An Agile Homomorphic Encryption and ML-KEM Scheme for Privacy-Preserving Health Data Processing

(Extended Version – Lightweight Cryptography for Security and Privacy 2026)*

Daniel Resende[1] ✉[0009-0005-6382-8731], Nadia El-Mrabet[1][0000-0003-3840-584X], and Ulrich Kühne[2][0000-0002-0855-8223]

[1] Mines Saint-Etienne, CEA-Leti, Centre CMP, Equipe commune SAS, F-13541, Gardanne, France
{daniel.resende, nadia.el-mrabet}@emse.fr
[2] Télécom Paris, Institut polytechnique de Paris, Palaiseau, France
ulrich.kuhne@telecom-paris.fr

Abstract. The storage and processing of health data raise serious privacy concerns; therefore, the data must be encrypted. Fully Homomorphic Encryption (FHE) allows computations on encrypted data, which makes it a very promising technology in this context. However, although the efficiency of FHE algorithms has improved in recent years, they remain very complex. The security of several prominent FHE schemes is based on variants of the Learning With Errors (LWE) problem, which is also the basis of Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM), the new post-quantum standard for asymmetric encryption. In this paper, we propose an agile cryptographic scheme – combining ML-KEM and BGV – that allows efficient implementations in an embedded use case. We describe the parameter selection for the proposed scheme and show its feasibility with a proof-of-concept implementation.

Keywords: Post-Quantum Cryptography · Homomorphic Encryption · BGV · ML-KEM · Crypto-Agility.

1 Introduction

The use of digital health data offers new opportunities for medical research and improved care, but it also poses an increasing risk to the privacy of the concerned patients. News about data breaches of sensitive health data has become common. The number of reported incidents affecting at least 500 patients in the US has grown from around 200 in 2010 to more than 700 in 2024 [3]. Even worse, almost 70 % of the US population was affected by these breaches in 2024. In order to manage vast amounts of data, hospitals and other health providers are using cloud storage as a scalable solution. While this offers clear advantages regarding availability and efficiency, it also exposes the stored data to attacks [16].

* Please cite the workshop version of this paper [43], published at LigthSec 2026.

Another source of health data are portable embedded devices, ranging from consumer electronics such as fitness trackers or smart watches to specialized medical devices such as heart rate monitors or implanted glucose trackers, some of which are connected to the internet in order to report or share the collected data. The possible consequences of carelessly sharing such data have been reported e.g. in [48], when the position of secret military bases was revealed by GPS data from soldiers' fitness trackers. At the same time, more and more countries – for example in Europe [20] – are transitioning to the official use of Electronic Health Records (EHRs). As the study in [25] reveals, the large scale adoption of such systems by patients critically depends on their *trustworthiness*. This shows that security is crucial in any application that handles health data, and that storing health records in plain text should be avoided in the first place. The proper use of cryptography is paramount.

Although there are many robust cryptographic algorithms, their use does not come without friction. We are currently in a transitional phase, where some widely adopted cryptographic standards are being deprecated due to the potential threat of quantum computers. While new (post-quantum) algorithms have already been standardized – such as the ML-KEM [35] – they are more complex than the ones they replace, and robust and efficient implementations are currently under development.

Furthermore, in a typical healthcare scenario, there can be multiple parties that should have legitimate access to the recorded data, including doctors and medical staff, pharmacies, research institutions, and the patient herself. This makes the use of symmetric cryptography difficult, since the secret key needs to be distributed in a secure way. Even when the data is stored encrypted, it needs to be temporarily decrypted for any subsequent processing or aggregation, exposing it to potential attackers.

In this context, Homomorphic Encryption (HE) is a very attractive solution. It allows computations to be performed on encrypted data *without decrypting* it. This offers the possibility to do computations on a cloud server, while maintaining full control and ownership over the data, and without needing to trust the server. The great potential of HE in health applications has been explored e.g. in [32,4]. A drawback of current HE schemes is the computational complexity of the cryptographic operations. While the homomorphic computations can be carried out on a powerful cloud server, at least encryption and decryption must take place on the user's end device. In the scheme proposed in [4], the key generation is taken care of by a dedicated server. This limits the user's trust in the scheme. It has yet to be shown that HE can be used in a realistic embedded setting, where a low power device has full control over both key generation and encryption.

In this paper, we propose an agile cryptographic scheme for exactly this application scenario. In particular, we leverage the fact that several interesting FHE algorithms are based on the same – or a very similar – mathematical framework as ML-KEM, namely LWE. With this observation in mind, we select algorithmic parameters in such a way as to maximize the synergies in the implementation of both ML-KEM and FHE, in order to reduce the code size of the cryptographic implementation. The contributions of this paper are the following:

- An embedded use case scenario for an agile cryptographic scheme.

- A systematic review of existing FHE algorithms and their suitability for the use case.
- The definition and parameterization of a cryptographic scheme combining ML-KEM and FHE.
- A proof of concept implementation for the above scheme.

The remainder of this paper is organized as follows: We will start with a presentation of the use case scenario in Sect. 2. The necessary background for the used cryptographic algorithms is given in Sect. 3. Based on these definitions, we select a suitable algorithm and its parameters in Sect. 4. Possible optimizations are discussed in Sect. 5. Finally, a proof-of-concept implementation is presented in Sect. 6 before concluding the paper in Sect. 7.

2 Use Case: Average Heart Rate Computation

The objective is to protect medical research and data analysis in healthcare monitoring. More specifically, this involves ensuring the confidentiality and security of data transmission and storage, particularly regarding personal health records stored on wearable devices, also called EHRs.

2.1 System Overview

We chose this use case due to the abundance of available studies and the ease of access to the required sensors, which are integrated into widely available connected devices (such as smartwatches and smartphones) or medical devices (like blood pressure monitors). A concrete example would be an athlete who wants to track their health on a mobile app and a smartwatch, with direct access to this data for a healthcare professional.

As depicted in Fig. 1, the concept is to leverage health data (discussed in Subs. 2.2) from an EHR, specifically a smartwatch, for processing or statistical analysis on a distant server or cloud. This ensures the protection and security of the data throughout the server-side processing. Once the data has been processed, the results are securely transmitted, either to the patient or directly to the healthcare provider, such as a doctor or hospital.

Given the constraints of the Internet of Things (IoT) environment (limited memory, reduced surface area), we favor a centralized approach: data is stored and encrypted on the server, accessible for statistical analysis at any time, without requiring processing on the devices.

To achieve this, we propose utilizing homomorphic encryption, as depicted in Fig. 1, to encrypt sensitive medical information.

Furthermore, to enable communication between different equipment (such as the EHR, smartphones, and servers), it is essential to establish and exchange cryptographic keys, including homomorphic keys, to ensure the integrity and protection of these critical assets.

The goal is to consolidate cryptographic functions into a single *crypto-core* that combines key exchange, homomorphic encryption and more traditional symmetric encryption[3] (such as Ascon [50]) for message exchange (for data that do not require confidentiality at the server level, because, as we will see in Subs. 3.2, it is more costly).

[3] We will not address this point of symmetric encryption in this article.

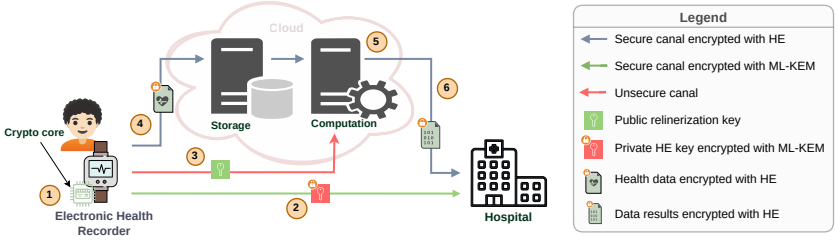


Fig. 1: Use case schema.

This data contains information about the electrical activity of the patient's heart, recorded over 24 hours. In order to demonstrate the feasibility of our concept, we selected average heart rate calculation as our case study. To compute the average heart rate we use information extracted from the annotation file of the ECG. The annotation file provides information related to cardiac contraction for each beat and the temporal distance (toc) between consecutive beats. The toc values between consecutive beats will be used to calculate the average heart rate.

Process flow described in Fig. 1:

1. Key Generation: The crypto-core will generate the homomorphic key pair.
2. Key Exchange (Optional): We exchange the private homomorphic key, using ML-KEM encryption, to allow a third party (such as a doctor or hospital) to decrypt the results returned by the server[4].
3. Key Exchange: Moreover, the relinearization key is sent to the server, as it is necessary after a multiplication.
4. Encryption: Data generated by the device (e.g., a smartwatch) is encrypted and sent to the server. The data is stored encrypted in the database.
5. Computation: When a user or third party makes a statistical request, the server performs arithmetic operations (such as addition or multiplication) on the encrypted data, resulting in an encrypted output.
6. Decryption: The server sends the encrypted results back to the user or third party, who can decrypt them using their private homomorphic key.

2.2 Analyzed Data

Kocabas et al. [28] describe the computation of the average heart rate using a sample Electrocardiogram (ECG) annotation file, such as the database provided by [15]. In our case, we only focus on descriptive statistical functions such as summation, mean, variance, or standard deviation. Given a signal sampling frequency of f_s = 1,000 Hz, each value toc_i corresponds to the time interval between two successive heartbeats, measured in units of samples. For N beats, the values are denoted as $toc_1, toc_2, \ldots, toc_N$.

1. **Sum of Intervals:** $S = \sum_{i=1}^{N} toc_i$.

[4] The keys of the KEM can be managed in different ways; in particular, one should avoid a centralized PKI operated by the same provider as the computing server.

2. **Mean RR Interval (in seconds):** The RR intervals represent the time between each detected heartbeat, measured from peak (R) to peak (R).

$$\overline{RR} = \frac{S}{N} \times \frac{1}{f_s} = \frac{S}{N} \times 0.001.$$

3. **Heart Rate (in beats per minute, bpm):**

$$\mathtt{HR}_{\text{bpm}} = \frac{60}{\overline{RR}} = 60 \times \frac{f_s \cdot N}{S} = 60{,}000 \times \frac{N}{S}.$$

Since N is a public value, we only need homomorphic operations to compute S; after decryption, the heart rate is obtained as $60{,}000 \times N/S$.

4. **Standard Deviation of NN Intervals (SDNN):** The same idea applies to SDNN, the overall standard deviation of the Normal-to-Normal (NN) intervals between consecutive normal beats over a given period [47]. It is defined as:

$$\mathtt{SDNN} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}((toc_i - toc_{i+1}) - \overline{RR})^2}.$$

We just need to perform homomorphic operations on the sum. For each encrypted toc_i, toc_{i+1}, there is a multiplicative depth of one (due to squaring).

3 Background

In this section, we define the cryptographic notions of Key-Encapsulation Mechanism (KEM) and HE, which are used in our use case. We have opted for the latest Post-Quantum Cryptography (PQC) National Institute of Standards and Technology (NIST) standard for key exchange, ML-KEM. Recent HE schemes and ML-KEM are based on variants of the LWE problem.

Definition 1 (LWE Distribution). *Let $n \geq 1$, q a prime, χ_e an error distribution on $\mathbb{Z}_q^n$, and $s \in \mathbb{Z}_q^n$. The **LWE distribution** A_{s,χ_e} on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is defined as:*

$$A_{s,\chi_e} = \left\{ (a,\ a \cdot s + e \bmod q) \mid a \xleftarrow{\$} \mathbb{Z}_q^n,\ e \leftarrow \chi_e \right\}.$$

This gives rise to two hard problems:

- ***Search LWE:** Given m samples from A_{s,χ_e}, recover s.*
- ***Decision LWE:** Given m samples from either A_{s,χ_e} (with $s \xleftarrow{\$} \mathbb{Z}_q^n$) or $\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$, no PPT adversary $\mathcal{A}$ can distinguish them with non-negligible advantage:*

$$\left|\Pr\left[\mathcal{A}(A_{s,\chi_e}^m) = 1\right] - \Pr[\mathcal{A}(\mathcal{U}^m) = 1]\right| \leq \varepsilon(n).$$

Both variants are conjectured to be equivalent [42].

Definition 2 (Ring LWE (R-LWE) Distribution). *Let $n \geq 1$ be a power of two, q a prime, $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, χ_e an error distribution on R_q, and $s \in R_q$. The* ***R-LWE distribution*** *A_{s,χ_e} on $R_q \times R_q$ is defined as:*

$$A_{s,\chi_e} = \left\{ (a,\ a \cdot s + e \bmod q) \mid a \xleftarrow{\$} R_q,\ e \leftarrow \chi_e \right\}.$$

Given m samples from either A_{s,χ_e} (with $s \xleftarrow{\$} R_q$) or $\mathcal{U}(R_q \times R_q)$, no PPT adversary $\mathcal{A}$ can distinguish them with non-negligible advantage:

$$\left|\Pr\left[\mathcal{A}(A^m_{s,\chi_e}) = 1\right] - \Pr[\mathcal{A}(\mathcal{U}^m) = 1]\right| \leq \varepsilon(n).$$

3.1 ML-KEM

ML-KEM is a quantum-resistant KEM whose security reduces to the Module LWE (M-LWE) problem, a generalization of R-LWE, itself a variant of LWE, which was introduced by Regev in 2005 [41] and is reducible to standard lattice problems. It consists of three steps: key generation, encryption, and decryption, parametrized by a prime modulus q, a dimension n over $\mathcal{R}_q^k = \left(\mathbb{Z}_q[X]/(X^n+1)\right)^k$, noise parameters η_1, η_2, and $k \in \{2, 3, 4\}$ for the desired security level (cf. Tab. 1). We refer the reader to [35] for further details.

Table 1: Parameter sets for ML-KEM

Parameter set	NIST Security Level	n	q	k	η_1	η_2	(d_u, d_v)
ML-KEM-512	1	256	3329	2	3	2	(10, 4)
ML-KEM-768	3	256	3329	3	2	2	(10, 4)
ML-KEM-1024	5	256	3329	4	2	2	(11, 5)

Key Generation. A private key $s \in \mathcal{R}_q^k$ and a secret error $e \in \mathcal{R}_q^k$ are sampled from binomial distributions. The public key (A, t) is calculated by the operation $t := A \cdot s + e \in \mathcal{R}_q^k$ where the matrix $A \in \mathcal{R}_q^{k \times k}$ is sampled uniformly at random.

Encryption. Given a message $m \in \{0,1\}^{256}$, we sample from binomial distributions a random vector $r \in \mathcal{R}_q^k$ and two errors $e_1 \in \mathcal{R}_q^k$ and $e_2 \in \mathcal{R}_q$. The ciphertext (u, v) is computed as follows:

$$u := A^T \cdot r + e_1 \in \mathcal{R}_q^k, \quad v := t^T \cdot r + e_2 + \mathsf{Decompress}_q(m, 1) \in \mathcal{R}_q \tag{1}$$

Decryption. Let (u, v) be a ciphertext and s the private key. To decrypt the message, we compute the operation $m := \mathsf{Compress}_q\left(v - s^T \cdot u, 1\right)$.

Compression/Decompression. The functions $\mathsf{Compress}_q(x,\ d) = \left\lfloor \frac{2^d}{q} \cdot x \right\rceil \bmod 2^d$ and $\mathsf{Decompress}_q(x,\ d) = \left\lfloor \frac{q}{2^d} \cdot x \right\rceil$ perform a compression (resp. decompression), reducing the representation of an element of $\mathbb{Z}_q$ to $d < \lceil \log_2 q \rceil$ bits in order to minimize the size of the transmitted ciphertexts [35].

3.2 Homomorphic Encryption

In 1978, Rivest, Adleman and Dertouzos [44] described the first HE scheme, based on the RSA problem, which supports homomorphic multiplication. The goal of HE is to evaluate (or compute) on encrypted data, for instance on a cloud server, without decrypting it. This ensures data confidentiality compared to traditional encryption, which requires decryption before computation. Subsequently, some proposals for partial HE (i.e. for homomorphic addition or multiplication) have been made using the problems of ElGamal [19], Diffie–Hellman [17] and Paillier [37]. Afterwards, in 2009, Gentry [23] introduced the first FHE scheme, which allows both homomorphic operations to be computed and is based on lattice problems.

Definition 3 (Homomorphic Encryption). *A cryptosystem* $(\mathsf{Enc}, \mathsf{Dec})$ *over a plaintext space* $\mathcal{M}$ *and a ciphertext space* C *is* homomorphic *if there exist binary operations* $\circ$ *on* $\mathcal{M}$ *and* $\star$ *on* C *such that*

$$\mathsf{Dec}(\mathsf{Enc}(m_1) \star \mathsf{Enc}(m_2)) = m_1 \circ m_2, \qquad \forall\, m_1, m_2 \in \mathcal{M}.$$

In particular:

- ***Additive homomorphism:*** $\mathsf{Dec}(\mathsf{Enc}(m_1) + \mathsf{Enc}(m_2)) = m_1 + m_2$.
- ***Multiplicative homomorphism:*** $\mathsf{Dec}(\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2)) = m_1 \cdot m_2$.

Definition 4 (Fully Homomorphic Encryption). *A FHE scheme is a quadruple of algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *defined over a plaintext space* $\mathcal{M}$*, a ciphertext space* C*, and a class of admissible functions* $\mathcal{F}$*. It consists of the following algorithms:*

- $\mathsf{KeyGen}(\lambda) \to (\mathsf{pk}, \mathsf{sk})$*: takes as input a security parameter* $\lambda \in \mathbb{N}$ *(written in unary) and outputs a public key* pk *and a secret key* sk*.*
- $\mathsf{Enc}_{\mathsf{pk}}(m) \to c$*: takes a plaintext* $m \in \mathcal{M}$ *and outputs a ciphertext* $c \in C$*.*
- $\mathsf{Dec}_{\mathsf{sk}}(c) \to m$*: takes a ciphertext* $c \in C$ *and outputs a plaintext* $m \in \mathcal{M}$*.*
- $\mathsf{Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_n) \to c_f$*: takes a function* $f \in \mathcal{F}$ *and ciphertexts* $c_1, \ldots, c_n \in C$*, and outputs a ciphertext* $c_f \in C$*.*

The scheme is said to be correct *if, for all* $f \in \mathcal{F}$*, all* $n \in \mathbb{N}$*, and all* $m_1, \ldots, m_n \in \mathcal{M}$*,*

$$\mathsf{Dec}_{\mathsf{sk}}\big(\mathsf{Eval}_{\mathsf{pk}}\big(f, \mathsf{Enc}_{\mathsf{pk}}(m_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(m_n)\big)\big) = f(m_1, \ldots, m_n).$$

4 Scheme Selection

In this section, we detail the selection of the HE scheme that most closely resembles ML-KEM. The candidate schemes Brakerski-Fan-Vercauteren (BFV), Brakerski-Gentry-Vaikuntanathan (BGV), Fastest Homomorphic Encryption in the West (FHEW), Fast Fully Homomorphic Encryption over the Torus (TFHE), and Cheon-Kim-Kim-Song (CKKS) are all included in popular homomorphic libraries such as SEAL [46], OpenFHE [5], HElib [24], or HEAAN [11]. To guide our selection, we compare their arithmetic structures in Tab. 2, focusing on three criteria: **(1)** plaintext space compatibility with ML-KEM, **(2)** ciphertext space structure, and **(3)** hardware implementation cost.

Table 2: Comparison of arithmetic aspects of cryptographic schemes.

Schemes	Plaintext	Ciphertext	Security Assumption
ML-KEM [35]	$\mathbb{Z}_q$	$\mathbb{Z}_q[X]/(X^N+1)^k$	Module-LWE
ML-DSA [34]	$\mathbb{Z}_q$	$\mathbb{Z}_q[X]/(X^N+1)^k$	Module-LWE
BFV [21]	$\mathbb{Z}_t$	$\mathbb{Z}_q[X]/(X^N+1)$	Ring-LWE
BGV [10]	$\mathbb{Z}_t$	$\mathbb{Z}_q[X]/(X^N+1)$	Ring-LWE
FHEW [18]	$\mathbb{F}_2$	Torus $\mathbb{T}$	LWE over Torus
CKKS [11]	$\mathbb{C}$ (approx.)	$\mathbb{Z}_q[X]/(X^N+1)$	Ring-LWE
TFHE [12]	$\mathbb{F}_2$	Torus $\mathbb{T}$	LWE over Torus

4.1 FHEW and TFHE – Binary Arithmetic

FHEW and TFHE operate over a binary plaintext space ($\mathbb{F}_2$) and a Torus ciphertext space, which is fundamentally incompatible with ML-KEM's integer arithmetic over $\mathbb{Z}_q$. Furthermore, supporting both binary and prime-field characteristics in hardware requires separate datapaths. As shown by Ras et al. [40], a unified butterfly architecture for Number Theoretic Transform (NTT) and Frobenius Additive FFT (FAFFT) incurs a significant increase in silicon area. For these reasons, FHEW and TFHE are excluded from consideration.

4.2 CKKS – Approximate Arithmetic

CKKS operates over complex numbers $\mathbb{C}$ with approximate decryption, making it well-suited for real-valued machine learning workloads but incompatible with the exact integer semantics required by ML-KEM. The encoding and decoding overhead between $\mathbb{C}$ and $R_q = \mathbb{Z}_q[X]/(X^N+1)$ introduces additional complexity without benefit in our context of average heart rate computation (see Sect. 2).

4.3 BFV vs. BGV

Both BFV and BGV share the same plaintext space $\mathbb{Z}_t$ and ciphertext space $R_q = \mathbb{Z}_q[X]/(X^N+1)$, and their R-LWE security assumption directly mirrors that of ML-KEM. The conversion from plaintext R_t to ciphertext R_q reduces to a simple modulus switch. Kim et al. [27] show that the two schemes are largely equivalent, but BGV outperforms BFV for small t moduli and offers finer granularity through its leveled variant. We therefore select **BGV** for the remainder of this work.

5 Description of BGV Optimization

5.1 Leveled Brakerski-Gentry-Vaikuntanathan

In this subsection, we will recall the definition of the leveled version of BGV [10]. We denote by t and q respectively the plaintext and the ciphertext modulus. Moreover, as described by Mono et al. [31], we set $t \equiv 1 \pmod{N}$ and q a chain of prime numbers,

such that $q = q_{L-1} = \prod_{j=0}^{L-1} q_j$, where q_i are roughly of the same size and $p_j \equiv 1 \pmod{N}$. The number of primes, L, in the circuit is determined by its multiplicative depth, M. Therefore, at any level ℓ, we can express $q_\ell = \prod_{j=0}^{\ell} q_j$.

Key Generation. We sample a ternary secret $s \leftarrow \chi_s$, a polynomial $a \leftarrow \mathcal{U}_{q_L}$ uniformly at random, and a Gaussian error $e \leftarrow \chi_e$. Define the public key pk and the secret key sk as $pk = (b, a) = (-a \cdot s + t \cdot e, a)$ and $sk = s \pmod{q_L}$.

Encryption. To encrypt a message $m \in \mathcal{R}_t$ under the public key $pk = (b, a)$, we generate a ternary sample $u \leftarrow \chi_s$ and two Gaussian errors $e_0, e_1 \leftarrow \chi_e$. We obtain a ciphertext $c = (c_0, c_1) = (b \cdot u + t \cdot e_0 + m, a \cdot u + t \cdot e_1) \pmod{q_L}$.

Decryption. Given a ciphertext $c = (c_0, c_1)$ and a secret key $sk = s$, we compute the decrypted message as $m' = (c_0 + c_1 \cdot s \pmod{q_\ell}) \pmod{t}$, which recovers the message m provided the error is small enough.

Homomorphic operations. Let $\alpha \in R_t$ be a constant polynomial and $c = (c_0, c_1)$, $c' = (c'_0, c'_1)$ two ciphertexts at the same level ℓ. We define:

$$\mathbf{Add}(c, c') = (c_0 + c'_0,\ c_1 + c'_1) \pmod{q_\ell} \tag{2}$$

$$\mathbf{MulConst}(\alpha, c) = (\alpha c_0,\ \alpha c_1) \pmod{q_\ell} \tag{3}$$

$$\mathbf{Mul}(c, c') = (c_0 c'_0,\ c_0 c'_1 + c_1 c'_0,\ c_1 c'_1) \pmod{q_\ell} \tag{4}$$

where **Mul** outputs a degree-3 tuple requiring a *relinearization* (key-switching) step to return to a standard two-component ciphertext.

Modulus Switching. Let $c = (c_0, c_1)$ be a ciphertext at level ℓ, defined modulo q_ℓ. Let $q_{\ell-1} < q_\ell$ be the next modulus in the modulus chain, we define the modulus switching operation as follows:

$$\mathbf{ModSwitch}(c) = \left(\left\lfloor \frac{q_{\ell-1}}{q_\ell} \cdot c_0 \right\rceil, \left\lfloor \frac{q_{\ell-1}}{q_\ell} \cdot c_1 \right\rceil \right)$$

This operation preserves the encrypted message while reducing the modulus from q_ℓ to $q_{\ell-1}$. Moreover, it proportionally reduces the noise, up to a small additional rounding error introduced during the scaling step.

Critical Quantity. Since noise accumulates with each homomorphic operation, it is important to track the noise level: beyond a certain limit, decryption fails because of excessive noise. To this end, Mono et al. define in [31] the notion of *critical quantity* (or noise budget). Let $\mathbf{c} = (c, \ell, \nu)$ be an extended ciphertext in the BGV scheme, where $c = (c_0, c_1)$ is a ciphertext pair, ℓ denotes the current level, and q_ℓ is the associated ciphertext modulus. The *critical quantity* of c is the polynomial $\nu = \left[c_0 + c_1 \cdot s\right]_{q_\ell}$, obtained by evaluating the ciphertext as a polynomial in the secret key s. Decryption is correct if and only if $\|\nu\|_\infty \leq c_m \|\nu\|_{\text{can}} < \frac{q_\ell}{2}$, where c_m is the ring expansion factor and $\|\cdot\|_{\text{can}}$ denotes the canonical embedding norm. The critical quantity grows with each homomorphic operation; its bounds for the main operations are given below.

Circuit. In a leveled FHE scheme, an *arithmetic circuit* is a graph whose nodes represent homomorphic operations (additions, multiplications, Boolean operations) applied to ciphertexts; the determining parameter is its multiplicative depth L, defined as the maximum number of multiplications on any path from input to output.

5.2 Crypto-Agility: BGV and ML-KEM

In this subsection, we explain how key generation, encryption and decryption can be shared between BGV and ML-KEM. It is important to note that our goal is not to optimize the homomorphic operations defined in Subs. 5.1, as they are executed in the server component of the HE protocol. Nevertheless, they still have an influence on the selection of parameters described in Subs. 5.4. As discussed in Sect. 4, both schemes operate on the same type of arithmetic.

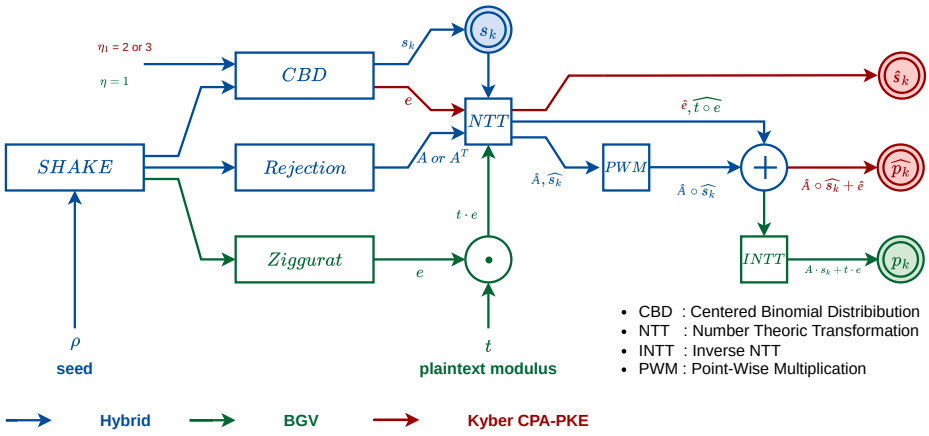


Fig. 2: Hybrid key generation for BGV/ML-KEM.

Remark 1. Here, we use the CPA-PKE version of ML-KEM [35]. For the CPA-KEM version, we can reuse the optimizations from the CPA-PKE version, since the keys are encapsulated. For the sake of simplicity, Fig. 3 is drawn for the case of ML-KEM. We note that the encrypted pair $(c_0 = A^T \cdot r + e_1, c_1 = t^T \cdot r + e_2 + m)$ where $m = \texttt{Decompress}_q(M, 1)$. In addition, we have omitted the $\texttt{Compress}_q$ and $\texttt{Decompress}_q$ stages from the diagram, since they are not part of BGV. These operations can be managed independently and concurrently. The same approach is applied to decryption in Fig. 4.

We observe that BGV and ML-KEM share many similarities for key generation (see Fig. 2) and encryption (see Fig. 3). Both algorithms involve polynomial multiplication using the NTT and random number generation. However, the main distinction is that, in ML-KEM, the keys remain in the NTT domain. Fig. 4 shows that both cases share identical functions. However, like in encryption, the key must be passed in order to perform calculations in the NTT domain.

Remark 2. Although the ML-KEM standard keeps the keys in the NTT domain, they could be transferred to the classical domain, as in BGV. This avoids having to repeat the transfer operation during encryption.

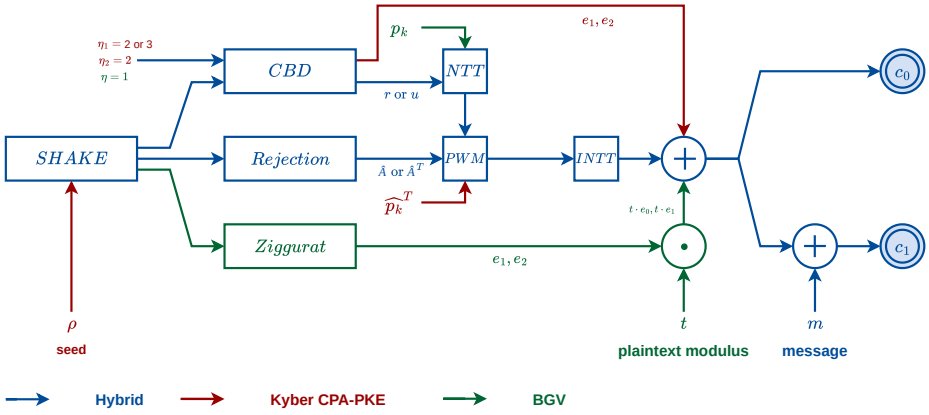


Fig. 3: Hybrid encryption for BGV/ML-KEM.

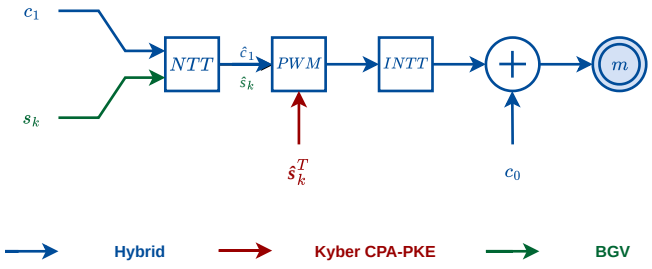


Fig. 4: Hybrid decryption for BGV/ML-KEM.

5.3 Random Number Generator

Uniform Rejection. In BGV, we need to sample an element $a \in \left[-\frac{q}{2}\ ;\ \frac{q}{2}\right]$ from the uniform distribution. The idea is to reuse the Pseudorandom Function (PRF) or eXtendable-Output Function (XOF) defined in [35] for ML-KEM, which uses SHAKE. In general, the same function can be used for both algorithms, namely SHAKE (SHAKE128 as XOF and SHAKE256 as PRF in ML-KEM), standardized in [33].

Gaussian Sample. When creating an error sample ($e, e_0, e_1 \leftarrow \chi_e$) for BGV, as suggested by the Homomorphic Encryption Security Standard (HESS) in [2], one should use a normal distribution with a standard deviation of $\sigma \approx 3.19$. There are several techniques for generating random normal numbers, including the Box-Muller, Knuth-Yao, Kinderman, Polar, Monty Python, Von Neumann-type, and Ziggurat methods. According to Matsumoto [30] and Prest [39], the Ziggurat method [29] is the fastest and has the lowest rejection rate of all the previously mentioned methods. Furthermore, since the Ziggurat method is a rejection method that requires a source of uniformly distributed random numbers, we can also use the SHAKE algorithm of ML-KEM to generate them.

Centered Binomial Distribution (CBD). For the generation of the secret sample ($s \leftarrow \chi_s$) in BGV, the HESS [2] recommends a ternary distribution, i.e., selecting each coefficient uniformly in $\{-1, 0, 1\}$. Generating a ternary distribution is equivalent to using a CBD with $\eta = 1$, defined by:

$$CBD_\eta := \left(\sum_{i=1}^{\eta} a[i] - \sum_{i=1}^{\eta} b[i] \ : \ a, b \xleftarrow{\$} \{0;1\}^\eta \right).$$

As can be seen in the above definition, the generation of random numbers in CBD does not depend on the modulus q or the degree N. For this reason, a single component is used for both ML-KEM and BGV, with only the output length and η changing. SHAKE can also be used to generate the seed for the CBD process.

5.4 Selection of BGV Parameters

To minimize the computational cost, we use the leveled version of BGV. Considering our use case (Sect. 2), we observed that only one or two multiplications are executed in the homomorphic domain. We therefore restrict ourselves to 2-level BGV ($q = q_0 \cdot q_1$, *e.g.* mean computation) and 3-level BGV ($q = q_0 \cdot q_1 \cdot q_2$, *e.g.* standard deviation), whose respective multiplicative depths exactly cover the circuits involved. In both cases, the noise generated by additions is negligible compared to that of scalar multiplication.

To ensure a minimum level of security and an optimal noise budget for decryption, we rely on the `FHEgen` tool, as recommended in [8,31]. In the first scenario, we configure `FHEgen`[5] with a 128-bit security level for one and two multiplication depths. In our case, we selected the moduli q_i as follows:

[5] Note that `FHEgen` does not offer an option for zero multiplications, and that the noise generated by additions is negligible compared to that of scalar multiplication.

- $q_0 = 13 \cdot 2^8 + 1 = 3\,329$ (as ML-KEM),
- $q_1 = 1023 \cdot 2^{13} + 1 = 8\,380\,417$ (as Module-Lattice-Based Digital Signature Algorithm (ML-DSA)),
- q_2 a prime number such that $\log_2(q_2) \approx 63$, e.g. $q_2 = 4611686160161308673$.

In line with Jyrwa et al.'s suggestion in [45], we approximate $\log_2(q_0) + \log_2(q_1) \approx 35$. Therefore, $\log_2(q_2)$ must be approximately 63 to obtain $\log_2(Q) \approx 97$ (as proposed by `FHEgen`) while maintaining the same multiplication depth.

However, `FHEgen` uses a heuristic to determine the level of security against known attacks (uSVP, BDD, and hybrid). Therefore, we use the `fastParameterSelection` tool [7], which provides exact verification for each attack. Note that it does not ensure correct decryption, which is why combining the two tools is beneficial. In the appendix, Tab. 6 summarizes the security levels for each modulus and each attack given by `fastParameterSelection`; we can see that, in the worst-case scenario, the security level is 285, which leaves us a sufficient margin for a target security level of 128 bits.

5.5 NTT Computation

In this subsection, we introduce the NTT, a finite-field version of the Discrete Fourier Transform (DFT) [1,38]. The NTT significantly accelerates polynomial multiplication by reducing the computational complexity from quadratic to quasi-linear time. We therefore build on existing work on the NTT in the LWE context to adapt it to our BGV and ML-KEM use case.

Definition 5 (Number Theoretic Transform). *Let $a \in \mathbb{Z}_q$ and ζ_n be a primitive n-th root of unity in $\mathbb{Z}_q$. The forward transform, called the **NTT**, is defined by $\hat{a} = \mathrm{NTT}(a)$, where*

$$\hat{a}_j = \sum_{i=0}^{n-1} a_i \, \zeta_n^{i \cdot j} \pmod{q}, \quad \text{for } j = 0, 1, \ldots, n-1.$$

*The inverse transform, called the **Inverse Number Theoretic Transform (INTT)**, is defined by $a = \mathrm{INTT}(\hat{a})$, where*

$$a_i = n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \, \zeta_n^{-i \cdot j} \pmod{q}, \quad \text{for } i = 0, 1, \ldots, n-1.$$

The NTT and INTT satisfy $a = \mathrm{INTT}(\mathrm{NTT}(a))$, $\forall a \in \mathbb{Z}_q$.

Proposition 1 (Convolution). *Let $c = a \cdot b \in \mathbb{Z}_q[X]/(X^N - 1)$ be the cyclic convolution of a and b. Then it holds that $\mathrm{NTT}(c) = \mathrm{NTT}(a) \circ \mathrm{NTT}(b)$, where $\circ$ denotes component-wise multiplication.*

Residue Number System. As mentioned in Subs. 5.4, at most two multiplication levels are required. To minimize noise from homomorphic multiplication while maintaining 128-bit security in BGV, we use a large q value, unlike in ML-KEM. In leveled BGV, q is defined as a product of primes, enabling efficient modulus switching after multiplication to reduce noise. This approach optimizes NTT computation.

Definition 6 (Chinese Remainder Theorem (CRT)). *The CRT states that if the m_i are pairwise co-prime and $x \equiv a_i \bmod m_i$ for all $i = 1, \ldots, k$, then x has a unique solution modulo $m_1 m_2 \cdots m_k$.*

Indeed, according to Definition 6, we can split the computation of the NTT modulo q into one NTT per q_i. We have:

$$\mathbb{Z}_q[X]/(X^n+1) \cong \mathbb{Z}_{q_0}[X]/(X^n+1) \times \mathbb{Z}_{q_1}[X]/(X^n+1) \times \mathbb{Z}_{q_2}[X]/(X^n+1)$$

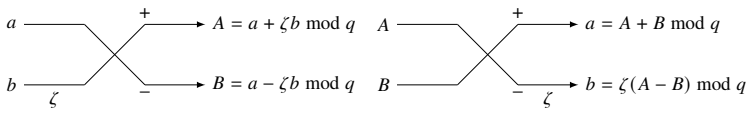
After applying NTT to each q_i, we recombine to obtain the result modulo q.

NTT radix-2. In order to calculate the NTT, the most common approach is based on *divide and conquer*: the problem is recursively broken down into smaller sub-problems, a strategy known as *radix-r*. The main advantages are the reduction in the degree of the polynomials manipulated at each stage and the possibility of parallelizing the calculations, which is particularly beneficial in a multi-core context. In the case of ML-KEM – as well as ML-DSA – the reference implementation [9] adopts a radix-2 NTT.

Definition 7 (NTT radix-2). *An NTT radix-2 is a recursive decomposition of the problem of size n into two subproblems of size $n/2$, exploiting the structure of the roots of unity in $\mathbb{Z}_q$. For a polynomial $a \in R_q$ of degree $n-1$, we separate the even and odd coefficients:*

$$\hat{a}[k] = \sum_{j=0}^{n-1} a[j]\,\zeta_n^{jk} = \underbrace{\sum_{j=0}^{n/2-1} a[2j]\,\zeta_{n/2}^{jk}}_{A_{\text{even}}[k]} + \zeta_n^k \cdot \underbrace{\sum_{j=0}^{n/2-1} a[2j+1]\,\zeta_{n/2}^{jk}}_{A_{\text{odd}}[k]},$$

which reduces the calculation of an NTT of size n to two NTTs of size $n/2$, reducing the complexity from $O(n^2)$ to $O(n \log n)$. The two decompositions are the Cooley-Tukey (CT) butterfly [14] and the Gentleman-Sande (GS) butterfly [22] in Fig. 5, defined respectively by:



(a) Butterfly CT (b) Butterfly GS

Fig. 5: Butterfly CT/GS

where ζ is a root of unity in $\mathbb{Z}_q$. The CT butterfly is mainly used in the forward direction (NTT) [14], while the GS butterfly is mainly used in the inverse direction (INTT); their composition ensures that $INTT \circ NTT = \mathrm{Id}$.

We recognize that the prime q of ML-KEM is constructed in the form $q = k \cdot 2^m + 1$, so that $\mathbb{Z}_q$ contains primitive 2^m-th roots of unity. This allows for faster computations using NTTs. For leveled BGV, $Q = \prod_{i=0}^{L-1} q_i$, where each q_i is a small prime. In order to optimize with ML-KEM, the idea is to take q_i to be of the form $q_i = k \cdot 2^m + 1$. As a result, the same accelerator used by ML-KEM can be used for NTTs.

We want to find the NTT representation $NTT(a) \in \mathbb{Z}_{q_i}[X]/(X^{8192} + 1)$ for each q_i. Moreover, for $q_0 = 3\,329$, $N = 8\,192$ does not divide $q_0 - 1$, so there is no primitive N-th root of unity in $\mathbb{Z}_{q_0}$.

We propose a fast decomposition technique that factors the polynomial ring into a 32×256 product, allowing reuse of an NTT module of size $n = 256$. However, this approach makes it impossible to directly apply 2-radix NTT to compute the NTT on the quotient ring:

$$\mathbb{Z}_{q_i}[X]/(X^n + 1) \cong \mathbb{Z}_{q_i}[X]/(X^{n_1} + 1) \times \mathbb{Z}_{q_i}[X]/(X^{n_2} + 1),$$

where $n = n_1 \cdot n_2$ with $n_1 = 32$ and $n_2 = 256$. To address this issue, we consider several options:

1. **Increase q_i:** A larger parameter q can provide a primitive root of unity. However, this approach carries a substantial memory cost, as it requires storing or recalculating the unique roots of unity associated with the new q_i.
2. **Decrease n:** Decreasing the parameter n in the BGV scheme would lower its security, which is not an acceptable option.
3. **Increase n:** Increasing the parameter n leads to a higher number of coefficients to process, increasing both storage requirements and algorithmic complexity.
4. **Use another algorithm:** Compute the NTT with a different approach.

Therefore, we decided to use *Nussbaumer's trick* to split the NTT.

Nussbaumer's Trick NTT. Unlike the Four-Step NTT, Nussbaumer's trick does not require a primitive root of unity of order $N = 8\,192$. Instead, only roots of unity for the smaller transforms of size $2n = 256$ are needed.

Definition 8 (Nussbaumer's trick [6,36]). *Let $N = 2nm$ be the polynomial degree. Nussbaumer's trick defines the ring isomorphism via the mapping*

$$\Psi : \mathbb{Z}_q[x]/(x^{2nm} + 1) \xrightarrow{\cong} \left(\mathbb{Z}_q[y]/(y^{2n} + 1)\right)[x]/(x^m - y)$$

$$\sum_{i=0}^{N-1} a_i \cdot x^i \mapsto \sum_{i=0}^{m-1} \underbrace{\left(\sum_{j=0}^{2n-1} a_{m\cdot j+i} \cdot y^j\right)}_{A_j \in \mathbb{Z}_q[y]/(y^{2n}+1)} x^i, \text{ with } y = x^m. \tag{5}$$

Remark 3. Note that y should be the primitive $4n$-th root of unity in $\mathbb{Z}_q[y]/(y^{2n} + 1)$ to be NTT-friendly.

To improve efficiency, we can rewrite Eq. 5 as a sum of m instances of the NTT on $\mathbb{Z}_q[y]/(y^{2n}+1)$, which, in our case, yields the following formulation:

Let $N = 8\,192 = 32 \times 256$.

1. **Partitioning the coefficient vector.** We can split the vector (a_i) into 32 sub-vectors of 256 elements such that:

$$a^{(i)} = \big(a_i,\ a_{i+32},\ a_{i+2\cdot 32},\ \ldots,\ a_{i+255\cdot 32}\big), \quad \text{for } i = 0, \ldots, 31. \tag{6}$$

2. **Applying the NTT_{256} to each sub-vector.** For each i, we can compute

$$\widehat{a^{(i)}} = \mathrm{NTT}_{256}\big(a^{(i)}\big) = \Big(\sum_{j=0}^{2n-1} a_{i+mj}\,\zeta^{jk}\Big), \quad \text{for } k = 0, \ldots, 255.$$

   where ζ is a primitive 256-th root of unity modulo q.

3. **Final expression in NTT form.** Replacing each inner sum with its NTT version, the global transform becomes

$$\sum_{i=0}^{N-1} a_i\,x^i \ \longmapsto\ \sum_{i=0}^{m-1}\Big(\sum_{k=0}^{2n-1} \widehat{a}_k^{(i)}\,y^k\Big)x^i, \tag{7}$$

This trick allows us to use the same NTT function for both cases (ML-KEM and BGV); it can be the reference implementation from the NIST standard [35] or any other optimized implementation. In addition, the fact that we have selected $q_0 = 3\,329$ for BGV allows us to reuse the ζ table from ML-KEM[6]. In Tab. 7, we summarize the complexity of various multiplication algorithms.

6 Implementation

In order to show the feasibility of our proposed scheme and to test the selected parameters with our use case, we have implemented the embedded part (key generation, encryption and decryption) on a PolarFire SoC Discovery Kit platform and the server part (addition and multiplication) on an Intel Xeon Gold 6226R with SageMath.

6.1 Parameter Selection

According to [47], the literature distinguishes several time windows for the analysis of interbeat intervals: an ultra-short window of 60 seconds (≈60–80 toc), a window of 240 seconds (≈240–320 toc), and the standard 5-minute window (≈300-400 toc). For long-term analyses, recordings can extend up to 24 hours (≈100,000 toc), particularly for calculating the SDNN. To ensure that the chosen parameters allow accurate decryption, we evaluate the critical noise level in a `SageMath`[7] implementation using the parameters

[6] The same reasoning applies to ML-DSA.

[7] `SageMath` is a free mathematical software system based on Python [49].

from Subs. 5.4. We do this for a standard 5-minute window of $N = 400$ ciphertexts. After $N = 400$ homomorphic additions, we measure $\|v\|_\infty \approx 2^{66} < B_{\max} \approx \frac{q}{2t} = 2^{79}$, which satisfies the correctness condition, leaving a margin of 13 bits before decryption failure. This confirms that the scheme remains correct well beyond the target workload, supporting up to ≈23 000 additions under these parameters. Further details on the runtime performance of each operation are reported in Tab. 3.

Table 3: Benchmark of the mean computation with BGV on an Intel Xeon Gold 6226R (2.90 GHz), $n = 8192$, $q \approx 2^{96}$, $t = 65537$, $\sigma = 3.2$. Timings reported as min. over 10 runs.

	$N = 400$		$N = 10000$	
Operation	Min (ms)	Min cycles	Min (ms)	Min cycles
KeyGen	152.33	548.4×10^6	138.07	538.5×10^6
KeySwitch	147.76	531.9×10^6	139.69	544.8×10^6
Encrypt	125.46	451.7×10^6	119.21	464.9×10^6
Add	35.87	129.1×10^6	38.11	148.6×10^6
Decrypt	27.98	100.7×10^6	28.58	111.5×10^6
Encrypt (total)	61.88 *s*	*(*154.71 *ms/op)*	1454.25 *s*	*(*145.43 *ms/op)*
Add (total)	19.56 *s*	*(*49.14 *ms/op)*	513.65 *s*	*(*51.38 *ms/op)*

6.2 NTT Implementation

In this subsection, we present the implementation of NTT_{8192}. As explained previously (cf. Sect. 5), one of the main ideas is to reuse as much as possible of the reference code for NTT_{256} from ML-KEM available in the official repository of the *Crystals-Kyber* project [9]. To multiply two polynomials $f, g \in \mathbb{Z}_q[x]/(x^{8192} + 1)$ using the optimization described in Subs. 5.5, we proceed as follows:

1. **Incomplete NTT Transformation:** We compute the incomplete NTT of f and g using the transformation ψ_{8192} defined in Alg. 1: $F = \texttt{psi_8192}(f, \boldsymbol{\zeta}, q)$ and $G = \texttt{psi_8192}(g, \boldsymbol{\zeta}, q)$. This function rearranges the coefficients of f and g according to the partitioning scheme in Eq. 6, then applies the `ntt_256` function to each of the 256-coefficient sub-blocks, as described in [9]. The list of required roots of unity $\boldsymbol{\zeta}$ can be precomputed and stored in memory or hard-coded.

Algorithm 1: `psi_8192`(f, ζ, q) – The forward transform ψ_{8192}

Input: $f \in \mathbb{Z}_q[X]/(X^{8192}+1)$ with underlying 256-point NTT_{256}
Output: $\hat{f} := \{\hat{F}_i\}_{i=0}^{31}$

Map f into $\Psi(f) = \sum_{j=0}^{31} F_j x^j$ where each F_j is a polynomial in $\mathbb{Z}_q[y]$; **for** $i \leftarrow 0$ **to** 31 **do**
  $F_i \leftarrow [0]_{j=0}^{255}$;
  for $j \leftarrow 0$ **to** 255 **do**
    $s \leftarrow 32 \cdot j + i$;
  $\hat{F}_i \leftarrow$ `NTT_256`(F_i, ζ, q);
return $\{\hat{F}_i\}_{i=0}^{31}$;

2. **Negacyclic Point-Wise Multiplication (PWM):** As specified in Alg. 2, we compute $H =$ `conv_8192`(F, G, ζ, q). This function performs negacyclic polynomial multiplication on each 256-coefficient sub-block using the `poly_basemul` function from [9]. After multiplication, we apply the reduction $X^{32} - Y$ when the index s is at least 32.

Algorithm 2: `conv_8192`(F, G, ζ, q) – Negacyclic Convolution

Input: $F, G \in \mathbb{Z}_q[X]/(X^{8192}+1)$, ζ roots, q modulus
Output: $H := F \odot G$
Initialize $H \leftarrow [\,[0]^{256}\,]^{32}$;
for $j_1 \leftarrow 0$ **to** 31 **do**
  for $j_2 \leftarrow 0$ **to** 31 **do**
    $s \leftarrow j_1 + j_2$;
    $j \leftarrow s \bmod 32$;
    `tmp` $\leftarrow$ `poly_basemul`$(F[j_1], G[j_2], \zeta, q)$;
    if $s \geq 32$; // $\bmod(X^{32} - Y)$
    then
      for $i \leftarrow 0$ **to** 63 **do**
        $z \leftarrow \zeta[64 + i]$;
        $a_0, a_1 \leftarrow$ `tmp`$[4i]$, `tmp`$[4i+1]$;
        $a_2, a_3 \leftarrow$ `tmp`$[4i+2]$, `tmp`$[4i+3]$;
        `tmp`$[4i] \leftarrow (a_1 \cdot z) \bmod q$;
        `tmp`$[4i+1] \leftarrow a_0$;
        `tmp`$[4i+2] \leftarrow (-a_3 \cdot z) \bmod q$;
        `tmp`$[4i+3] \leftarrow a_2$;
    for $k \leftarrow 0$ **to** 255 **do**
      $H[j][k] \leftarrow (H[j][k] +$ `tmp`$[k]) \bmod q$;
return H;

3. **INTT Transformation:** We compute the inverse transformation ψ_{8192}^{-1} defined in Alg. 3: $h =$ `psi_inv_8192`$(H, \zeta, q) = f \cdot g$.

   This function computes the inverse NTT for each 256-coefficient sub-block using the `iNTT_256` function from [9], then reorders the coefficients to obtain the final multiplication result.

Algorithm 3: `psi_inv_8192`(H, ζ, q) – The inverse transform ψ_{8192}^{-1}

Input: $H := \{\hat{H}_i\}_{i=0}^{31}$ with underlying 256-point iNTT_{256}
Output: $h \in \mathbb{Z}_q[X]/(X^{8192}+1)$ where $h := \{h_k\}_{k=0}^{8191}$

Initialize $h \leftarrow [0]_{k=0}^{8191}$;
for $i \leftarrow 0$ **to** 31 **do**
  $H_i \leftarrow \text{iNTT}_{256}(\hat{H}_i, \zeta, q)$;
  for $j \leftarrow 0$ **to** 255 **do**
    idx $\leftarrow i + 32 \cdot j$;
    $h[\text{idx}] \leftarrow H_i[j]$;
return h;

6.3 Embedded Implementation

For the embedded implementation, we selected the PolarFire SoC Discovery Kit board from Microchip, which contains five 64-bit RISC-V cores. We use only a single core of the PolarFire for the benchmarks in Tab. 4 and Tab. 5. This benchmark confirms Nussbaumer's structure: the `psi_8192` and `psi_inv_8192` operations, which correspond to multiplications by monomials (simple rotations of coefficients), are cheap ($\approx 8\,\%$ and $\approx 6\,\%$ of the total time, respectively), while `conv_8192` accounts for $\approx 85\,\%$ of the computation time.

Each multiplication modulo q requires a reduction using native integer division, which can be expensive. These reductions could be replaced with Montgomery reduction, which replaces costly divisions with multiplications and bit shifts, significantly reducing the dominant cost of the `conv_8192` function. Furthermore, we observed that BGV key generation using Nussbaumer's trick takes 94 679 558 cycles, or 157.80 μs.

Note that our implementation also works for $q = 8380417$ (ML-DSA) with performance identical to that of $q = 3329$. Furthermore, it shares the same implementation, up to a few constants (roots of unity).

Table 4: Benchmark of NTT computation on a RISC-V U54 PolarFire SoC Discovery Kit with `-O3` option (600 MHz), for $n = 8192$, $q = 3329$ and $q = 8380417$.

Operation	Max. clock cycles	Max. time (ms)
`psi_8192`	4 073 114	6.79
`conv_8192`	41 265 659	68.78
`psi_inv_8192`	2 959 048	4.93
`add_8192`	245 801	0.41

7 Conclusion and Future Work

In this paper we have explored the possibilities for an agile cryptographic scheme that combines ML-KEM with FHE in an embedded setting for the secure processing of

Table 5: Benchmark of the BGV implementation on a RISC-V U54 PolarFire SoC Discovery Kit with `-O3` option (600 MHz), for $n = 8192$, $q = 3329$ and $q = 8380417$.

Operation	Max. clock cycles	Max. time (ms)	Code size (KB)	Memory usage (KB)
BGV KeyGen	56 235 000	93.73	47.95	567.92
BGV Encrypt	94 679 558	157.80	48.42	699.97
BGV Decrypt	49 602 603	82.67	48.03	567.98

health data. We have shown how large parts of the algorithms can be shared with the goal of minimizing the implementation's code size and memory footprint, such as the random number generators and the primitive roots. Choosing BGV as the homomorphic framework further allows the use of common arithmetic computations. However, the fusion of the parameters entails some challenges, in particular for the implementation of the NTT. In addition, our implementation trick allows us to reuse the hardware accelerator of an existing ML-KEM implementation to add a BGV implementation. We have demonstrated with a proof-of-concept implementation that our solution is sound and effective for typical computations in a health data case study.

As a next step in our work, we plan to explore different trade-offs between optimized software implementations (e.g., using multiple cores) and the use of hardware accelerators. There is a large design space regarding the algorithmic choices for the involved arithmetic computations, and they need to be adapted to the target platform. Finally, in an embedded setting, side-channel attacks should be taken into account. In future work, we would like to study the vulnerability of our co-design to such attacks and propose side-channel resistant implementations.

Appendix

Table 6: Security analysis using `fastParameterSelection` for an LWE dimension of 8192

Q	Log Q	USVP	USVP$_s$	USVP num	BDD	BDD$_s$	BDD num	Output
q_0	12	2737	4159	2673	2788	3698	2696	3698
q_0q_1	35	948	1145	1009	1009	1063	1010	1063
$q_0q_1q_2$	97	285	330	320	320	322	320	322

Table 7: Comparison of polynomial multiplication algorithms in $\mathbb{Z}_q[X]/(X^n + 1)$ for $n = 8192$.

Algorithm	**Complexity**	**Condition**	**Mult. for $n = 8192$**
Schoolbook	$O(n^2)$	None	67 108 864
Karatsuba [26]	$O(n^{1.585})$	None	$\approx 1.84 \times 10^6$
Toom–Cook 3 [13]	$O(n^{1.465})$	None	$\approx 6.9 \times 10^5$
Radix-2 NTT (GS) [14]	$O(n \log n)$	$q \equiv 1 \pmod{2n}$	53 248
Radix-2 NTT (CT) [14]	$O(n \log n)$	$q \equiv 1 \pmod{2n}$	53 248
Nussbaumer [36]	$O(n \log n)$	None	32 768

```
Generated your BGV configuration !
model:   Base
sec:     189
n:       4096
t:       65537
log(q):  65 (61, 4, 4)
log(P):  10
```

(a) One multiplication depth

```
Generated your BGV configuration !
model:   Base
sec:     210
n:       8192
t:       65537
log(q):  98 (63, 31, 4)
log(P):  38
```

(b) Two multiplication depth

Fig. 6: Example of `FHEgen` execution (terminal output)

Algorithm 4: `BaseMul`$(a_0, a_1, b_0, b_1, z, q)$ [9]

Input: $a_0, a_1, b_0, b_1, z \in \mathbb{Z}_q$
Output: (c_0, c_1) such that $c = a \cdot b$ in $\mathbb{Z}_q[x]/(x^2 - z)$

1 $t \leftarrow a_1 \cdot b_1$;
2 $t \leftarrow t \cdot z$;
3 $c_0 \leftarrow a_0 \cdot b_0 + t \bmod q$;
4 $c_1 \leftarrow a_0 \cdot b_1 + a_1 \cdot b_0 \bmod q$;
5 **return** (c_0, c_1);

Algorithm 5: $\texttt{PolyMul}(a, b, \zeta, q)$ – Negacyclic PWM [9]

Input: f, g two polynomials of degree 256
Output: $h = f \cdot g$

$\hat{f} \leftarrow \texttt{NTT_256}(f, \zeta, q)$;
$\hat{g} \leftarrow \texttt{NTT_256}(g, \zeta, q)$;
for $i \leftarrow 0$ **to** 127 **do**
    $z \leftarrow \zeta[64 + i]$;
    $(\hat{h}[4i],\ \hat{h}[4i + 1]) \leftarrow \texttt{BaseMul}(\hat{f}[4i], \hat{f}[4i + 1], \hat{g}[4i], \hat{g}[4i + 1], z, q)$;
    $(\hat{h}[4i + 2],\ \hat{h}[4i + 3]) \leftarrow$
     $\texttt{BaseMul}(\hat{f}[4i + 2], \hat{f}[4i + 3], \hat{g}[4i + 2], \hat{g}[4i + 3], -z, q)$;
$h \leftarrow \texttt{NTT}^{-1}(\hat{h})$;
return h;

Algorithm 6: $\texttt{NTT_256}(f, \zeta, q)$ – Number Theoretic Transform [9]

Input: $f = [f_0, \ldots, f_{255}]$ (array of 256 integers)
Output: $F = \texttt{NTT}_{256}(f)$

$k \leftarrow 1$;
for $len \leftarrow 128$; $len \geq 2$; $len \leftarrow len \gg 1$ **do**
    for $start \leftarrow 0$; $start < 256$; $start \leftarrow j + len$ **do**
        $z \leftarrow \zeta[k{+}{+}]$;
        for $j \leftarrow start$ **to** $start + len - 1$ **do**
            $t \leftarrow z * F[j + \text{len}] \pmod q$;
            $F[j + \text{len}] \leftarrow F[j] - t$;
            $F[j] \leftarrow F[j] + t$;
return F;

Algorithm 7: `iNTT_256`$(F, \boldsymbol{\zeta}, q)$ – Inverse NTT [9]

Input: $F = [F_0, \ldots, F_{255}]$
Output: f

```
1  const inv ← 128^{-1} (mod q);
2  k ← 127;
3  for len = 2, 4, 8, ..., 128 do
4      for start = 0 to 255 step 2 · len do
5          z ← ζ[k − −];
6          for j = start to start + len − 1 do
7              t ← f[j];
8              f[j] ← (t + f[j + len]) mod q;
9              f[j + len] ← f[j + len] − t;
10             f[j + len] ← (z · f[j + len]) mod q;
11 for j = 0 to 255 do
12     f[j] ← (f[j] · inv) mod q;
13 return f;
```

Acknowledgments. This study was funded by the "Futur & Ruptures 2024" program managed by the IMT, Carnot TSN Institute, and the Mines-Télécom Foundation.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Agarwal, R.C., Burrus, C.S.: Number theoretic transforms to implement fast digital convolution. Proceedings of the IEEE **63**, 550–560 (1975)
2. Albrecht, M., et al.: Homomorphic encryption security standard. Tech. rep., HomomorphicEncryption.org, Toronto, Canada (2018)
3. Alder, S.: The biggest healthcare data breaches of 2024. https://www.hipaajournal.com/biggest-healthcare-data-breaches-2024/ (2025), The HIPAA Journal
4. B., P.H., Shivaprakasha, K.S., Bhat, G.V.: Secure Healthcare Data Processing using Homomorphic Encryption. In: ICDCECE 2025. pp. 1–6 (2025). https://doi.org/10.1109/ICDCECE65353.2025.11035434
5. Badawi, A.A., et al.: OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915
6. Bernstein, D.J.: Multidigit multiplication for mathematicians. Tech. rep., University of Illinois at Chicago (2001), http://cr.yp.to/papers.html#m3
7. Biasioli, B., Kirshanova, E., Marcolla, C., Rovira, S.: A tool for fast and secure LWE parameter selection: the FHE case. Cryptology ePrint Archive, Paper 2024/1895 (2024), https://eprint.iacr.org/2024/1895
8. Biasioli, B., Marcolla, C., Calderini, M., Mono, J.: Improving and automating BFV parameters selection: An average-case approach. Cryptology ePrint Archive, Paper 2023/600 (2023), https://eprint.iacr.org/2023/600

9. Bos, J., et al.: Kyber: A CCA2-secure module-lattice-based key-encapsulation mechanism. https://github.com/pq-crystals/kyber (2023)
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (2014)
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Annual International Conference on the Theory and Applications of Cryptology and Information Security. pp. 409–437. Springer (2017)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
13. Cook, S.A., Aanderaa, S.O.: On the minimum computation time of functions. Transactions of the American Mathematical Society **142**, 291–314 (1969)
14. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**(90), 297–301 (1965), www.jstor.org/stable/2003354
15. Couderc, J.P.: The telemetric and Holter ECG warehouse initiative (THEW): A data repository for the design, implementation and validation of ECG-related technologies. In: IEEE EMBC 10. pp. 6252–6255 (2010)
16. Diaz, N.: 1 year later: The change healthcare cyberattack and its lasting impact on healthcare (2025), Becker's Hospital Review, https://www.beckershospitalreview.com/healthcare-information-technology/cybersecurity/1-year-later-the-change-healthcare-cyberattack-and-its-lasting-impact-on-healthcare/
17. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
18. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: EUROCRYPT 2015. pp. 617–640. Springer (2015)
19. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
20. European Commission: European health data space regulation (ehds). https://health.ec.europa.eu/ehealth-digital-health-and-care/european-health-data-space-regulation-ehds_en (2025), accessed: March 11, 2026
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive **2012**, 144 (2012)
22. Gentleman, W.M., van der Sande, G.: Fast fourier transforms: for fun and profit. In: AFIPS '66 (Fall) (1966), https://api.semanticscholar.org/CorpusID:207170956
23. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. STOC '09, Association for Computing Machinery (2009). https://doi.org/10.1145/1536414.1536440
24. Halevi, S., Shoup, V.: Algorithms in HElib. IACR Cryptol. ePrint Arch. (2014)
25. von Kalckreuth, N., et al.: Trustworthiness of the electronic health record in Germany: An exploratory, user-centered analysis. Frontiers in Digital Health **7**, 1473326 (2025). https://doi.org/10.3389/fdgth.2025.1473326
26. Karatsuba, A., Ofman, Y.: Multiplication of Multidigit Numbers on Automata. Soviet Physics Doklady **7**, 595 (Jan 1963)
27. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. Cryptology ePrint Archive (2021), https://eprint.iacr.org/2021/204
28. Kocabas, O., et al.: Assessment of cloud-based health monitoring using homomorphic encryption. In: ICCD 2013 (2013). https://doi.org/10.1109/ICCD.2013.6657078
29. Marsaglia, G., Tsang, W.: The ziggurat method for generating random variables. Journal of Statistical Software **5** (2000). https://doi.org/10.18637/jss.v005.i08
30. Matsumoto, M.: Secure and Private Data Collection and Application. Ph.D. thesis, Ochanomizu University, Tokyo, Japan (March 2025), https://teapot.lib.ocha.ac.jp/records/2005333

31. Mono, J., Marcolla, C., Land, G., Güneysu, T., Aaraj, N.: Finding and evaluating parameters for BGV. Cryptology ePrint Archive (2022), https://eprint.iacr.org/2022/706
32. Munjal, K., Bhatia, R.: A systematic review of homomorphic encryption and its contributions in healthcare industry. Complex & Intelligent Systems **9**(4), 3759–3786 (2023). https://doi.org/10.1007/s40747-022-00756-z
33. NIST: SHA-3 standard: Permutation-based hash and extendable-output functions. Tech. Rep. FIPS 202, U.S. Department of Commerce (2015). https://doi.org/10.6028/NIST.FIPS.202
34. NIST: Module-Lattice-Based Digital Signature Standard. Tech. Rep. FIPS 204, U.S. Department of Commerce (2024). https://doi.org/10.6028/NIST.FIPS.204
35. NIST: Module-Lattice-Based Key-Encapsulation Mechanism Standard. Tech. Rep. FIPS 203, U.S. Department of Commerce (2024). https://doi.org/10.6028/NIST.FIPS.203
36. Nussbaumer, H.: Fast polynomial transform algorithms for digital convolution. IEEE Transactions on Acoustics, Speech, and Signal Processing **28**(2), 205–215 (1980). https://doi.org/10.1109/TASSP.1980.1163372
37. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
38. Pollard, J.M.: The fast fourier transform in a finite field. Mathematics of Computation **25**, 365–374 (1971)
39. Prest, T.: Gaussian sampling in lattice-based cryptography. Theses, Ecole normale supérieure - ENS PARIS (Dec 2015), https://theses.hal.science/tel-01245066
40. Ras, A., Loiseau, A., Carmona, M., Pontié, S., Renault, G., Smith, B., Valea, E.: PHOENIX: Crypto-agile hardware sharing for ML-KEM and HQC. Cryptology ePrint Archive, Paper 2025/601 (2025), https://eprint.iacr.org/2025/601
41. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing. pp. 84–93. STOC '05, Association for Computing Machinery (2005)
42. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography (2024), https://arxiv.org/abs/2401.03703
43. Resende, D., El Mrabet, N., Kühne, U.: An agile homomorphic encryption and ml-kem scheme for privacy-preserving health data processing. In: Lightweight Cryptography for Security and Privacy. Springer Nature Switzerland
44. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press pp. 169–179 (1978)
45. Roy Jyrwa, J.J., Sinha, S., Dutta, B.: Comparative analysis of noise generated in BGV homomorphic encryption: Lattigo vs FHEgen parameters. In: ICACRS 2025 (2025)
46. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL (Jan 2023), Microsoft Research, Redmond, WA
47. Shaffer, F., Ginsberg, J.: An overview of heart rate variability metrics and norms. Frontiers in Public Health **5** (2017). https://doi.org/10.3389/fpubh.2017.00258
48. Sly, L.: U.S. soldiers are revealing sensitive and dangerous information by jogging. The Washington Post (2018), Jan. 28
49. The Sage Developers: SageMath, the Sage Mathematics Software System (2025), `https://www.sagemath.org`
50. Turan, M.S., McKay, K., Kang, J., Kelsey, J., Chang, D.: Ascon-based lightweight cryptography standards for constrained devices. Tech. Rep. SP 800-232, U.S. Department of Commerce, NIST, Gaithersburg, MD (August 2025)