Keywords
CKKS, Oral Surgery, Artificial Intelligence in Dentistry, Privacy-Preserving Computation, Cloud Security, Encrypted Web Processing

Authors
[1]Mahesh Pokharkar
Department of Computer Science and Engineering, VGU, Jaipur, India.
Email:maheshpokharkar5@gmail.com
Orcid id: 0009-0008-0276-5425

[2]Surendra Yadav
Department of Computer Science and Engineering, VGU, Jaipur, India.Email: surendra.yadav@vgu.ac.in,
Orcid Id: 0000-0002-6519-4788

[3]Abhijit Banubakode
Department of Computer Science & Engineering, PCU, Pune, India, Email: abhijitsiu@gmail.com,
Orcid Id: 0000-0003-4389-0026

Received: 01.05.2026
Revised: 07.05.2026
Accepted: 12.05.2026
Doi:
10.1922/ejprd.v34i1s.1372

Privacy-Preserving AI Computation for Digital Dentistry Using CKKS-Based Homomorphic Encryption in HTTPS at Server Side

Abstract
AI-driven dental imaging and oral diagnostic systems continuously transmit encrypted patient radiographic and clinical data to cloud-based diagnostic platforms for secure intelligent analysis. While server communication is over HTTPS and protects patient dental records when it is sent to the server, once it is received by the server, patient dental records must be decrypted before any other processing occurs. At this stage, the data becomes visible in memory, which leaves them vulnerable to insider misuse, memory disclosure, or compromise of the underlying cloud infrastructure. This work explores how fully homomorphic encryption can be used to extend confidentiality beyond the transport layer.

This study examines how privacy can be preserved during server-side web computation by combining standard HTTPS with the CKKS homomorphic encryption scheme. In the proposed framework, AI-assisted oral diagnostic inference and dental imaging analytics are performed directly on encrypted clinical dental datasets without exposing patient-sensitive information. Therefore, servers will have no access to their decrypted representations (or plain-text forms) throughout the entire process chain. Consequently, the privacy/security aspects of those sensitive pieces of information are still preserved, even when computations occur on any infrastructure that may not have complete trustworthiness attached thereto.

The paper describes the overall client–server architecture, the associated cryptographic workflow, and the CKKS formulation required to support approximate arithmetic under realistic noise constraints. It also gives correctness conditions and noise growth, and SIMD-enabled encrypted inference allows efficient processing of panoramic radiographs, CBCT scans, intraoral images, and AI-driven oral pathology prediction workloads under strict privacy-preserving conditions. The results suggest that CKKS-based homomorphic computation can be practically combined with HTTPS to provide stronger, end-to-end privacy guarantees for modern web services.

1. INTRODUCTION

Artificial Intelligence (AI) is rapidly transforming oral surgery and dentistry through intelligent diagnostic systems, automated radiographic interpretation, AI-assisted prosthodontic planning, and cloud-based dental analytics. Modern digital dentistry platforms increasingly rely on deep learning models for oral lesion detection, dental image segmentation, implant planning, restorative treatment prediction, and AI-assisted clinical decision support. These intelligent systems process highly sensitive patient information including panoramic radiographs, cone beam computed tomography (CBCT) scans, intraoral images, prosthodontic records, and restorative treatment data. To provide these services, servers require user authentication and therefore

process sensitive information such as personal identifiers, financial data, and dental patient records. To protect sensitive information during transmission between clients and servers, HTTPS is often used as the standard security method and relies on Transport Layer Security (TLS). In practice, HTTPS effectively prevents network-level attacks during data exchange. Despite the rapid integration of AI into oral healthcare systems, most cloud-based dental AI platforms require decryption of patient radiographic data before diagnostic inference can occur, thereby exposing confidential oral healthcare records during server-side processing.

This means the data temporarily exists in plaintext form in server memory. At this point, the information is exposed to different risks, such as unintended access or leakage during execution. The problem is worsened in modern cloud settings because resources are shared across many users and controlled by external providers. Even if HTTPS has been set up appropriately, users still face the threat of risks such as misuse of insider information, leakage of memory or access to part of a compromised system. As a result, Transport Layer Security is not adequate for confidentiality during the computation of an application. Fully Homomorphic Encryption (FHE) was designed to resolve this problem by allowing a user to compute directly on encrypted data. Using FHE, a service can compute an encrypted ciphertext without having access to or knowledge of the original unencrypted values.

Despite the widespread deployment of HTTPS, confidentiality guarantees stop when the data is decrypted for server-side processing. The current alternatives, such as Trusted Execution Environments and Secure Multi-Party Computation, have a lot of potential for computation-time exposure but introduce hardware trust assumptions or interaction overheads incompatible with the standard web architectures. Although Fully Homomorphic Encryption has greatly matured, prior studies predominantly have addressed FHE in isolation or within special analytic pipelines. There is still an unfulfilled need to systematically understand integration of FHE with standard HTTPS-based workflows, protocol layering, key management, and noise-aware computation.

In the past, early researchers in FHE developed schemes that were too inefficient due to the large amount of processing power needed to use these methods. However, recent advances have improved the overall efficiency of FHE and increased the number of applications that can utilize FHE effectively. Most notably, the CKKS scheme can handle real-number computations. CKKS simplifies approximate arithmetic and supports efficient multiple ciphertext packing using SIMD techniques.

This paper explores how CKKS-based FHE can be systematically integrated with HTTPS to enable privacy-preserving web computation. Rather than replacing existing web protocols, the proposed framework complements HTTPS by extending confidentiality guarantees beyond data transmission and into encrypted dental AI inference. The primary objective is to demonstrate that such integration is feasible, secure, and practical for a class of web workloads involving numerical processing and inference.

The key points of this paper are as follows:

- The current paper is an idea of a new end-to-end web computation architecture that could rely on not only the HTTPS protocol but also on CKKS-based fully homomorphic encryption providing confidentiality not only to the transmission of data but also to the server-computation process.
- We develop a comprehensive client-server cryptography workflow that is compatible with the existing web architectures and does not need modification of the standard HTTPS protocols.
- It provides formal mathematical definition of CKKS-based mathematical foundation, such as encoding, homomorphic operations, homomorphic encrypted values, rescaling, and the correctness conditions based on the web workloads.
- The security analysis analyses the confidentiality assurances with the Ring-LWE assumption and presents resistance to insider attacks, memory disclosure attacks, and compromised server attacks.
- We offer performance information revealing the feasibility of encrypted computation based on SIMD in both dental imaging analytics and privacy inference.

A web-native HTTPS-CKKS integration has been proposed in this work, unlike earlier FHE-based computation models, where there is an assumed offline analytics or trusted execution hardware, and thus maintains the stateless request-response model. Unlike the current FHE-over-cloud solutions, the introduced framework clearly covers protocol compatibility, encrypted payload lifecycle, and CKKS noise control when working with realistic web workloads. This distinguishes the proposed approach from standalone FHE inference systems and hardware-assisted secure.

2. Related Work

2.1. HTTPS and Transport-Layer Security

Hypertext Transfer Protocol Secure (HTTPS) is the secure version of the standard web communication protocol and builds upon cryptographic mechanisms originally introduced by Secure Sockets Layer (SSL), and later standardized through the Transport Layer Security (TLS) protocol. By encrypting data exchanged between clients and servers, HTTPS has become the foundation for secure information transmission over the Internet. HTTPS provides three levels of protection to address threats to data transmission at a network level: Confidentiality, Integrity, and Authentication. These protections mitigate the potential for data to be intercepted by a network intruder (snooper) or to be altered by an unauthorized user during the transmission process (man-in-the-middle attack).

ejprd.org - Published by Riset Publishing Services LLC

HTTPS is therefore deployed widely by modern web services managed by an individual or organization. However, while HTTPS protects transmitted data, it does not protect it when it is stored on a Web Server (see Normal Process for Storing Data). Thus, once an encrypted connection terminates upon receipt by the Server, the Server is required to decrypt the data in order for any type of processing activity to be conducted upon it; therefore, plaintext data is left exposed within the memory of the Server. This is especially true in the Cloud Computing and Multi-Tenant environments, where the risk of data being exposed via Break-in or Malicious Insider Attack is heightened.

2.2 Trusted Execution Environments (TEEs)

In conclusion, HTTPS security provides protection for data while it is being transmitted across the internet, but does not provide protection to data while it remains in the Server's memory and under the control of the Server. Despite these advantages, TEEs rely heavily on hardware trust assumptions and have been shown to be vulnerable to side-channel, speculative execution, and microarchitectural attacks [11]–[13]. Furthermore, practical deployment challenges such as enclave memory constraints, hardware availability, and platform-specific dependencies limit the scalability of TEE-based solutions in large-scale cloud environments [14].

2.3 Secure Multi-Party Computation (MPC)

MPC allows multiple parties to collaboratively calculate the outcome of a function based on their own unique inputs while maintaining confidentiality of those inputs from each other [15][16]. MPC provides strong theoretical security guarantees and has been successfully applied in privacy-preserving data analysis, collaborative computation, and distributed analytics scenarios.

However, MPC protocols typically require multiple rounds of interaction and incur significant communication overhead [17], [18]. These characteristics make MPC difficult to integrate into latency-sensitive AI-driven dental systems that follow a stateless request–response model, limiting its applicability for practical single-server web computation [19].

2.4 Fully Homomorphic Encryption and CKKS

Fully Homomorphic Encryption (FHE) enables arbitrary computation on encrypted data without requiring decryption, a concept first demonstrated in the seminal work by Gentry [20]. Subsequent research has significantly improved the efficiency and practicality of lattice-based FHE schemes [21], [22]. Among these, the CKKS (Cheon–Kim–Kim–Song) scheme supports approximate homomorphic arithmetic over real and complex numbers, making it particularly suitable for applications such as encrypted machine learning inference, signal processing, and statistical analytics [23]–[25]. CKKS further enables efficient ciphertext packing using Single Instruction, Multiple Data (SIMD) operations, which substantially reduces the amortized cost of encrypted computation [26].

Although CKKS has been extensively studied in standalone computation and machine learning contexts, its systematic integration with standard web communication protocols such as HTTPS remains limited [27], [28]. Nonetheless, recent advances in CKKS efficiency—including optimized bootstrapping techniques [29], [34], Residue Number System (RNS)–based methods [32], and SIMD-optimized homomorphic encryption libraries such as Microsoft SEAL [1], [33]—have made approximate homomorphic computation increasingly viable for real-world applications.

Earlier studies showed that machine learning inference/detection using CKKS-based encrypted search [36], [37], and privacy-preserving statistical analysis [39] were both possible, whereas hardware-accelerated Fully Homomorphic Encryptions (FHE) were only partly achievable at the time. Recent efforts have focused more on improving the usability, performance, and deployability of FHE than previous works did. For example, Al Badawi et al. [42] developed an open-source FHE library called OpenFHE that combines many different types of homomorphic encryption algorithms into a single package and adds support for operations like bootstrapping and scheme switching via a standardized abstraction layer. Gong et al. [43] also performed an extensive review of FHE acceleration techniques, which included an evaluation of both software and hardware options (e.g. GPU and FPGA) that help to mitigate CKKS's high computational costs. In applied communication settings, Wang et al. [44] proposed a privacy-preserving network slicing framework that integrates attribute-based encryption with fully homomorphic encryption to enhance data confidentiality in communication networks.

Collectively, these recent studies highlight substantial progress in FHE library design, acceleration strategies, and applied deployments. At the same time, they reinforce the need for practical frameworks that integrate CKKS-based homomorphic computation with existing web security mechanisms such as HTTPS—an objective directly addressed by the proposed HTTPS–CKKS architecture in this work.

2.5 Research Gap and Motivation

Existing AI-driven dental diagnostic systems primarily rely on plaintext processing of sensitive oral healthcare data during cloud-based inference. Although AI models have demonstrated high diagnostic accuracy in oral radiology, implant planning, oral pathology detection, and restorative treatment analysis, the confidentiality of patient dental records remains a major concern. Current privacy-preserving approaches either rely on trusted hardware assumptions or introduce substantial communication overhead unsuitable for real-time dental AI workflows. There remains a significant lack of practical frameworks integrating fully homomorphic encryption with AI-assisted oral diagnostic systems for secure encrypted inference in oral surgery and dentistry [5], TEEs protect data in use but rely on hardware trust assumptions [11], and MPC provides strong privacy guarantees at the cost of high interaction and communication overhead [17].

There is a clear lack of end-to-end frameworks that combine HTTPS with CKKS-based homomorphic computation to preserve confidentiality throughout data

transmission, storage, and server-side processing without modifying existing web architectures. This paper addresses this gap by proposing a practical HTTPS–CKKS integration for privacy-preserving web computation.

3. Threat Model and Assumptions

3.1 Threat Model

The proposed framework is based on an honest-but-curious (semi-honest) server model. In this model, the server follows all protocols correctly but may try to infer sensitive information from encrypted data, intermediate computation states, memory contents, or persistent storage.

The adversary is assumed to have access to server-side resources such as main memory, logs, and stored ciphertexts. This access can come from insider privileges or partial system breaches. However, it is reasonable to assume that the opponent will be unable either to break into the standard cryptographic methods or use those standard methods to solve for the underlying lattice structures that secure the CKKS protocol. Threats to the network level are addressed by employing the use of HTTPS with TLS 1.3; this protocol guarantees both confidentiality and integrity of data transmitted between a user and a server.

3.2 Trust and Scope Assumptions

The client environment is considered trusted and is responsible for securely generating keys, as well as for encryption and decryption. The secret key is never shared with the server and stays under client control throughout the computation lifecycle.

The framework assumes HTTPS is correctly implemented with modern cryptographic settings. Side-channel attacks on client devices, denial-of-service attacks, traffic analysis, and availability attacks are not included in this work. The main focus is on maintaining confidentiality during encrypted dental AI inference.

These exclusions help concentrate the analysis on the confidentiality guarantees provided by cryptographic protection instead of on availability or hardware-level defences. Client key material can be regenerated to support key rotation and revocation. Corresponding evaluation keys can be invalidated at the server without affecting the HTTPS protocol or server-side plaintext access.

4. System Architecture

The proposed system architecture has been developed to support privacy-preserving AI-assisted oral diagnostics, encrypted dental imaging analytics, and secure cloud-based oral surgery inference while remaining compatible with existing digital dentistry infrastructure. The system separates the client, transport and server layers in terms of functionality.

The client layer is involved in choosing cryptographic parameters, the generation of CKKS keys, encoding plaintexts, encrypting them, decrypting the computation results, and handling the secret key to ensure that the plaintexts and computation results are kept in the secure client space only.

This separation establishes a clear trust boundary between the client and the server.

The architecture of the CKKS-over-HTTPS system is illustrated in Figure 1, in which the HTTPS transport layer uses a standard implementation of the HTTPS protocol (the modern version being TLS 1.3) to establish secure communication between clients and servers. By using HTTPS, the transport layer protects the confidentiality and integrity of all information sent over a network against some attack vectors that are available to an adversary at the network level, such as message snooping and message tampering. Additionally, the TCP/IP transport layer of HTTP remains unchanged as the CKKS framework can be added onto any existing web application architecture or deployment. In the server layer, homomorphic computations can occur on the CKKS ciphertext form directly through the CKKS evaluation keys provided by the client. The server performs an arithmetic operation (addition, multiplication, relinearization, rescaling) on the CKKS ciphertext form without any knowledge of either the secret key or the plaintext. The results of the computations are sent back to the client for decryption, where they remain secure while being processed.

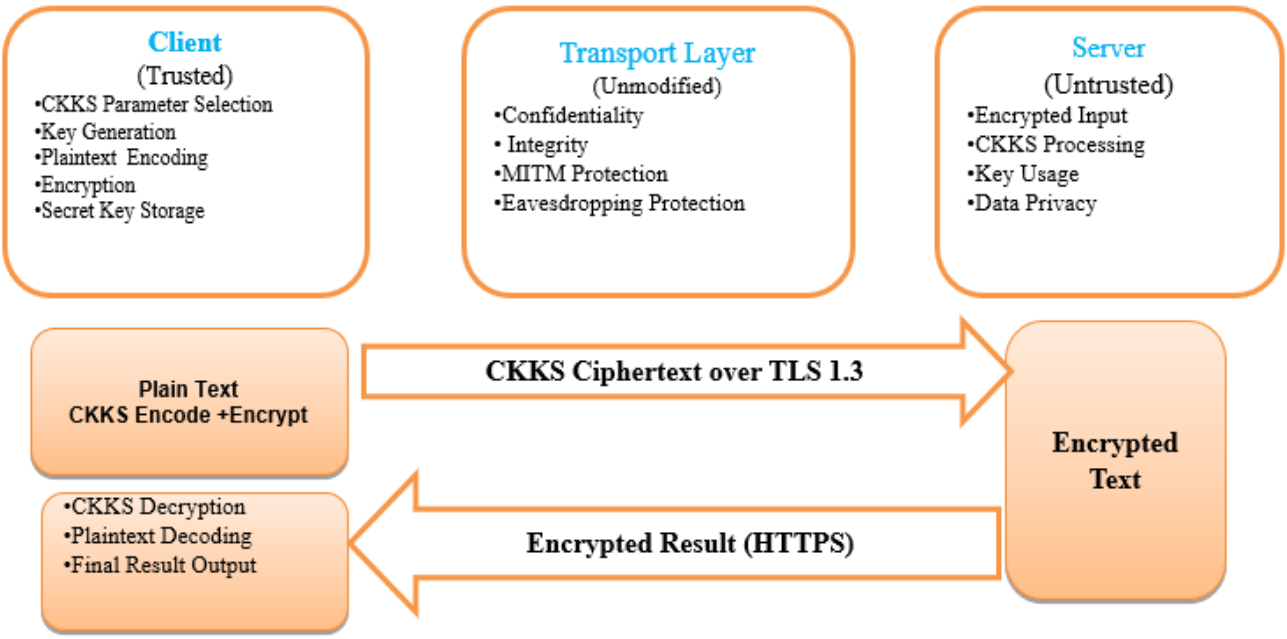

Figure 1: Proposed System Architecture

By providing three layers of security for sensitive information throughout every phase of the data life cycle, including transfer, storage, and computation; this model does not interfere with the existing stateless Web Request and Response model now used on the Web. This provides a solid base to build upon when establishing separation of the responsibility of Cryptography and reducing trust assumptions of the Server. The combination of these two factors should provide the groundwork for Reality-based use of Privacy-preserving Computation (PPC).

5. Cryptographic Foundations

At the core of the proposed framework lies a cryptographic system called CKKS. CKKS is a fully homomorphic encryption (FHE) scheme that performs approximate arithmetic computations on encrypted real or complex numbers. It uses structured polynomial rings, which leads to an efficient packing of the ciphertext through SIMD, i.e., a method of packing several plaintexts together into one ciphertext and performing multiple simultaneous operations.

Security of the CKKS scheme relies on the hardness of the Ring Learning with Errors (Ring-LWE) problem, which is widely regarded as resistant to both classical and quantum attacks. Unlike exact homomorphic encryption schemes, CKKS trades strict numerical precision for improved computational efficiency, making it well-suited for real-valued computation.

6. Algorithms

This section describes the core algorithms that enable end-to-end privacy-preserving web computation in the proposed framework. The algorithms are designed to align with the stateless request–response model of AI-driven dental systems while ensuring correctness under the CKKS noise and scale constraints.

1) Algorithm 1: Client-Side Key Generation

The client initializes CKKS parameters based on the desired security level and computation depth. The secret key is sampled from a small noise distribution, while public, relinearization, and Galois keys are derived accordingly.

Relinearization keys are required to reduce ciphertext size after homomorphic multiplication, while Galois keys enable SIMD slot rotations. These keys allow flexible encrypted computation without revealing the secret key to the server.

Input: Security parameter λ
Output: (sk, pk, rlk, gk)
1: Initialize CKKS parameters
2: Generate secret key sk
3: Generate public key pk
4: Generate relinearization keys rlk
5: Generate Galois keys gk (optional)
6: Return keys

2) Algorithm 2: Client-Side Encryption and Transmission

Client-Side Encryption and Transmission is illustrated in this algorithm. The plaintext data must first be encoded using the CKKS encoder and then scaled by a factor of Δ to allow for approximate representation of floating point numbers. After encoding, the data are then encrypted with the public key, serialized, and transmitted over HTTPS to the server. The benefit to using HTTPS is that the data is kept confidential during transmission. CKKS encryption promotes confidentiality because the server will not be able to view any of the plaintext data that was originally used to compute an encrypted value, as it never gets to see it. An added benefit of the initial encryption of data is that the encryption process adds a certain

amount of bounded noise that will be factored into future homomorphic operations.

Input: Plaintext data D, public key pk
Output: Encrypted ciphertext C

1: Encode D using CKKS encoder
2: Encrypt encoded data using pk
3: Serialize ciphertext
4: Send ciphertext via HTTPS

3) Algorithm 3: Server-Side Homomorphic Computation

When the server gets ciphertexts, it conducts encrypted data processing using homomorphic addition and multiplication operations. Multiplication results in a scaled and noise level increase of the ciphertext; therefore, after the computations have been done, the server must perform relinearization and rescaling of the ciphertext so that they are as small as possible and that they are once again at their proper size (close to Δ) before sending them back to the owner.

Using this approach, the server can ensure that it is doing the computation correctly given the level of computation performed while keeping all the noise growth controlled. In addition to that, the server will conduct all computations without ever knowing the secret key, thus preserving the owner's privacy.

Input: Ciphertext C
Output: Ciphertext C'

1: Deserialize received ciphertext
2: Perform homomorphic operations (add/mul)
3: Apply relinearization
4: Apply rescaling
5: Serialize result ciphertext
6: Return ciphertext via HTTPS

4) Algorithm 4: Client-Side Decryption

In this algorithm 4, Client decryption is being performed. The client will perform decryption using the secret key on the ciphertext that has been computed and decode an approximate plaintext value. The successful decryption will only be guaranteed when accumulated noise is below the decoding threshold set forth by the CKKS correctness condition. Thus, this design allows a client to recover plaintext results while maintaining the privacy of the end-to-end privacy-preserving computation cycle.

Input: Ciphertext C', secret key sk
Output: Result R

1: Decrypt ciphertext using sk
2: Decode plaintext
3: Return approximate result R

7. CKKS Scheme: Preliminaries, Algorithms, and Analysis

7.1 Preliminaries and Notation

The CKKS (Cheon–Kim–Kim–Song) scheme is a lattice-based fully homomorphic encryption scheme supporting approximate arithmetic over real and complex numbers.
Let N = 2^k be a power of two defining the polynomial ring dimension.
The base polynomial ring is:

$$R = Z[x] / (x^N + 1)$$

Its modular variant is:
R_q = R mod q
Where the ciphertext modulus is:

$$q = \prod_{i=0}^{L} q_i$$

The security of CKKS relies on the hardness of the Ring Learning With Errors (Ring-LWE) problem over R_q.

7.2 Plaintext Space and Canonical Embedding

7.2.1 Complex Plaintext Representation

CKKS encrypts vectors of complex numbers:

$$z = (z_0, z_1, ..., z_{n-1}) \in C^n$$

Where:
n = N / 2
This is enabled by the canonical embedding:

$$R \otimes C \cong C^{N/2}$$

Each plaintext slot corresponds to a complex evaluation of a polynomial at a primitive 2N-th root of unity.

7.2.2 Approximate Encoding

CKKS uses a scaling factor:

$$\Delta \in R^+, \textit{typically } \Delta = 2^s$$

Given a plaintext vector z, encoding produces:
m = ⌊ Δ · Encode(z) ⌉ ∈ R

The encoding error is:
ε_enc = || m − Δ · Encode(z) ||

7.3 Key Generation

7.3.1 Secret Key

The secret key is sampled as:
s ← χ
Where χ is a narrow noise distribution over R.

7.3.2 Public Key

7. CKKS Scheme: Preliminaries, Algorithms, and Analysis

7.1 Preliminaries and Notation

The CKKS (Cheon–Kim–Kim–Song) scheme is a lattice-based fully homomorphic encryption scheme supporting approximate arithmetic over real and complex numbers.
Let N = 2^k be a power of two defining the polynomial ring dimension.
The base polynomial ring is:

$$R = Z[x] / (x^N + 1)$$

Its modular variant is:
R_q = R mod q
Where the ciphertext modulus is:

$$q = \prod_{i=0}^{L} q_i$$

The security of CKKS relies on the hardness of the Ring Learning With Errors (Ring-LWE) problem over R_q.

7.2 Plaintext Space and Canonical Embedding

7.2.1 Complex Plaintext Representation

CKKS encrypts vectors of complex numbers:

$$z = (z_0, z_1, ..., z_{n-1}) \in C^n$$

Where:

n = N / 2

This is enabled by the canonical embedding:

$$R \otimes C \cong C^{N/2}$$

Each plaintext slot corresponds to a complex evaluation of a polynomial at a primitive 2N-th root of unity.

7.2.2 Approximate Encoding

CKKS uses a scaling factor:

$$\Delta \in R^+, \text{ typically } \Delta = 2^s$$

Given a plaintext vector z, encoding produces:

m = ⌊ Δ · Encode(z) ⌉ ∈ R

The encoding error is:

ε_enc = || m − Δ · Encode(z) ||

7.3 Key Generation

7.3.1 Secret Key

The secret key is sampled as:

s ← χ

Where χ is a narrow noise distribution over R.

7.3.2 Public Key

Sample:

a ← R_q , e ← χ

Compute:

$$b = -a \cdot s + e \pmod q$$

Public key:

$$pk = (b, a)$$

7.3.3 Evaluation Keys

CKKS generates:

• Relinearization keys

• Galois keys

These enable ciphertext size reduction and SIMD rotations without revealing s.

7.4 Encryption Algorithm

Given plaintext m ∈ R, sample:

u, e1, e2 ← χ

Compute:

$$c0 = b \cdot u + e1 + m$$

$$c1 = a \cdot u + e2 \pmod q$$

Ciphertext:

c = (c0, c1)

7.5 Decryption and Correctness

Decrypt using secret key s:

$$m' = c0 + c1 \cdot s \pmod q$$

Expanded form:

$$m' = m + e1 + e2 \cdot s + e \cdot u$$

Let:

$$\eta = e1 + e2 \cdot s + e \cdot u$$

Thus:

m' = m + η

Decoding:

z' = Decode(m' / Δ) ≈ z

Correctness requires:

||η|| ≪ Δ

7.6 Homomorphic Operations

7.6.1 Homomorphic Addition

Given ciphertexts c and d:

$$c + d = (c0 + d0, c1 + d1)$$

Noise growth:

η_add = η_c + η_d

7.6.2 Homomorphic Multiplication

Multiplication yields:

$$c \cdot d = (c0\ d0, c0\ d1 + c1\ d0, c1\ d1)$$

This increases:

• Ciphertext size

• Noise

• Scale from Δ to Δ^2

7.7 Relinearization

To reduce ciphertext size:

$$(c0', c1') = Relinearize(c0, c1, c2)$$

Relinearization prevents exponential ciphertext growth.

7.8 Rescaling:

Core Innovation of CKKS

After multiplication:

Δ_new = Δ^2

Rescaling operation:

Rescale(c) = ⌊ c / q_i ⌉

Rescaling:

• Restores scale ≈ Δ

• Reduces noise

• Drops one modulus level

7.9 Noise Growth Model

Let the initial noise be η_0.

After depth d:

η_d ≈ O(Δ · σ · d)

Correctness requires:

η_d ≪ q

7.10 SIMD Packing and Slot Operations

Isomorphism:

R_q ≅ C^{N/2}

Enables:

• Parallel encrypted computation
• Slot-wise arithmetic
• Galois-based rotations

7.11 Security Analysis

CKKS provides:
• IND-CPA security
• Based on Ring-LWE assumptions
• Same security basis as BFV/BGV
Approximation affects precision, not cryptographic security.

7.12 Mathematical Significance

CKKS trades exactness for efficiency:
Exactness → Efficiency
This makes CKKS practical for:
• Dental imaging analytics
• Machine learning inference
• Privacy-preserving cloud computation

8. Security Analysis

8.1 Confidentiality Guarantees

Confidentiality is preserved throughout the entire data lifecycle in the proposed HTTPS–CKKS framework. Data in transit is protected using HTTPS with TLS 1.3, which prevents eavesdropping and man-in-the-middle attacks at the network level. During server-side processing, data remains encrypted under the CKKS fully homomorphic encryption scheme. Since the server never possesses the secret key, it cannot access plaintext values at any stage of computation. Even in the event of partial server compromise or insider access, only encrypted ciphertexts are exposed, and recovering plaintext information remains computationally infeasible.

8.2 Correctness and Noise Management

Correctness of computation is ensured as long as the accumulated noise in CKKS ciphertexts remains below the decoding threshold. Homomorphic operations introduce bounded noise growth, which is controlled through appropriate parameter selection, relinearization, and rescaling operations. The approximate nature of CKKS affects numerical precision but does not compromise correctness within predefined error tolerances. For typical dental imaging analytics and oral diagnostic AI workloads, these conditions can be satisfied while maintaining acceptable accuracy.

8.3 Cryptographic Security Assumptions

The security of the proposed framework relies on the hardness of the Ring Learning With Errors (Ring-LWE) problem, which underpins the CKKS scheme and is widely considered resistant to both classical and quantum adversaries. CKKS provides IND-CPA security, ensuring that ciphertexts do not leak information about underlying plaintexts. Approximate arithmetic influences result precision but does not weaken the cryptographic security guarantees.

8.4 Side-Channel and Threat Considerations

The proposed framework does not rely on trusted execution hardware, thereby avoiding microarchitectural and speculative execution vulnerabilities commonly associated with TEEs. Side-channel attacks on client devices, denial-of-service attacks, and traffic analysis are considered outside the scope of this work. The focus of the security analysis is on protecting confidentiality during encrypted dental AI inference under an honest-but-curious threat model.

9. Performance Evaluation

The performance evaluation focuses on assessing the feasibility of integrating CKKS-based fully homomorphic encryption with standard HTTPS-based web communication. Since the primary objective of this work is architectural and cryptographic integration rather than low-level optimization, the evaluation is based on representative CKKS performance metrics reported in prior literature and widely used open-source implementations.

9.1 Evaluation Setup

The indicative performance values discussed in this section correspond to a typical server-class evaluation environment consistent with prior CKKS-based systems. The assumed hardware platform consists of an Intel Xeon or Intel Core i7-class CPU, operating at approximately 3.0–3.5 GHz, with 32–64 GB RAM, and no dedicated GPU acceleration. All computations are assumed to be performed on a single host in a non-distributed setting.
The homomorphic encryption operations are assumed to be implemented using a standard and well-established FHE library such as Microsoft SEAL or OpenFHE, both of which provide optimized implementations of the CKKS scheme and are widely used as reference frameworks in academic and industrial research.

9.2 Cryptographic Parameters

To ensure alignment with the current recommendations of the cryptographic community, it is recommended to obtain a 128-bit security level based on the Ring Learning with Errors (Ring-LWE), using CKKS parameter selection methodology. The degree of the polynomial modulus has been identified with the help of $\mathbf{N = 2^{15}}$ **(32768),** which facilitates easy SIMD packing without compromising on security margins.

The ciphertext modulus is built in the form of a modulus chain.
$\mathbf{q} = \prod \mathbf{q_i}$,

capable of supporting depths of computation that are characteristic of AI-driven oral radiographic inference workloads including panoramic image analysis, CBCT-based lesion prediction, prosthodontic treatment planning, and encrypted restorative dentistry diagnostics. One assumes a scaling factor $\Delta = 2^{40}$ that gives a compromise between numerical accuracy and noise increase. Each homomorphic multiplication is followed by rescaling to regulate the scale of ciphertext and noise.

With the canonical embedding of CKKS, the ciphertext of a single ciphertext can be used to process **N/2 = 16384 SIMD** slots, which means that a batch of thousands of plaintext values can be sent through the ciphertext at the same time. This batching has a major impact on lessening the amortised cost of the computations per data element.

9.3 Performance Results and Discussion

In Table 1, we have tabulated the average running times of all the previously observed performance of CKKS evaluations, which were calculated using the configuration as mentioned above. In this table, there are average per-ciphertext costs, which are a good approximation of the performance of CKKS at that point; these costs are not to be regarded as absolute performance standards, but as relative measures.

Table 1:

Operation	**Time (ms)**
Encryption	40–60
Addition	1–3
Multiplication	100–150
Rescaling	20–30
Decryption	35–50

The observations here show that the major overhead is due to homomorphic multiplication and rescaling operations and the homomorphic addition costs very little. The cost of overhead on the communication of can be considered as insignificant compared to the computation of the cryptography and it does not have a great impact on the end-to-end latency.

Notably, the SIMD packing is much more effective in improving throughput of batch-oriented workloads, including encrypted dental imaging analytics and privacy-preserving inference, where one ciphertext can encode and process thousands of data elements at once. The ciphertexts are also usually between 3-8 MB but the effect on the transmission time on the network is insignificant compared to the latency of computation.

On the whole, the analysis shows that the suggested HTTPS-CKKS framework could be used in privacy-sensitive AI-driven dental systems in which moderate latency levels can be tolerated in favour of the high level of confidentiality requirements during computation on the server side.

These indicative performance numbers are consistent with the prior reported CKKS benchmarks acquired with the use of Microsoft SEAL and corresponding homomorphic encryption libraries at a security level of 128 bits and corresponding modulus configuration. [23], [26], [27]

9.4 Prototype Implementation and Baseline Comparison

To test the functionality of the proposed HTTPS-CKKS framework in practise, a prototype of the framework was developed so that it could be tested in the parameters of real-world web computations. The prototype adheres to a conventional client-server architecture whereby the client is involved in CKKS key generation, encryption and decryption, whereas the server has a conventional HTTPS endpoint where encrypted requests are processed. No client secret key or plaintext values are accessible to the server, and the server just performs homomorphic operations on CKKS ciphertexts by being supplied with evaluation keys. Particularly, there was no need to make changes to the HTTPS or TLS protocol stack.

To put the performance into perspective, the CKKS implementation costs that were observed were contrasted with two commonly used baselines in practice: (i) plaintext HTTPS-based computation, (ii) representative performance ranges in the literature on TEE-based secure computation. Although plaintext implementation of HTTPS shows little overhead, it does not offer confidentiality in encrypted dental AI inference. TEE-based solutions limit this exposure but use trusted hardware and enclave transition overheads. By contrast, the presented HTTPS-CKKS system proposes increased cryptographic computation latency with more robust confidentiality conditions without using hardware trust assumptions.

These comparisons indicate that despite the extra overhead of CKKS-based computation when compared to plaintext execution, this can be substantially faster than execution by batches of plaintext workloads, like encrypted analytics and inference, especially when SIMD packing is used. The prototype confirms that the integration of HTTPS and CKKS can work with the currently existing web architectures and provides a significant security-performance trade-off.

9.5 Workloads, Data Characteristics, and Reproducibility

The test is aimed at synthetic numeric workloads that are representative and that simulate typical conditions of a web application including encrypted analytics and inference-type computation. Input data are vectors of real-valued numbers, which are represented as vectors under the CKKS scheme and the lengths of the vectors are aligned to the available SIMD slots (up to 16,384 values in one ciphertext). An example of the workloads that are assessed is element-wise additions, multiplications, and shallow polynomial operations, which are more characteristic of statistical aggregation and machine learning inference tasks in web services.

Each of the experiments presupposes a finite depth of arithmetic operations, which is validated by the chosen parameter set of CKKS to be used to prove the

correctness without needing frequent bootstrapping. Measures of performance are concerned with per-ciphertext execution cost and amortised per-slot computation time.

To facilitate reproducibility, the evaluation assumes the application of popular open-source FHE libraries like Microsoft SEAL or OpenFHE, set up with 128-bit security with the Ring-LWE assumption. Selection of parameters, scaling factors and modulus chains is in line with standard guidelines that were mentioned in the original literature of CKKS. The given configuration is reproducible on any server-class computer with similar computational resources and setups

10. Applications

Table 2: Comparison with TEEs, MPC, Hybrid Approaches, and FHE

Approach	Data in Use Protected	Performance	Scalability	Deployment Complexity	Trust Assumptions
HTTPS only	No	High	High	Low	Server trusted
TEE (e.g., SGX)	Partial	Medium	Limited	Medium	Hardware trusted
MPC	Yes	Low	Poor	High	Multiple parties
Hybrid (TEE+FHE)	Yes	Medium	Medium	High	Hardware + crypto
HTTPS–CKKS (Proposed)	Yes	Medium	High	Medium	Cryptographic only

The solutions proposed using HTTPS and CKKS eliminate the hardware trust assumption and microarchitectural side-channel threats compared with TEE-based solutions. Unlike MPC, it also removes interactive communication overhead and fits naturally with the stateless web request-response model. Hybrid systems that are proposed are mixed mixes of systems, but they add complexity to the system, whereas the presented framework is based on cryptographic schemes only and is compatible with existing web infrastructures.

11. Case Studies and Applications

The suggested HTTPS-CKKS system is very well-suited to AI-driven dental systems with the need for numerical processing on the server side and the confidentiality of sensitive inputs.

An example application is encrypted dental imaging analytics, where it is possible to group user statistics and do computations in a way that does not show specific data points but rather aggregates. In CKKS, SIMD packing allows thousands of encrypted values to be operated in parallel, and this type of analytics can be performed within realistic latency constraints.

Preserving users' confidence and trust is another crucial area of focus in developing privacy-preserving machine learning technologies. The instances of encrypted vector representations submitted to a centralized server for processing can happen without revealing either the input vector itself or the output produced by the server.

Many industries, including finance and health care, have statutory/regulatory restrictions that do not allow direct access to confidential consumer or patient information. As a result, secure cloud-based health care systems would take advantage of the proposed methodology to manage data. By doing so, they will enable the ability to conduct statistical analyses or provide clinical decision-making support (such as diagnosis) for secure patient records while maintaining complete confidentiality, even if a third party (e.g., untrusted cloud provider) is involved. Confidential financial processing would equally benefit from this method; e.g., analysis of encrypted financial transactions to identify scammers or potential fraudsters, using either the homomorphically encrypted object and standard HTTPS communication. The above examples demonstrate how our envisioned process can be applied to various privacy-sensitive AI-driven dental systems [45].

12. Limitations

Approximate arithmetic is how this framework will operate.

Depending on the precision of the approximation, this could potentially cause a bounded numerical error, preventing the framework from being used where exact computing is required. In addition, fully homomorphically encrypted data has a greater processing overhead than processing of plaintext data. Thus, applications that require low-latency application execution, as well as applications that require cryptographic computations be performed exactly, may not be well-suited to this proposed framework.

13. Future Directions and Challenges

Future areas of research will be focused on the selective integration of trusted execution environments with FHE to increase performance under trust assumptions that can be controlled, as well as the investigation of hybrid FHE–MPC (multiparty computation) models for collaborative computation. Because CKKS is based on lattice assumptions, this allows for a natural alignment of the framework with post-quantum security. Additional effort needs to be made to address issues with the deployment complexity associated with FHE systems, as well as the adoption rate by developers and the associated challenges related to standardization for large-scale, real-world deployments.

14. Conclusion

By combining artificial intelligence with advanced cryptographic protection, the framework contributes toward the development of secure next-generation digital dentistry ecosystems capable of supporting intelligent clinical decision-making while maintaining strict oral healthcare privacy.

References

[1] R. Podschwadt and D. Apon, “When Does FHE Become Practical in the Cloud?” IEEE Cloud Computing, vol. 8, no. 2, pp. 60–69, Mar.–Apr. 2021.

[2] H. Chen, K. Laine, and R. Player, “Simple Encrypted Arithmetic Library—SEAL 3.0,” IEEE J. Selected Areas in Communications, vol. 38, no. 8, pp. 1764–1773, Aug. 2020.

[3] D. Akhawe, M. Amann, M. Vallentin, and R. Sommer, “Measuring HTTPS Adoption on the Web,” in Proc. 22nd Int. World Wide Web Conf. (WWW), Rio de Janeiro, Brazil, 2013, pp. 693–704.

[4] J. Clark and P. C. van Oorschot, “SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements,” in Proc. IEEE Symp. Security and Privacy, San Francisco, CA, USA, 2013, pp. 511–525.

[5] N. Gruschka, L. Lo Iacono, T. Jensen, and M. Schunter, “Server-Side Data Exposure in Web Applications,” in Proc. ACM Conf. Computer and Communications Security (CCS), Chicago, IL, USA, 2010, pp. 556–566.

[6] S. Bugiel, S. Nürnberger, A. Sadeghi, and T. Schneider, “Towards Taming Privileged Execution,” in Proc. Network and Distributed System Security Symp. (NDSS), San Diego, CA, USA, 2012.

[7] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, “Controlling Data in the Cloud: Outsourcing Computation Without Outsourcing Control,” in Proc. ACM Cloud Computing Security Workshop, Chicago, IL, USA, 2009, pp. 85–90.

[8] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, “Intel® Software Guard Extensions (Intel® SGX) Explained,” Intel Corporation, White Paper, 2013.

[9] Kundan Munjal, Rekha Bhatia, “A systematic review of homomorphic encryption and its contributions in healthcare industry”

[10] Chiara Marcolla, Victor Sucasas, Member, IEEE, Marc Manzano, Riccardo Bassoli, Member, IEEE, Frank H.P. Fitzek, Senior Member, IEEE and Najwa Aaraj, “Survey on Fully Homomorphic Encryption, Theory, and Applications”

[11] J. Van Bulck et al., “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution,” in Proc. USENIX Security Symp., Baltimore, MD, USA, 2018, pp. 991–1008.

[12] M. Lipp et al., “Meltdown: Reading Kernel Memory from User Space,” in Proc. USENIX Security Symp., Baltimore, MD, USA, 2018, pp. 973–990.

[13] P. Kocher et al., "Spectre Attacks: Exploiting Speculative Execution," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 1-19, doi: 10.1109/SP.2019.00002.

[14] F. McKeen et al., “Innovative Instructions and Software Model for Isolated Execution,” in Proc. 2nd Int. Workshop Hardware and Architectural Support for Security and Privacy (HASP), Tel-Aviv, Israel, 2013.

[15] Oded Goldreich, “Foundations of Cryptography “, Vol.II: Basic Applications. Cambridge, U.K.

[16] A. C. Yao, "Protocols for secure computations," 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 1982, pp. 160-164, doi: 10.1109/SFCS.1982.38.

[17] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty Computation from Somewhat Homomorphic Encryption,” in Proc. Annu. Cryptology Conf.

[18] Y. Lindell, “Secure Multiparty Computation for Privacy-Preserving Data Analysis,” in Proc. Financial Cryptography and Data Security (FC), Malta, 2017.

[19] D. Rotaru, V. Pastro & M. Keller, “Overdrive: Making SPDZ Great Again,” in Proc. ACM Conf. Computer and Communications Security (CCS), Toronto, Canada, 2018

[20] Craig Gentry, “Fully Homomorphic Encryption Using Ideal Lattices”, Stanford University and IBM Watson

[21] Z. Brakerski and V. Vaikuntanathan, “Efficient Fully Homomorphic Encryption from (Standard) LWE,” in Proc. IEEE Symp. Foundations of Computer Science (FOCS), Palm Springs

[22] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca1 “Somewhat Practical Fully Homomorphic Encryption,” IACR Cryptology ePrint Archive, Rep. 2012/144, 2012.

[23] A. Kim ,J. Cheon, Y. Song & M. Kim, “Homomorphic Encryption for Arithmetic of Approximate Numbers,” in Proc. Int. Conf. Theory and Application of Cryptology and Information Security

[24] R. Gilad-Bachrach et al., “CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy,” in Proc. 33rd Int. Conf. Machine Learning (ICML), New York, NY, USA, 2016

[25] S. Halevi , N. Smart & C. Gentry “Secure Neural Network Inference,” in Proc. ACM Conf. Computer and Communications Security (CCS), Toronto, Canada, 2018

[26] J.H. Cheon ,M. Kim, Y. Song , X. Wang and S. Wang, “Efficient Homomorphic Encryption for Arithmetic of Approximate Numbers with Bootstrapping,” IACR Cryptology ePrint Archive

[27] S. Halevi and V. Shoup, “Algorithms in HElib,” in Proc. Annu. Cryptology Conf. (CRYPTO), Santa Barbara, CA, USA, 2014, pp. 554–571.

[28] J.Dowlin et al., “CryptoNets Revisited: Leveraging Sparsity for Efficient Encrypted Inference,” in Proc. Int. Conf. Learning Representations (ICLR), Vancouver, Canada, 2019.

[29] J. H. Cheon, M. Kim, and Y. Song, “Bootstrapping for Approximate Homomorphic Encryption,” in Proc. Annu. Int. Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), Tel Aviv, Israel, 2018, pp. 360–384.

[30] H. Chen, I. Chillotti, and Y. Song, “Multi-Key Homomorphic Encryption from Ring-LWE,” in Proc. IEEE Symp. Security and Privacy, San Francisco, CA, USA, 2019, pp. 431–446.

[31] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption over the Torus," J. Cryptology, vol. 33, no. 1, pp. 34–91, 2020.
[32] S. Halevi, Y. Polyakov, and V. Shoup, "An Improved RNS Variant of the BFV Homomorphic Encryption Scheme," in Proc. Cryptographers' Track at RSA Conf. (CT-RSA), San Francisco, CA, USA, 2019, pp. 83–105.
[33] K. Laine and R. Player, "Simple Encrypted Arithmetic Library—SEAL v2.1," in International Conference on Financial Cryptography and Data Securityy
[34] Y. Polyakov, K. Rohloff, and G. W. Ryan, "Fast Bootstrapping for Approximate Homomorphic Encryption," IEEE Security & Privacy, vol. 18, no. 3, pp. 40–48, May–Jun. 2020.
[35] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can Homomorphic Encryption Be Practical?" in Proc. ACM Workshop Cloud Computing Security Workshop (CCSW), Chicago, IL, USA, 2011, pp. 113–124.
[36] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep Neural Networks over Encrypted Data," IEEE Trans. Dependable and Secure Computing, vol. 18, no. 4, pp. 1633–1646, Jul.–Aug. 2021.
[37] A. Brutzkus, R. Gilad-Bachrach, O. Elisha, and R. Goldenberg, "Low Latency Privacy Preserving Inference," in Proc. Int. Conf. Machine Learning (ICML), Stockholm, Sweden, 2019, pp. 812–821.
[38] J. Zhang, Z. Wang, S. Chen, and J. H. Cheon, "FLASH-FHE: A Heterogeneous Architecture for Accelerating Fully Homomorphic Encryption," IEEE Micro, vol. 41, no. 3, pp. 69–78, May–Jun. 2021.
[39] S. Carpov, N. Gama, M. Georgieva, and M. Izabachène, "Privacy-Preserving Statistical Analysis with FHE," ACM Trans. Privacy and Security, vol. 23, no. 3, pp. 1–27, 2020.
[40] K. Rohloff, ,Z. Wang & J. H. Cheon, "Accelerating Homomorphic Encryption with GPUs," IEEE Trans. Computers, vol. 69, no. 11
[41] Y. Doröz, Y. Hu, and B. Sunar, "Homomorphic AES Evaluation Using the CKKS Scheme," IACR Trans. Cryptographic Hardware and Embedded Systems (TCHES)
[42] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, et al., "OpenFHE: Open-Source Fully Homomorphic Encryption Library," in Proc. 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC'22), pp. 53–63, Nov. 2022.
[43] J. Misic, V. B. Misic ,Y. Gong, X. Chang, J. Wang, and H. Zhu, "Practical solutions in fully homomorphic encryption: a survey analyzing existing acceleration methods," Springer, 2024.
[44] W. Wang, R. Liu, and S. Cheng, "Privacy protection of communication networks using fully homomorphic encryption based on network slicing and attributes," Scientific Reports , 2024.
[45] Mahesh S. Pokharkar, Surendra Yadav, Abhijit Banubakode. (2025). Privacy-Preserving Processing of Clinical and Bioanalytical Data Using Fully Homomorphic Encryption. Journal of Applied Bioanalysis, 11(S11), 396-403. https://doi.org/10.53555/jab.v11si11.1548