q)|/(q2^r))^{(2m-1)}/(2m-1)!} \\ &= \frac{(2\pi|(x+\ell q)|/(q2^r))^2}{(2m+1)(2m)} \end{aligned}$$

Assuming $x \ll q$ and $2^r \approx K+1$, the above holds when $m > \pi$. Thus, if the Taylor series is computed partially up to any degree $2m-1$, then the error in the approximation of sine is at most

$$(2\pi)^{2m+1}/(2m+1)! < (2\pi e/(2m+1))^{2m+1},$$

which is at most $2^{-(2m+1)}$ if we require that $m > 2\pi e$.

Thus, having computed $\sin(2\pi(x+\ell q)/(q2^r))$ partially up to m terms, we now investigate the error for the higher order terms in the sine series, i.e. $\sin(2\pi k(x+\ell q)/q)$ for $k \geq 1$. If the error in the approximation of the original term is small, say $\delta \ll 1$, then the error for this

k-th term is approximately $k2^r * \delta$ (as it requires $r + \log k$ squarings). Thus, the total error in the sine series due to the Taylor series approximation of $\sum_{k=1}^{n} \beta_k \sin(2\pi k(x + \ell q)/q)$ is upper bounded in absolute value by $\sum_{k=1}^{n} |\beta_k| * k2^r\delta$, which is approximately $(K+1)\delta \sum_{k=1}^{n} |\beta_k| * k$, which is at most $n^2(K+1)\delta$ by Lemma 3.1.3, which, in turn, is at most $n^2(K+1)2^{-(2m+1)}$.

Finally, using Corollary 3.0.2, the total error in the mod function approximation, for an input $X = x + \ell q$ with $\ell \in [-K..K]$ and $|x| < \epsilon * q$ for any $\epsilon < 1/\pi\sqrt{n}$ is

$$(q/2\pi) * n^2(K+1)2^{-(2m+1)} + \frac{e^2 * q}{2\pi} * (n+1) * (\epsilon * \pi)^{2n+1}.$$

Thus, it makes sense to have m about $-n \log_2(\epsilon * \pi)$ (which is typically greater than $2\pi e$ for $n > 1$; if this value is less than $2\pi e$, then the above analysis must be redone for potentially a larger r).

Determining the Evaluation Precision

We must also determine the precision to which to evaluate the polynomials. Setting $Y = 2\pi(x + \ell q)/(q2^r)$, we observe that the degree m Taylor expansion of $e^{2\pi i(x+\ell q)/(q2^r)}$ is simply the polynomial

$$\sum_{j=0}^{m} (iY)^j / j!.$$

Recall that we have chosen r so that $|Y| < 1$. Moreover, setting $c_j = i^j/j!$, the polynomial becomes $\sum_{j=0}^{m} c_j Y^j$, where $|c_j| \leq 1$. We need to determine the precision to which we evaluate the powers Y^j (we will first evaluate the Y^{2^j}'s by repeated squaring and then use these powers to evaluate all intermediate powers). Let Y^j denote the exact values and let $\tilde{Y}^j$ denote the approximated values (to some precision to be determined). Suppose we evaluate the powers Y^j up to w bits (and simply chop off the additional bits). Then, $|\tilde{Y} - Y| < 2^{-w}$. Computing $\tilde{Y}^2$ by squaring $\tilde{Y}$ and rounding, we have that $\tilde{Y}^2$ differs from Y^2 by at most $\approx 2 * 2^{-w}$. To see this, note that $\tilde{Y} = Y \pm \delta$, where $\delta < 2^{-w}$. Then, $\tilde{Y}^2 = Y^2 \pm 2Y\delta + \delta^2 < Y^2 \pm 2\delta + \delta^2 \approx Y^2 \pm 2 * 2^{-w}$. By an analogous argument, it follows that $\tilde{Y}^j$ differs from Y^j by at most approximately $j * 2^{-w}$. Thus, the error of $\sum_{j=0}^{m} c_j \tilde{Y}^j$ is bounded by

$$\sum_{j=0}^{m} j * 2^{-w} * \frac{1}{j!} = \sum_{j=1}^{m} \frac{2^{-w}}{(j-1)!} < e * 2^{-w}.$$

Thus, to obtain error 2^{-d}, it suffices to compute the powers $\tilde{Y}^j$ to precision w for $w > d + \log_2 e$, only slightly higher than the minimum precision d required to obtain this approximation.

In the above, we saw that having small coefficients c_j (and coefficients that decrease in magnitude as j increases) enabled the approximation of the polynomial $\sum_{j=0}^{m} c_j Y^j$ by evaluating the powers of Y to precision only a couple bits larger than the minimum precision required for the desired error. This is crucial during bootstrapping as a higher evaluation precision directly corresponds to losing more bits of the modulus during the polynomial evaluation. In contrast, suppose that the c_j's were large and bounded in magnitude $|c_j| < 2^k$ for some k. Then, if the powers of Y are evaluated to precision w, the error of the polynomial evaluation is bounded by

$$\sum_{j=0}^{m} j * 2^{-w} * 2^k < \frac{m(m+1)}{2} * 2^{k-w}.$$

Thus, to obtain error 2^{-d}, the powers of Y would need to be evaluated to precision $w > d + k + 2\log m - 1$. Note the additional dependence on both k and the number of terms m.

Chapter 4

Implementation and Evaluation

To demonstrate the applicability of our polynomial approximation to high precision bootstrapping for approximate homomorphic encryption, we updated the bootstrapping procedure of the HEAAN library [HEA] to utilize our sine series during the "Compute the Mod Function" step (see Section 2.5). Additionally, we updated HEAAN to use the quadmath library, since we wanted to achieve bootstrapping error smaller than the precision of a double. We ran our implementation using a PC with an AMD Ryzen 5 3600 3.6 GHz 6-Core CPU.

Table 4.1 gives our bootstrapping results for sine series of various orders. As before, ϵ represents the ratio p/q, where p is an upper bound on the size of the message (including any errors associated from the approximate arithmetic and prior homomorphic operations) and q is the size of the modulus prior to bootstrapping. In Table 4.1, ϵ is set to 2^{-10}. The Hamming weight of the secret key is set to $h = 256$, so that on average K is about $\sqrt{h} = 16$. However, our implementation can handle K as large as 31. q_L denotes the modulus of the largest level, which is the modulus of a fresh ciphertext prior to any homomorphic

Table 4.1: High-Precision Bootstrapping Results for $\epsilon = 2^{-10}$. The Hamming weight of the secret key is set to $h = 256$. The errors reported are for K up to 31.

Input	Sine	Modulus	Ring	Boot.	Modulus	Error	Runtime[††]
Precision[†]	Series	(Fresh)	Dim.	prec.	(After)	(Boot.)	(secs)
$\log_2 p$	Order	$\log_2 q_L$	N		$\log_2 q_{\ell'}$	$\beta_{\mathsf{bs}} = \text{err}/p$	
30	2	1200	2^{16}	55	344	2^{-25}	22
50	3	1600	2^{16}	75	531	2^{-45}	32
60	4	2400	2^{17}	85	1008	2^{-54}	119
80	5	2400	2^{17}	105	583	$< 2^{-80}$	129
100	6	3000	2^{17}	125	843	$< 2^{-100}$	167

[†] The modulus q_ℓ of the ciphertext prior to bootstrapping is p/ϵ. The number of bits of q_ℓ is $p - \log\epsilon = p + 10$, and bootstrapping (computational) precision is set to $(p - \log\epsilon + \log_2 K) + 10$.

[††] Includes runtime of "Coefficients to Slots" and "Slots to Coefficients" steps. Number of slots fixed to be 8 so that the "Compute the Mod Function" step dominates runtime. Results reported are from an AMD Ryzen 5 3600 3.6 GHz 6-Core CPU using quadmath, NTL and GMP software libraries.

operations. N denotes the ring dimension, which we increase as q_L increases to maintain 128-bit security [CP19, Alb17, APS15]. Results in this table were obtained using 8 slots, and the dependence on a larger number of slots is reported below. $q_{\ell'}$ denotes the modulus of the ciphertext after bootstrapping. The reported error is the decryption error after performing bootstrapping. In other words, if the decryption before bootstrapping would have resulted in message slot value M, then the decryption after bootstrapping would result in a message slot value M' such that $|M' - M| \leq \beta_{\mathsf{bs}}|M|$. As can be seen from Table 4.1, for $\log_2 p = 80$ and $\log_2 p = 100$, the bootstrapping error is essentially zero. This is because the bootstrapping procedure is performed at a precision that is ten bits more than the number of bits required to represent $M + Kq$ (i.e. the value which needs to be reduced mod q).

Recall that the sine series approach begins by approximating e^{ix} using a Taylor series approximation, since CKKS-HE allows computation on complex numbers. In this particular implementation, we approximated $e^{ix/K}$ to degree 63 using the Paterson-Stockmeyer polynomial evaluation optimization [PS73] and then performed $\log K$ squarings to obtain an approximation of e^{ix}. Below, we report results for other variants for approximating e^{ix}.

We see that our methodology is capable of achieving high precision bootstrapping, with the resulting message precision as large as 100 bits. Prior to our work, the highest precision bootstrapping of CKKS was the recent work of [JM20] which could achieve a resulting message precision of up to 67 bits. However, that result was only for $K = 12$ and secret key Hamming weight $h = 64$, whereas our 100 bit precision bootstrapping is for $h = 256$ and can handle K up to 31. Observe that using a sparser key (in addition to weakening security) reduces the number of intervals required for approximation, making the approximation

Table 4.2: Timing and Error Dependence on Number of Slots. In this table $\epsilon = 2^{-10}$, $\log_2 p = 80$, and the sine series order is fixed to $n = 5$.

Num Slots	Input Precision $\log_2 p$	Sine Series Order	Modulus (Fresh) $\log_2 q_L$	Ring Dim. N	Boot. prec.	Modulus (After) $\log_2 q_{\ell'}$	Error (Boot.) $\beta_{\mathsf{bs}} = \mathrm{err}/p$	Runtime[††] (secs)
8	80	5	2400	2^{17}	105	583	$< 2^{-80}$	129
16	80	5	2400	2^{17}	105	583	$< 2^{-80}$	151
32	80	5	2400	2^{17}	105	583	2^{-72}	178
64	80	5	2400	2^{17}	105	583	2^{-71}	208
128	80	5	2400	2^{17}	105	583	2^{-69}	269

[††] Includes runtime of "Coefficients to Slots" and "Slots to Coefficients" steps. For all rows, the mod function evaluation time is almost the same at 82 secs.

easier. Thus, we view our result as a substantial improvement for bootstrapping in settings where high precision is required, such as the inference step of a convolution neural network or even the learning stage of the neural network. As mentioned earlier, since CKKS is for approximate arithmetic, it is only possible to have unlimited computation for stable computations that do not lose precision. However, even such stable computations lose precision in early stages prior to convergence. Thus, it is important to begin such computations with high precision and, later, one can switch to smaller precision during the stable regime.

4.1 Time and Error Dependence on the Number of Slots

As the number of slots is increased, the time of the mod function evaluation step during bootstrapping remains the same (assuming we use at most $N/4$ slots, so that all the polynomial coefficients can be packed into a single ciphertext during the "Coefficients To Slots" step). However, the linear transforms that send the coefficients to slots and vice versa take a substantial hit since their runtime scales with the number of slots. Since the linear transforms also involve more rotations, key-switchings, multiplications by constants, and additions, for every doubling of the number of slots, the bootstrapping error also increases proportionately. However, since our error is so low, the error for a high number of slots still remains low enough to be termed high-precision. This dependence of runtime and bootstrapping error is reported in Table 4.2 for one particular parameter, where the sine series is of order five. Observe that for 8 and 16 slots, our bootstrapping method gives essentially no error. However, for a larger number of slots, the error is about 2^{-69}. This is because once the number of slots becomes larger, the error is dominated by the error introduced during the linear transform steps.

4.2 Comparison with Basic Sine and Other Variants

While the implementation results reported in Table 4.1 used a Taylor series approximation of degree 63 of $e^{ix/K}$, the implementation in [HEA] instead used a degree 7 approximation of $e^{ix/K*2^4}$ followed by 4 additional squarings. We investigated if we could use a similar approach for the sine series, as the different order sine terms are obtained by squarings of e^{ix} anyways. We found that for small precision, i.e. $\log_2 p \leq 40$, this approach can lead to a faster implementation while yielding effectively the same error. However, for $\log_2 p \geq 50$, this approach led to substantially worse error. For example, at $\log_2 p = 50$, the error increased from 2^{-45} to 2^{-30}. But, as mentioned, for smaller $\log_2 p$ we get the following improvements. First of all, the basic sine approach (i.e. $n = 1$) with $r = 4$ and degree 7 Taylor series yields an error of 2^{-19} for $\log_2 p = 30$. If the fresh modulus used is 1600 bits, then the modulus after bootstrapping has 795 bits. The time taken is 10.5 secs. Interestingly, with sine series of order two, i.e. $n = 2$, using the same approach we get an error of 2^{-26}, with modulus after bootstrapping having 685 bits. Moreover, the time taken is 10.7 secs. Yet another implementation, with a degree 31 Taylor series approximation, and $r = 0$, also yields error 2^{-25}, but takes time 16.5 secs. However, the modulus after bootstrapping has more bits at 744 bits. Regardless, it seems that the sine series of order two with a degree 7 Taylor series and $r = 4$ seems to be beneficial at low precision.

We also experimented with different values of ϵ, in particular ϵ set to 2^{-5}, 2^{-10}, 2^{-15}, 2^{-20}. The errors at each input precision were not much different, and, in fact, $\epsilon = 2^{-10}$ seems to be the best option.

4.3 Comparison with Other Prior Works

The work [CCS19] followed an interesting approach of obtaining Chebyshev interpolants of the scaled sine function. In particular, using the Taylor series of $\sin(2\pi K \cos x)$, they obtained approximations of $\sin(2\pi K x)$ in terms of Chebyshev polynomials. Furthermore,

Table 4.3: Comparison with [LLL+21]. Note, [LLL+21] cites results for $K = 25$, whereas our results are for K up to 31.

[LLL+21]			This Work		
Key Hamming Weight (h)	Ciphertext Bits Lost	Bootstrapping Precision (bits)	Key Hamming Weight (h)	Ciphertext Bits Lost	Bootstrapping Precision (bits)
192	1080	40.5	256	1069	44
N/A	N/A	N/A	256	1392	53
N/A	N/A	N/A	256	1817	80
N/A	N/A	N/A	256	2157	100

Table 4.4: Comparison with Modular Lagrange Interpolation [JM20]. Note, [JM20] cites results for $K = 12$, whereas our results are for K up to 31.

	[JM20]			This Work		
Input Precision	Key Hamming Weight (h)	Ciphertext Bits Lost	Error (Boot.)	Key Hamming Weight (h)	Ciphertext Bits Lost	Error (Boot.)
30	64	935	2^{-24}	256	856	2^{-25}
50	64	1725	2^{-46}	256	1069	2^{-45}
60	64	1800	2^{-54}	256	1392	2^{-54}
80	64	2150	2^{-63}	256	1817	$< 2^{-80}$
100	N/A	N/A	N/A	256	2157	$< 2^{-100}$

this approach also leads to an almost optimal minmax polynomial approximation, as well as yielding small coefficients. Since the scaling K is already incorporated in the function, it removes the $\log K$ squarings required in [CHK$^+$18a] and in this work. However, Chebyshev interpolants do not readily submit to the Paterson-Stockmeyer evaluation optimization and while [CCS19] did show a variant of this method, it leads to coefficients increasing in size. Thus, as explained in Section 2.5, this then requires a larger computational precision that leads to loss of many more (ciphertext modulus) bits per multiplication depth in the bootstrapping circuit. For a direct comparison of our approach to [CCS19], we take data from Tables 2-4 from that work, as their implementation is unfortunately not public, and note that the best approximation they obtain has error 2^{-21} for data set IV*. A look at our Table 4.1 shows that the worst error we obtain is 2^{-25} for $\log_2 p = 30$. The number of ciphertext (modulus) bits lost for that error is $1200 - 344 = 856$, whereas [CCS19] loses $1240 - 43 * 6 = 982$ bits. Moreover, our implementation can handle K up to 31 since we set the secret key Hamming weight $h = 256$, whereas [CCS19] gives results for $K = 12$ and use $h = 64$. Thus, our approach is clearly better at even this low precision.

In [HK20], the authors obtain better approximation error than [CCS19] by leveraging the fact the approximation is only needed in small intervals around multiples of the modulus. However, their approach also uses a baby-step giant-step, or alternately the Paterson-Stockmeyer variant applied to Chebyshev polynomials that can lead to a blowup in the size of coefficients. The authors do not give details on the number of ciphertext (modulus) bits lost in the bootstrapping procedure, nor is their implementation public. The maximum bootstrapping precision they achieve is 18.5 bits.

In [LLL$^+$21], the authors report high-precision bootstrapping using a composition of sine/cosine and arcsine. The polynomials to approximate these functions are found via algorithmic search using the Remez algorithm (which gives no guarantee on the size of the coefficients), and the authors do not provide any details on the size of these coefficients apart from noting that they "are small enough not to distort the messages." Moreover, their implementation is not public. The authors report a practical implementation of up to 40-bits precision bootstrapping. In Table 4.3, we compare our results with theirs using the relevant available information in their paper. We note that [LLL$^+$21] gives an implementation of RNS-CKKS [CHK$^+$18b], which improves performance over the original CKKS implementation by utilizing an RNS basis. This introduces an additional challenge of having to ensure that rescaling errors are small, but this can be done without significantly increasing error, and, in fact, the recent work [KPP20] shows a method of managing the scaling factor so that homomorphic multiplication error in RNS-CKKS is about the same as that of the original CKKS scheme.

The work [JM20] gives a direct approximation of the mod function, i.e. without going through the sine function, and hence bypasses the fundamental error of the sine function approach. Thus, they can get arbitrarily high precision, and they also show that the coefficients of their polynomial approximation are not too large. Nevertheless, the coefficients are large enough that our approach beats [JM20]. Moreover, they only give implementation numbers for $K = 12$, and for $K = 31$, the number of ciphertext modulus bits lost during bootstrapping would be higher. In Table 4.4, we compare their results with ours for $\epsilon = 2^{-10}$ and various plaintext precisions.

The recent work [BMTPH21] optimized the performance of bootstrapping for RNS-CKKS. They introduce a scale-invariant polynomial evaluation method as well as a "double hoisting" technique for evaluating the homomorphic linear transforms. These techniques improve the performance of bootstrapping considerably and are compatible with our sine series approximation of the mod function. Moreover, to the best of our knowledge, [BMTPH21] gives the first public implementation of full RNS-CKKS with bootstrapping. We note that they do not focus on obtaining better approximations to the mod function and utilize previous techniques and variants thereof to perform the "Compute the Mod Function" step in bootstrapping. Their maximum bootstrapping precision achieved is 32.6 bits, but we stress that this was not the focus of their work. An interesting direction would be to combine both their bootstrapping optimizations with our sine series approximation of the mod function.

References

[Alb17] Martin R. Albrecht. On Dual Lattice Attacks Against Small-Secret LWE and Parameter Choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 103–129, Cham, 2017. Springer International Publishing.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS '12*, 2012.

[BHHH19] Flavio Bergamaschi, Shai Halevi, Tzipora T. Halevi, and Hamish Hunt. Homomorphic Training of 30,000 Logistic Regression Models. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 592–611, Cham, 2019. Springer International Publishing.

[BMTPH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing.

[Bra12] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, pages 868–886, 2012.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved Bootstrapping for Approximate Homomorphic Encryption. In *EUROCRYPT*, pages 34–54, 2019.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[CHK$^+$18a] Jung Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for Approximate Homomorphic Encryption. In *EUROCRYPT*, pages 360–384, 01 2018.

[CHK$^+$18b] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography – SAC 2018*, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT*, 2017.

[CP19] Benjamin R. Curtis and Rachel Player. On the Feasibility and Impact of Standardising Sparse-secret LWE Parameter Sets for Homomorphic Encryption. In Michael Brenner, Tancrède Lepoint, and Kurt Rohloff, editors, *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019*, pages 1–10. ACM, 2019.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, 2015.

[ElG84] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology – CRYPTO*, pages 10–18, 1984.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology – CRYPTO 2013*, pages 75–92, 2013.

[HEA] HEAAN. https://github.com/snucrypto/HEAAN.

[HHC19] K. Han, M. Hhan, and J. H. Cheon. Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. *IEEE Access*, 7:57361–57370, 2019.

[HK20] Kyoohyung Han and Dohyeong Ki. Better Bootstrapping for Approximate Homomorphic Encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 364–390, Cham, 2020. Springer International Publishing.

[JM20] Charanjit S. Jutla and Nathan Manohar. Modular Lagrange Interpolation of the Mod Function for Bootstrapping of Approximate HE. Cryptology ePrint Archive, Report 2020/1355, 2020. `https://eprint.iacr.org/2020/1355`.

[KHB$^+$20] Miran Kim, Arif Harmanci, Jean-Philippe Bossuat, Sergiu Carpov, Jung Cheon, Ilaria Chilotti, Wonhee Cho, David Froelicher, Nicolas Gama, Mariya Georgieva, Seungwan Hong, Jean-Pierre Hubaux, Duhyeong Kim, Kristin Lauter, Yiping Ma, Lucila Ohno-Machado, Heidi Sofia, Yongha Son, Yongsoo Song, and Xiaoqian Jiang. Ultra-Fast Homomorphic Encryption Models enable Secure Outsourcing of Genotype Imputation. bioRxiv, 2020.

[KPP20] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate Homomorphic Encryption with Reduced Approximation Error. *IACR Cryptol. ePrint Arch.*, page 1118, 2020.

[KSK$^+$18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics*, 11(4):83, 2018.

[KSW$^+$18] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation. *JMIR Med Inform*, 6(2):e19, Apr 2018.

[LLL$^+$21] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In *Advances in Cryptology – EUROCRYPT 2021*, pages 618–647, 2021.

[LM21] Baiyu Li and Daniele Micciancio. On the Security of Homomorphic Encryption on Approximate Numbers. In *Advances in Cryptology – EUROCRYPT 2021*, pages 648–677, Cham, 2021. Springer International Publishing.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A Toolkit for Ring-LWE Cryptography. In *Advances in Cryptology – EUROCRYPT 2013*, pages 35–54, 2013.

[MHS$^+$20] Oliver Masters, Hamish Hunt, Enrico Steffinlongo, Jack Crawford, Flavio Bergamaschi, Maria E. Dela Rosa, Caio C. Quini, Camila T. Alves, Feranda de Souza, and Deise G. Ferreira. Towards a Homomorphic Machine Learning Big Data Pipeline for the Financial Services Sector. In *RWC*, 2020.

[PS73] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2:pp. 60–66, 1973.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg09] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM*, 56(6), sep 2009.

[Rem34] Gilbert Remez, E. Sur la determination des polynomes d'approximation de degre' donnee'. *Comm. of the Kharkov Math. Soc.*, 10(196):41–63, 1934.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SPTP$^+$20] Sinem Sav, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. POSEIDON: Privacy-Preserving Federated Neural Network Learning, 2020.