A Fast RLWE-Based IPFE Library and its Application to Privacy-Preserving Biometric Authentication

Supriya Adhikary, Angshuman Karmakar

(Author's version)

Abstract

With the increased use of data and communication through the internet and the abundant misuse of personal data by many organizations, people are more sensitive about their privacy. Privacy-preserving computation is becoming increasingly important in this era. Functional encryption allows a user to evaluate a function on encrypted data without revealing sensitive information. Most implementations of functional encryption schemes are too time-consuming for practical use. Mera et al. first proposed an inner product functional encryption scheme based on ring learning with errors to improve efficiency. In this work, we optimize the implementation of their work and propose a fast inner product functional encryption library. Specifically, we identify the main performance bottleneck, which is the number theoretic transformation based polynomial multiplication used in the scheme. We also identify the micro and macro level parallel components of the scheme and propose novel techniques to improve the efficiency using *open multi-processing* and *advanced vector extensions 2* vector processor. Compared to the original implementation, our optimization methods translate to 89.72%, 83.06%, 59.30%, and 53.80% improvements in the **`Setup`**, **`Encrypt`**, **`KeyGen`**, and **`Decrypt`** operations respectively, in the scheme for **standard** security level. Designing privacy-preserving applications using functional encryption is ongoing research. Therefore, as an additional contribution to this work, we design a privacy-preserving biometric authentication scheme using inner product functional encryption primitives.

Index Terms

Number theoretic transformation, Inner product functional encryption, OpenMP, AVX2, Privacy-preserving, Biometric authentication

1 Introduction

In the past few decades, privacy in the digital world has become a very important and sensitive matter. Many countries have come up with laws such as the General Data Protection Regulation act of the European Union, the Data Security Regulations of Israel, the Act on the Protection of Personal Information of Japan, etc. to protect their citizens' privacy. However, they are often rendered ineffective due to their limitation of jurisdiction. This is why we need cryptographic methods, such as computation on encrypted data (COED) techniques. As the name implies, these methods facilitate performing computations, thus allowing data manipulation routines to run on encrypted data, which ensures the privacy and confidentiality of the data. Due to their construction from cryptographic methods, these techniques provide a universal and provable solution to the problems described above.

Homomorphic encryption (HE), multi-party computation (MPC), and functional encryption (FE) are the three main COED techniques. Among them, HE and MPC have been studied extensively. There exist different applications and efficient implementations of these techniques on a wide variety of platforms. In comparison, FE techniques have received much less attention than the other two techniques. Briefly, a FE scheme provides a secret key $\mathbf{sk}_f$ to an authorized user. This user can use this key to calculate a function $f(m)$ from the encryption of m. Although there exist FE schemes that support arbitrary functions, they require weaker security notions, such as security against only bounded numbers of collusions [1] or stronger primitives [2]. Further, they are very inefficient. Therefore, an alternate research area has become popular to design FE schemes with limited functionalities. Among them, FE schemes that provide inner product functionality have become particularly very popular [3], [4], [5]. In this work, we are mostly concerned with the inner product functional encryption (IPFE) scheme based on the ring learning with errors (RLWE) problem proposed in [5]. As the name suggests, this scheme is based on the RLWE problem, in contrast to the learning with errors (LWE) problem used in earlier versions

- *Supriya Adhikary and Angshuman Karmakar are with Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India, E-mail:* {adhikarys, angshuman}@cse.iitk.ac.in

of IPFE schemes [3], [4]. The immediate advantage of the former over the latter is that in the RLWE-based schemes, we can use fast polynomial multiplication routines instead of slower matrix-vector multiplications used in LWE-based schemes. This translates into efficient IPFE schemes. We provide more details of this scheme in Section. 3.4.

Our primary motivation behind this work is to find efficient alternatives to widely studied privacy-preserving COED techniques such as HE and MPC. Being based on RLWE the IPFE scheme proposed by Mera et al. [5] is highly parallelizable. We exploit the different levels of parallelism in the scheme for efficient implementation on platforms like multi-core architectures and vector instructions. FE works in a very heterogeneous environment. There are lots of actors in FE, like *Keyserver*, *Client*, *Server*, etc. We would further like to note that even with lots of advancements in computing technologies, largely parallel processing elements like field-programmable gate arrays (FPGA) and graphics processing units (GPU) are relatively less ubiquitous compared to multi-core and vector processors. Hence, exploring parallel and fast implementations on these architectures is also very important without being limited to GPU and FPGA implementations only.

Mera et al. presented the first RLWE-based IPFE scheme in [5], which we will call RLWE-IPFE in this work. This work is focused on efficient implementation and use cases of the RLWE-IPFE scheme. We briefly summarize our contributions in this work below:

1. We have analysed the bottlenecks of the RLWE-IPFE scheme and have improved the performance by parallelizing various components of the scheme. We have used Open Multi-Processing (OpenMP) and Advanced Vector Extensions 2 (AVX2) for parallelization. We have shown how to rigorously analyse multiple parallelization techniques using OpenMP, and choose the most suitable technique for RLWE-IPFE.
2. Number theoretic transformation (NTT) is one of the most computationally expensive components in most lattice-based cryptography schemes. There are some methods to speed-up the NTT multiplication when the size of the polynomials is small (256-1024). However, for large polynomials, those methods do not scale very well due to limitations of memory and registers. We have optimized the NTT using AVX2 instructions for large polynomials of size 4096 or 8192.
3. To show the usability and practicality of IPFE schemes, we deduce novel techniques to map biometric authentication mechanism to inner product operation, which we then implement using RLWE-IPFE scheme for a privacy-preserving biometric authentication scheme. We also prove the formal security and propose parameters for 125-bits and 305-bits of security.

Our implementation is available in the public domain at `https://github.com/s-adhikary/IPFE`

2 NOTATIONS

In this paper, we shall denote R to be a polynomial ring $\mathbb{Z}[x]/\langle x^n+1\rangle$. Also, we shall use a standard notation $R_{\mathbf{q}}$ to denote $R/\mathbf{q}R = \mathbb{Z}_{\mathbf{q}}[x]/\langle x^n+1\rangle$. The modulus $\mathbf{q}$ is an integer of the form $\mathbf{q} = q_1 q_2 \cdots q_t$ such that all q_i's are primes. Here, n is of the form 2^k.

For $a \in R$ (or $a \in R_{\mathbf{q}}$), a polynomial of degree less than n, we shall denote $\boldsymbol{a} \in \mathbb{Z}^n$ (or $\boldsymbol{a} \in \mathbb{Z}_{\mathbf{q}}^n$) as the vector of coefficients of the polynomial a. When the coefficients are sampled from a distribution χ, we write $\boldsymbol{a} \leftarrow \chi$. We write $x \xleftarrow{R} X$ to denote that the element x is sampled uniformly at random from the set X. We have used the notations $\mathcal{D}_\sigma$ and $\mathcal{D}_{\sigma I_n}$ to denote the discrete Gaussian distributions over the sets $\mathbb{Z}$ and $\mathbb{Z}^n$ respectively, with standard deviation σ and mean 0. The notation $[\ell]$ denotes the set $\{1, 2, \cdots, \ell\}$ and for any vector $\boldsymbol{v}$, the notations $||\boldsymbol{v}||_\infty$ and $||\boldsymbol{v}||$ denotes the infinity norm and Euclidean norm, respectively. We will denote 1_R to be the polynomial in R with all coefficients being the element $1 \in \mathbb{Z}$. We shall use the notation $\langle \boldsymbol{u}, \boldsymbol{v}\rangle$ to denote the inner product between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, defined as $\langle \boldsymbol{u}, \boldsymbol{v}\rangle = \sum_{i=1}^{n} \boldsymbol{u}_i \cdot \boldsymbol{v}_i$.

Furthermore, we shall be using number theoretic transformation (NTT) in this work. We shall denote the forward transformation as NTT and the inverse of the transformation as NTT^{-1}. We will denote ζ as the $2n$-th *primitive root of unity*. Since all polynomials are reduced modulo x^n+1, we use the negacyclic transformation.

We shall also be dealing with binary strings. For two binary strings A and B, we use $A \cdot B$, $A \oplus B$ to denote **bit-wise "and"** operation and **bit-wise "xor"** operation, respectively. Also, for any two vectors $A, B \in \mathbb{Z}^\ell$, we shall use the notation $A \odot B$ to denote the pointwise multiplication of A and B.

The operator $|\cdot|$ is used very often. For a set, S the notation $|S|$ denotes the cardinality of S, for a binary bit-string A, $|A|$ denotes the *Hamming weight* of A.

3 PRELIMINARIES

The security of public-key cryptography depends upon the hardness of some underlying computationally hard problem. There are several problems that are still presumed to be difficult to solve even with the help of quantum computers, such as finding the shortest vectors or finding a vector closest to a given vector in random lattices, finding pre-image of cryptographically secure hash functions, etc. Public-key cryptography, based on these hard problems, constitutes the current post-quantum cryptography. *Lattice-based cryptography* is one such post-quantum cryptography that is widely used today. Since this work focuses on the scheme based on RLWE problem, we discuss this problem only.

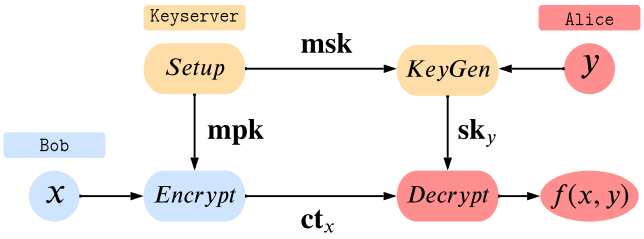


Fig. 1. Functional encryption and its different actors and operations.

3.1 Ring learning with errors

Learning with errors (LWE) was first introduced by Oded Regev [6], in the year 2004. RLWE is one variant of the LWE problem [7]. In this problem, the polynomial $a_i \in R_{\mathbf{q}}$ is sampled uniformly, secret $s \in R_{\mathbf{q}}$ is a small polynomial sampled uniformly or from a distribution χ and the error polynomials $e_i \in R_{\mathbf{q}}$ are sampled from a distribution χ. The RLWE distribution is defined as $(a_i, b_i = a_i \cdot s + e_i) \in R_{\mathbf{q}} \times R_{\mathbf{q}}$. We have two variants of RLWE problems.

Definition 3.1.1. (Search RLWE [7]) *Given polynomially many samples from the RLWE distribution, the problem is to find the secret $s \in R_{\mathbf{q}}$ with non-negligible probability.*

Definition 3.1.2. (Decisional RLWE [7]) *Given polynomially many samples (a_i, b_i) from RLWE distribution and equal number of samples (a_i, b_i') sampled uniformly from $R_{\mathbf{q}} \times R_{\mathbf{q}}$, the problem is to distinguish between these two distributions with non-negligible probability.*

3.2 Functional encryption

This section describes the *functional encryption* scheme as shown in Fig. 1 and one of its variants, *inner product functional encryption*.

Definition 3.2.1. (Functional encryption [3]) *A functional encryption is parameterized by $\rho = (X, Y, Z, f)$ for functionality $f : X \times Y \to Z$, is defined by the following four algorithms*

- **`Setup`**(1^κ) *:* **`Setup`** *receives a security parameter κ and outputs a pair of master public key and master secret key, which is* (**mpk**, **msk**).
- **`Encrypt`**(**mpk**, x) *:* **`Encrypt`** *receives a message $x \in X$ and the master public key* **mpk** *and returns a ciphertext* $\mathtt{ct}_x$.
- **`KeyGen`**(**msk**, y) *:* **`KeyGen`** *receives a $y \in Y$ and the master secret key* **msk** *and returns a functional key* $\mathtt{sk}_y$.
- **`Decrypt`**($\mathtt{ct}_x$, $\mathtt{sk}_y$) *:* **`Decrypt`** *receives a ciphertext* $\mathtt{ct}_x$ *and a functional key* $\mathtt{sk}_y$ *and outputs the value $f(x, y)$ or $\perp$ if the input is invalid.*

For *inner product functional encryption*, the functionality f is defined by $f(x, y) = \langle x, y \rangle$ where $x, y \in \mathcal{M}^\ell$, which is the message space.

3.3 Number theoretic transformation (NTT)

Number theoretic transformation is one of the most time-consuming component of most of the lattice-based cryptography schemes. This is defined over the ring of polynomials $R_p = \mathbb{Z}_p[x]/\langle \Phi(x) \rangle$. Here, $\Phi(x)$ is an irreducible polynomial in $\mathbb{Z}[x]$. This $\Phi(x)$ is the polynomial $x^n + 1$, n is an integer such that the primitive $2n$-th root of unity is in $\mathbb{Z}_p$. Let $\zeta \in \mathbb{Z}_p$ be the primitive $2n$-th root of unity. Then,

$$x^n + 1 = \prod_{i=0}^{n-1} (x - \zeta^{2i+1})$$

If we recall the Chinese remainder theorem (CRT) for rings, we can write

$$R_p \cong \prod_{i=0}^{n-1} \mathbb{Z}_p[x]/\langle x - \zeta^{2i+1} \rangle$$

This above isomorphism is defined by NTT, and the inverse of it is NTT^{-1}. Therefore, for any $f \in R_p$

$$\text{NTT}(f) = (f(\zeta), f(\zeta^3), \cdots, f(\zeta^{2n-1}))$$

So the i-th component of NTT(f) is

$$\hat{f}_i = \sum_{k=0}^{n-1} f_k \zeta^{(2i+1) \cdot k}$$

Similarly, i-th component of $\text{NTT}^{-1}(\hat{f})$ is calculated as

$$f_i = \frac{1}{n} \cdot \sum_{k=0}^{n-1} \hat{f}_k \zeta^{-(2i+1)\cdot k}$$

For two polynomials $f, g \in R_p$, we can compute the multiplication $h \in R_p$ in the following manner

$$h = \text{NTT}^{-1}(\text{NTT}(f) \odot \text{NTT}(g))$$

For NTT and NTT^{-1}, we have used the Cooley-Tukey [8] and Gentleman-Sande butterfly [9] respectively. These are very well known algorithms, so we are not explaining these algorithms here. Note that NTT takes input in standard order and outputs in bit-reversed order, and NTT^{-1} takes input in bit-reversed order and outputs in standard order. Therefore, after forward NTT we can leave the polynomials in the bit-reversed order and pointwise multiply them together, and later we can apply NTT^{-1} to get back the multiplication in standard order.

3.4 RLWE-IPFE scheme

As we have stated before, one of our goals in this work is to develop an efficient IPFE library based on RLWE-IPFE [5]. The RLWE-IPFE scheme by Mera et al. is the first IPFE scheme based on the RLWE assumption. Here we will only discuss the construction of the scheme, for more details on security proof and parameter selection of the scheme we kindly refer the reader to the original publication [5].

There are two notions of the security of *functional encryption*, namely *selectively secure* and *adaptively secure*. Here, we will only discuss the selectively secure IPFE as it is most relevant to our work, but one can easily extend the techniques described here to the adaptively secure scheme described in the original work.

3.4.1 Selectively secure RLWE-IPFE

Let x be a vector to be encrypted, of dimension ℓ, where $||x||_\infty \leq B_x$. The decryption should return $\langle x, y \rangle$, where $||y||_\infty \leq B_y$. Let K be greater than the maximum value of the resulting inner product *i.e.*, $K > \ell B_x B_y$. We describe the construction of the scheme below.

Construction

`Setup :`

(i) First sample $a \xleftarrow{R} R_{\mathbf{q}}$.
(ii) Sample $s_i, e_i \in R$ for $i \in [\ell]$, whose coefficients are sampled from the distribution $\mathcal{D}_{\sigma_1}$
(iii) Compute $\mathbf{pk}_i = a \cdot s_i + e_i \in R_{\mathbf{q}}$ for $i \in [\ell]$
(iv) Set, $\mathbf{msk} = \{s_i \mid i \in [\ell]\}$ and $\mathbf{mpk} = \big(a, \{\mathbf{pk}_i \mid i \in [\ell]\}\big)$

`Encrypt :` Given a vector $x = (x_1, x_2, \cdots, x_\ell) \in \mathbb{Z}^\ell$ with $||x||_\infty \leq B_x$, below is the encryption algorithm.

(i) First sample $r, f_0 \in R_{\mathbf{q}}$ with coefficients from the distribution $\mathcal{D}_{\sigma_2}$.
(ii) Sample $f_i \in R_{\mathbf{q}}$ independently with coefficients from the distribution $\mathcal{D}_{\sigma_3}$, for all $i \in [\ell]$.
(iii) Calculate $\mathbf{ct}_0 = a \cdot r + f_0$, $\mathbf{ct}_i = \mathbf{pk}_i \cdot r + f_i + \lfloor \mathbf{q}/K \rfloor x_i 1_R$ for all $i \in [\ell]$.
(iv) Output $\big(\mathbf{ct}_0, \{\mathbf{ct}_i \mid i \in [\ell]\}\big)$ as encryption of x.

`KeyGen :` Given a vector $y = (y_1, y_2, \cdots, y_\ell) \in \mathbb{Z}^\ell$ such that $||y||_\infty \leq B_y$, We need to generate a functional key corresponding to y. We simply calculate

$$\mathbf{sk}_y = \sum_{i=1}^{\ell} y_i s_i \in R$$

`Decrypt :` To decrypt the ciphertext $\big(\mathbf{ct}_0, \{\mathbf{ct}_i \mid i \in [\ell]\}\big)$ using the functional key, $\mathbf{sk}_y$ and y we calculate

$$d = \left(\sum_{i=1}^{\ell} y_i \mathbf{ct}_i \right) - \mathbf{ct}_0 \cdot \mathbf{sk}_y$$

This d should be close to $\lfloor \mathbf{q}/K \rfloor \langle x, y \rangle 1_R$, and we can extract $\langle x, y \rangle$ easily.

Correctness

We can write the decryption d as

$$\begin{aligned} d &= \left(\sum_{i=1}^{\ell} y_i \mathbf{ct}_i \right) - \mathbf{ct}_0 \cdot \mathbf{sk}_y \\ &= \sum_{i=1}^{\ell} (y_i e_i r + y_i f_i - y_i s_i f_0) + \lfloor \mathbf{q}/K \rfloor \langle x, y \rangle 1_R \end{aligned} \tag{1}$$

The term $\sum_{i=1}^{\ell}(y_i e_i r + y_i f_i - y_i s_i f_0)$ is the "noise". For correctness, we need the condition $||\texttt{noise}||_\infty < \lfloor \mathbf{q}/2K \rfloor$. For the security parameter κ, with non-negligible probability we have, $||e_i||_\infty, ||s_i||_\infty \leq \sqrt{\kappa}\sigma_1$, also $||r||_\infty, ||f_0||_\infty \leq \sqrt{\kappa}\sigma_2$ and $||f_i||_\infty \leq \sqrt{\kappa}\sigma_3$. Thus, we have

$$\left|\left| \sum_{i=1}^{\ell}(y_i e_i r + y_i f_i - y_i s_i f_0) \right|\right|_\infty \leq \ell(2n\kappa\sigma_1\sigma_2 + \sqrt{\kappa}\sigma_3)B_y$$

So for correctly extracting the result we choose suitable parameters such that $\ell(2n\kappa\sigma_1\sigma_2 + \sqrt{\kappa}\sigma_3)B_y < \lfloor \mathbf{q}/2K \rfloor$.

Since $||\texttt{noise}||_\infty$ is bounded by $\lfloor \mathbf{q}/2K \rfloor$, one can easily eliminate the noise and get back $\langle x, y \rangle 1_R$. From Eqn. (1) we have

$$d = \lfloor \mathbf{q}/K \rfloor \langle x, y \rangle 1_R + \texttt{noise}$$

To extract $\langle x, y \rangle 1_R$ we calculate, $\lceil \frac{K}{\mathbf{q}} \cdot d \rfloor$. Now,

$$\begin{aligned} \frac{K}{\mathbf{q}} \cdot d &= \frac{K}{\mathbf{q}} \cdot \left(\left\lfloor \frac{\mathbf{q}}{K} \right\rfloor \langle x, y \rangle 1_R + \texttt{noise} \right) \\ &= \langle x, y \rangle 1_R + \varepsilon \end{aligned}$$

Note that $||\texttt{noise}||_\infty < \lfloor \frac{\mathbf{q}}{2K} \rfloor$ implies that $||\varepsilon||_\infty < 1/2$, which means $\lceil \frac{K}{\mathbf{q}} \cdot d \rfloor$ gives us the $\langle x, y \rangle 1_R$. The symbol, $\lceil \cdot \rfloor$, represents the rounding function.

Unlike HE, where the noise or error grows with the depth of multiplication, in FE, we encounter with the noise at the time of decryption, and we can bound this noise by choosing the parameters carefully.

Parameters

The parameters for the original work are given in the Table. 1. Based on the security levels, there are three different sets of parameters of RLWE-IPFE. At the end of this work, we will be presenting an application of IPFE which requires message size $\ell = 2048$. This will change the parameters, and we give parameters for **Standard** and **High** security levels. Therefore, we will only discuss the performance improvements of **Standard** and **High** level of security. However, the improvement methods discussed in this work can also be applied to the **Low** security level for a significant performance improvement.

TABLE 1
The parameters of the RLWE-IPFE scheme.

Security Level (PQ security)	FE Bounds	Gaussian Parameters	Ring Parameters	CRT moduli (q_i)
Low (76.3)	B_x : 2 B_y : 2 ℓ : 64	σ_1 : 33 σ_2 : 59473921 σ_3 : 118947840	n : 2048 $\lceil \log \mathbf{q} \rceil$: 66	$2^{14} - 2^{12} + 1$ $2^{23} - 2^{17} + 1$ $2^{29} - 2^{18} + 1$
Standard (119.2)	B_x : 4 B_y : 16 ℓ : 785	σ_1 : 225.14 σ_2 : 258376412.19 σ_3 : 516752822.39	n : 4096 $\lceil \log \mathbf{q} \rceil$: 86	$2^{24} - 2^{14} + 1$ $2^{31} - 2^{17} + 1$ $2^{31} - 2^{24} + 1$
High (246.2)	B_x : 32 B_y : 32 ℓ : 1024	σ_1 : 2049 σ_2 : 5371330561 σ_3 : 10742661120	n : 8196 $\lceil \log \mathbf{q} \rceil$: 101	$2^{17} - 2^{14} + 1$ $2^{20} - 2^{14} + 1$ $2^{32} - 2^{20} + 1$ $2^{32} - 2^{30} + 1$

4 PERFORMANCE BOTTLENECKS AND PARALLELIZATION IN RLWE-IPFE

The authors in [5] provided a reference implementation of RLWE-IPFE. However, the implementation is not optimized, as most of the improvements are unexplored. Table. 2 shows the benchmark results of different operations of the RLWE-IPFE implementation provided by the authors of [5]. We observe that the **`Setup`** and **`Encrypt`** operations are the most computationally expensive operations of all. Therefore, we concentrate most of our optimization efforts on these two operations.

We illustrate the **`Setup`** and **`Encrypt`** operations in the figures Fig. 2 and Fig. 3 respectively. In each iteration of a for-loop, there are some tasks that are to be executed. For each iteration, we denote the collection of these tasks corresponding to the iteration as "block". The "blocks" inside each for-loop in the figures Fig. 2 and Fig. 3, does not have any conflicts among themselves. Even the "blocks" in the first and the second for-loop have no conflicts between them. These "blocks" could be distributed among multiple threads, and we parallelly compute parts of the desired result. In the end, these results are merged to get the final result. Similarly, we identify the "blocks" in **`KeyGen`** and **`Decrypt`** operations and distribute them among multiple threads.

TABLE 2
Performance table for different operation

		cpucycles ($\times 10^6$)
Standard	**Setup**	7821
	Encrypt	4835
	KeyGen	246
	Decrypt	227
High	**Setup**	34916
	Encrypt	25326
	KeyGen	977
	Decrypt	929

We have benchmarked the major components of each operation in Table. 3 for **Standard** security level. Similar results can be obtained for **High** security level. *Discrete Gaussian* and *polynomial multiplication* are the two main computationally intensive operations in both **Setup** and **Encrypt**. The authors of the RLWE-IPFE scheme already provided a highly optimized discrete Gaussian sampling routine using AVX2. Therefore, we target to optimize the polynomial multiplication.

TABLE 3
Workload distribution of major components within each operation

	Setup	**Encrypt**	**KeyGen**	**Decrypt**
Discrete Gaussian Sampling	15.97%	21.53%	-	-
Polynomial Multiplication	82.91%	72.66%	-	5.54%
Scalar Multiplication	-	-	99.97%	92.91%
CRT	-	-	0.03%	1.49%
Others[1]	1.12%	5.81%	-	0.06%

As we can see, the third for-loop in Fig. 2, the first and the second for-loop in Fig. 3 mostly involves NTT or NTT^{-1} operations. Hence, even a small improvement in the NTT operation will have a big impact on the overall efficiency of the whole scheme. The RLWE-IPFE scheme uses large moduli $\mathbf{q}$ that can range from 66-bits to 101-bits, as shown in Table. 1. For efficiency and to avoid multi-word arithmetic, the authors proposed a residue number system-based implementation. In this implementation, the modulus $\mathbf{q}$ is composed of many smaller prime numbers as $\mathbf{q} = q_1 \cdot q_2 \cdots q_t$. All the computations are done under these primes independently, and the final result is obtained by combining these

1. Polynomial addition, subtraction

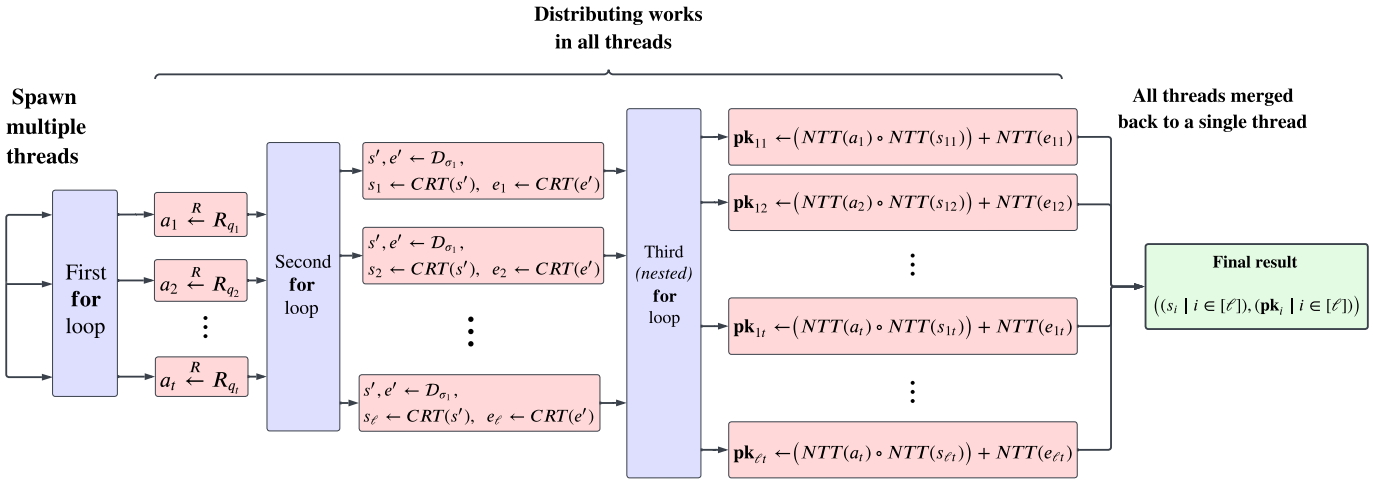


Fig. 2. The **Setup** operation and its parallel *for-loops*.

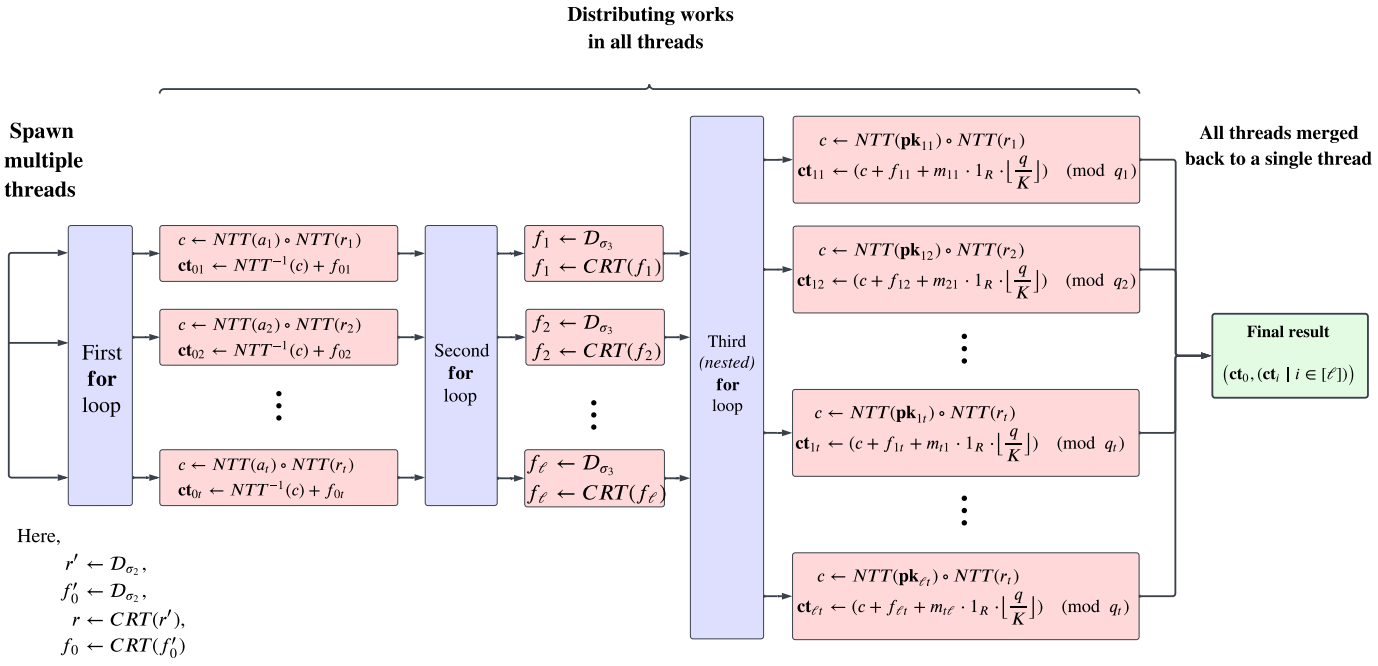


Fig. 3. The **`Encrypt`** operation and its parallel *for-loops*.

results using the Chinese remainder theorem. Hence, the polynomial multiplication in $R_{\mathbf{q}}$ is equivalent to t many NTT-based polynomial multiplication in R_{q_i} for each $i \in [t]$.

We took a two-step optimization strategy. First, on a macro level, we parallelly compute all the "blocks" inside the for-loops with *OpenMP* [10]. In the second step, on a micro level, we optimize each of the NTT-based polynomial multiplication using AVX2 vector processors. The reason for such a design choice is explained in the next section.

5 Improving efficiency of RLWE-IPFE

In this section, we will analyse the bottlenecks of RLWE-IPFE and provide a detailed description of our optimization technique.

5.1 OpenMP Optimization

OpenMP is an API that supports multi-threading programming in C, C++ and Fortran over many platforms. It includes compiler directives, library routines and environment variables that influence run-time behaviour. It uses a portable, scalable model that gives programmers a simple interface to build parallel codes for many platforms. With OpenMP, our target is to distribute the expensive components among multiple threads of the system, which gives a faster computation than the original implementation of RLWE-IPFE.

5.1.1 Parallelizable section of code

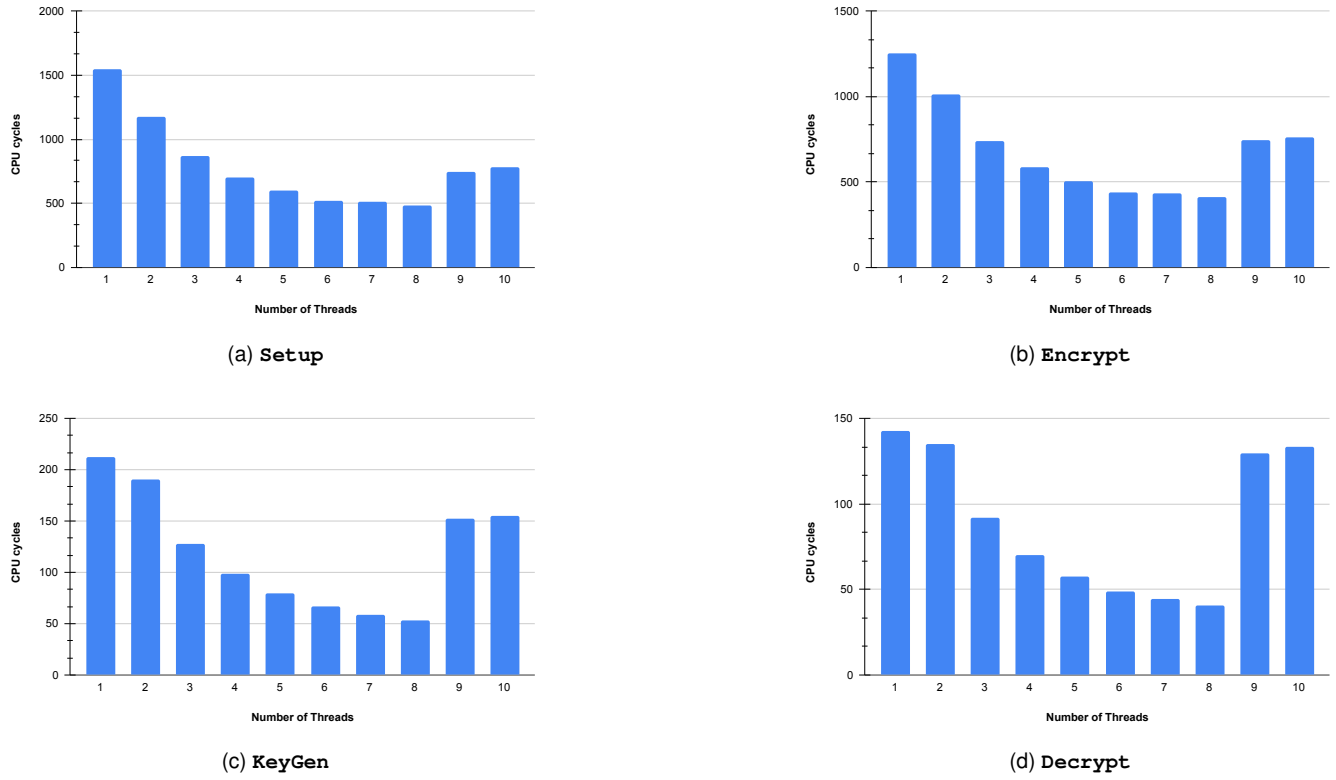
There are multiple ways to parallelize the RLWE-IPFE. one way is to distribute the "blocks" of each for-loop among multiple threads, as shown in figures Fig. 2 and Fig. 3. Also, we can parallelize the NTT-based polynomial multiplication. We list down our options below

(i) Parallelize the operations by distributing all the "blocks" of each for-loop among multiple threads as shown in Fig. 2 and Fig. 3 respectively.
(ii) Parallelize the NTT and NTT^{-1} using OpenMP.
(iii) We can use a hybrid approach. We can distribute "blocks" of each for-loop among m many threads, and each of these threads will spawn k many threads to parallelize the NTT transformation.

Now the question is, which approach is suitable for us? To answer this question, we look at Table. 4.

It is clear from the Table. 4, that the *approach* (iii) is not ideal. In this approach, each of the m many threads further spawns k many threads for NTT transformation. But spawning new threads incurs overhead which eclipses our performance gain.

To overcome this problem, it is better if we spawn all the threads required at the beginning of an operation and complete the whole operation inside the parallel region. This is what we can do with both *approach* (i) and *approach* (ii). But as we can see in Table. 4, parallelizing the "blocks" within the for-loops instead of parallelizing the polynomial multiplication gives us a better performance. Therefore, we use the *approach* (i) to optimize the implementation of RLWE-IPFE.



(a) **`Setup`** (b) **`Encrypt`**

(c) **`KeyGen`** (d) **`Decrypt`**

Fig. 4. Change in performance with different number of threads

TABLE 4
performance with different parallelization approach

Operations	cpucycles ($\times 10^6$)		
	Parallel component		Hybrid approach
	For-loops	NTT	
`Setup`	1960	2400	4974
`Encrypt`	1616	1731	3654

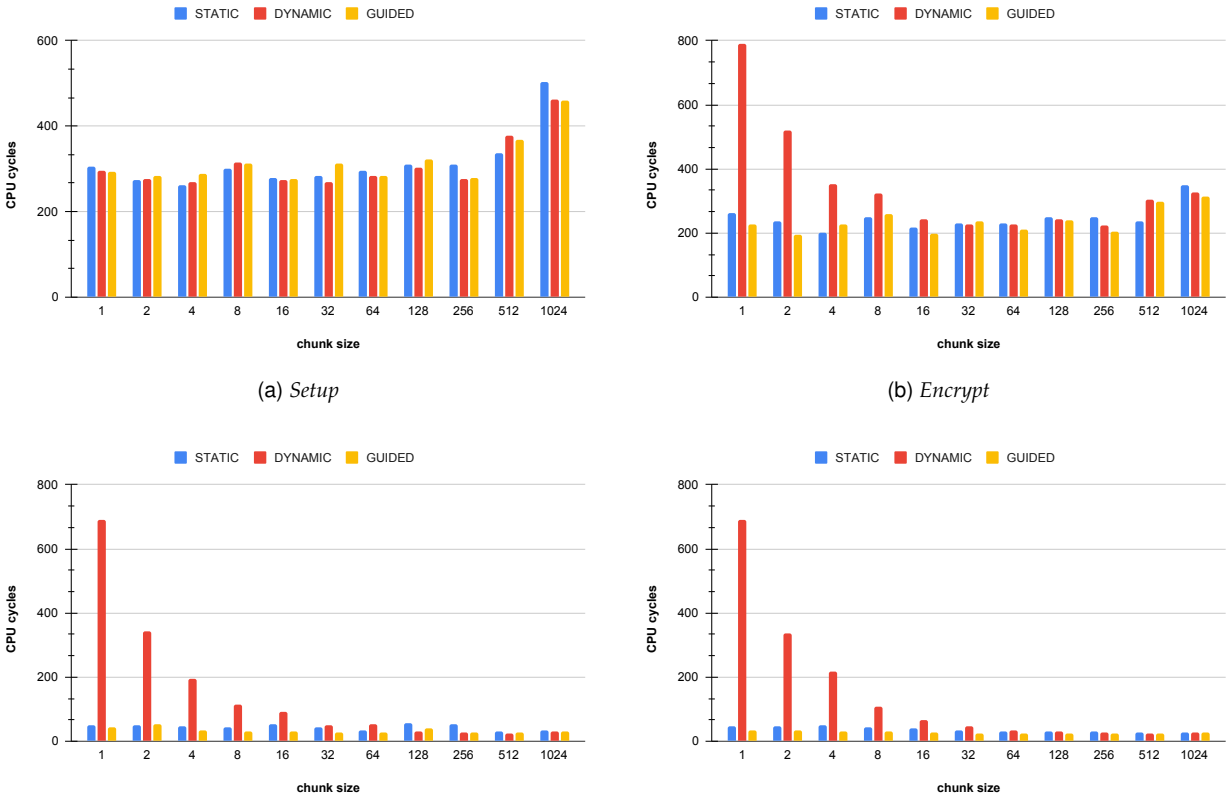
5.1.2 *Number of threads*

The number of cores in a system is limited, so spawning too many threads would increase the amount of context switching, which affects performance. Therefore, we need to find the number of threads that is optimal for our system. As we can see in Fig. 4, the performance of all the operations increases as the number of used threads increases. But after a certain point, the performance starts decreasing (Fig. 4) as the number of context switching increases. The experiments were run on quad-core Intel(R) Core(TM) i5-8265U CPU running at 1.60GHz and multi-core support as standard practice on Ubuntu 22.04. We have run these experiments on a few other machines with similar configurations and the result remains the same. However, the result may be different on a system with different configuration. Therefore, one may need to experimentally find the optimal number of threads.

5.1.3 *Scheduling*

In OpenMP, we are allowed to specify `schedule(type, chunk size)` clause with a specific type and chunk size for different types of scheduling. When we split a for-loop to distribute among threads, we split it into small chunks and these contain a specific number of "blocks" within the loop, which is called the chunk size. There are mainly three types of scheduling that we can use in OpenMP which are *static*, *dynamic*, and *guided*. See [11] for more details.

We have used different scheduling methods and different chunk sizes to see which is the most optimal setup for RLWE-IPFE. As we can observe from Fig. 5a, the **`Setup`** operation works better with smaller chunk sizes. As the chunk size gets larger, the operation starts slowing down. The type of schedule doesn't seem to make a difference in terms of performance. So we can use "static" scheduling with a small chunk size. We have used chunk size 8 for this operation. The **`Encrypt`** (Fig. 5b) operation has similar performance with chunk size 32 to 64 with any scheduling method, but it starts slowing down with increasing chunk size. On the other hand, the **`KeyGen`** and **`Decrypt`** operations (Fig. 5c, Fig. 5d) have optimal performance with larger chunk size.



(a) *Setup* (b) *Encrypt*

(c) *KeyGen* (d) *Decrypt*

Fig. 5. Change in performance with different scheduling method and chunk size

To utilize all the threads most optimally, we need to distribute all the works uniformly among threads. The "static" scheduling is the best option for the scheduling method, as all the "blocks" within one for-loop, which are to be distributed among threads, have similar workload. Therefore, we have used static scheduling with different chunk sizes for each operation. We have derived the chunk size for each operation experimentally. This chunk size may be different for a different system.

5.2 AVX2 Optimization

Advanced Vector Extensions (AVX) are extensions to the `x86` instruction set architecture for microprocessors from Intel and Advanced Micro Devices (AMD). They were proposed and first supported by Intel with the Sandy Bridge processor, and later by AMD with the Bulldozer processor. It supports registers of size 128-bits. AVX2 is an expansion of the AVX instruction set introduced in Intel's Haswell microarchitecture. AVX2 can support up to 256-bits registers. We can load eight 32-bits integers or four 64-bits integers, and the instructions can be applied to all of these packed integers simultaneously.

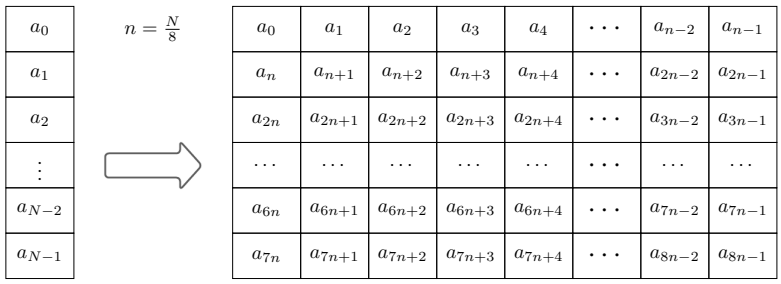


Fig. 6. Reshapeing an N length polynomial into an $8 \times \frac{N}{8}$ matrix

5.2.1 NTT Transformation

In this section, we will discuss the details of our AVX2 optimization of the NTT over 32-bit primes. AVX2 instructions support 16 vector registers (`ymm0-ymm15`) of size 256-bits each. Since all the integers are reduced in prime modulus of 32-bits, we can pack eight integers of 32-bits inside one register. We pack copies of q and $q^{-1} \pmod{2^{32}}$ in two different registers. Eight of the registers are used to store the coefficients of the polynomial, meaning at once we can operate on 64 many coefficients of the polynomial. Five registers are used to store temporary data and computational results. The powers of the primitive root of unity, also known as twiddle factors, are stored in a fixed table. One register is used to load the values of the twiddle factors. That is in total 16 registers in use.

As we can observe in Fig. 6, an N length polynomial can be viewed as an $8 \times \frac{N}{8}$ matrix. First, we apply 8-point NTT on each of the columns of the matrix in Fig. 6, which is equivalent to three levels of radix-2 forward NTT (Cooley-Tukey [8]) on the polynomial of length N. Instead of calling it three levels of radix-2 NTT, we will specify this step as one level of radix-8 NTT transformation. After this step, we need to apply forward NTT on each of the rows of the matrix in Fig. 6. But instead, we apply the mentioned method on each row of the matrix in Fig. 6, to compute another three levels of radix-2 NTT or one level of radix-8 NTT.

To implement the above in AVX2, first, we need to load each of the eight coefficients of one column into eight different registers. Since one register can pack eight 32-bits integers at once, we can load the first eight columns into eight AVX2 registers such that coefficients from the same row are packed into the same register. We apply 8-point NTT on these loaded coefficients. Next, we load the next eight columns of the matrix and apply 8-point NTT until we exhaust all the columns of the matrix in Fig. 6. This completes one level of radix-8 NTT.

NTT starts with a polynomial of length N and there are total $\log_2 N$ many levels in radix-2 NTT. After each level, the length of the polynomial is decreased by a factor of 2. More specifically, at level-ℓ, there are 2^ℓ many polynomials of length $N/2^\ell$ where $\ell \in \{0, 1, \cdots, \log_2 N\}$. Since one level of radix-8 NTT is equivalent to three levels of radix-2 NTT, after k levels of radix-8 NTT, the length of the polynomials becomes $N/8^k$. Observe that, eight registers assigned to load the coefficients, can pack 64 many integers at once. So when the length of the polynomials becomes 64, we pack all 64 coefficients of each polynomial into the eight assigned registers, at once, and complete the last 6 levels of radix-2 NTT transformation.

For example, given a polynomial of length 4096, we can first apply 2 levels of radix-8 NTT as explained above, which decreases the length of the polynomials to 64. After this point, we can load each polynomial into the registers one at a time and complete the rest of the transformation. On the other hand, given a polynomial of length 8192, the first two levels of radix-8 NTT decrease the length of each polynomial to 128. So we apply one level of radix-2 NTT transformation to reduce the size of polynomials to 64 and after that, the operations are the same.

Now, we deal with polynomials of length 64 for the last six levels. The 64 many coefficients of a polynomial can be reshaped into a matrix of size 8×8 as shown in Fig. 6. This matrix can be packed into the 8 many AVX2 vector registers easily. First, we apply one level of radix-8 NTT, as we explained before. Now, for the last three levels of radix-2 NTT, exactly half of the coefficients packed in each register are to be multiplied with twiddle factors. We can deal with this problem by grouping half of the coefficients that are to be multiplied together and loading them into four registers, and the rest of the coefficients are grouped and loaded into the other four registers. We have used the trick explained in [12], we use permutations to shuffle coefficients and group them and after the transformation, we again shuffle them back to their original position.

To implement the NTT^{-1} in AVX2, we have used the same idea as above, but in the reverse order.

5.2.2 Modular reductions

Modular reductions are one of the essential parts of NTT transformation for fast vectorized implementation. Divisions are not constant time, and it is inefficient in terms of performance. So divisions need to be avoided and replaced with fast, constant time modular reduction.

In NTT and NTT^{-1}, the coefficients of the polynomials are to be multiplied by the twiddle factors. We use the Montgomery modular multiplication [13] for the multiplication inside NTT. But the problem with Montgomery modular multiplication is that it returns $ab \cdot \beta^{-1} \pmod{q}$ when we reduce the multiplication of a and b, where β is a constant. Since we can precompute the twiddle factors and store them, then we can multiply these twiddle factors with β before storing. This way, we can ensure the correct result for the multiplication in NTT and NTT^{-1}. We have selected $\beta = 2^{32}$ for Montgomery modular multiplication.

5.3 Performance

All experiments in this section are performed on a quad-core Intel(R) Core(TM) i5-8265U processor running at 1.60GHz and multi-core support as standard practice on Ubuntu 22.04. All codes have been compiled using `gcc-11.2.0` with flags `-O3 -fomit-frame-pointer -march=native`.

TABLE 5
Performance table with AVX2 optimization

		cpucycles ($\times 10^6$)		
		Original	AVX2	PI
Standard	**`Setup`**	7821	1898	75.73%
	`Encrypt`	4835	1766	63.47%
High	**`Setup`**	34916	5241	84.99%
	`Encrypt`	25326	8853	65.04%

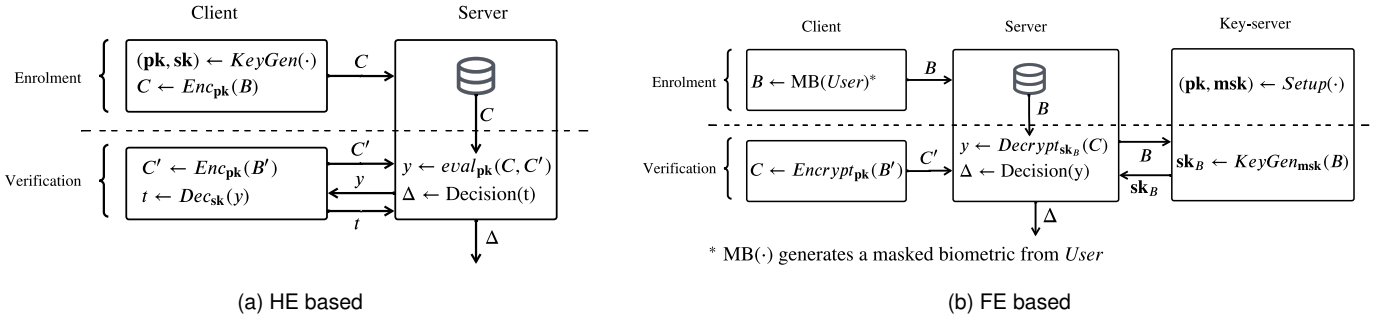
TABLE 6
Performance table with AVX2 + OpenMP optimization

		cpucycles ($\times 10^6$)		
		Original	AVX2+OpenMP	PI
Standard	**`Setup`**	7821	804	89.72%
	`Encrypt`	4835	819	83.06%
	`KeyGen`	246	100	59.30%
	`Decrypt`	227	105	53.80%
High	**`Setup`**	34916	2148	93.85%
	`Encrypt`	25326	3297	86.98%
	`KeyGen`	977	312	68.07%
	`Decrypt`	929	355	61.79%

After the AVX2 optimization, we can see a significant speed-up in Table. 5 for the operations **`Setup`** and **`Encrypt`**. As the number of polynomial multiplication is not significantly high in **`KeyGen`** and **`Decrypt`** algorithm, we do not see any significant speed-up here. In Table. 5, we have shown the performance gain of the operations in different security levels in the column named PI.

We get further improvements in the performance after parallelizing the implementation with OpenMP. In Table. 6, the performance gain for OpenMP optimization along with AVX2 is given.

6 Related works



(a) HE based

(b) FE based

Fig. 7. Biometric authentication protocols

The use of cryptography or more specifically COED techniques in the context of privacy-preserving biometric authentication is a continuously growing field of study. There are instances of privacy-preserving biometric authentication protocols based on homomorphic encryption (HE) [14], [15], [16], [17]. A typical homomorphic encryption-based biometric authentication protocol is given in Fig. 7a. Here, the client generates the public key and secret key pair. This implies, after evaluation of some operation on the ciphertexts, the client has to perform the final decryption, based on which server makes its decision. The server or any third party does not get any information about the plaintext. There are also instances of biometric authentication protocol based on predicate encryption (PE) [18]. But in PE, the decryption returns the plaintext if the predicate evaluates to true.

On the other hand, functional encryption (FE) is fundamentally different from HE and it is a more general form of PE. FE allows one to get information about only a functional value, rather than the complete decrypted plaintext. In FE, the client can encrypt its data and the server or any third party can get the result of some function computed on the client's data. This allows the server to decrypt and make a decision without disclosing the private biometric information of the client. This property of FE has been used in the biometric authentication protocols in [19], [20]. In these works, the authors propose biometric authentication protocols based on the matching algorithm, which is based on Hamming distance. However, we propose a protocol based on the biometric matching algorithm in [21], which uses fractional Hamming distance also known as normalized Hamming distance.

7 Privacy-preserving biometric authentication protocol

In this part, we propose a secure privacy-preserving biometric authentication protocol. Although there are many types of biometrics, fingerprint and iris image are primarily used as biometrics. Here we are using iris as the biometric data. We use the iris matching algorithm in [21]. In [22], the authors have also mentioned a similar matching algorithm.

7.1 Normalized Hamming Distance

In [21], the algorithm transforms an iris image into a 2048-bit binary string, this is called the *feature vector*. A *mask* of the same length is also generated, which denotes the positions of valid bits in the feature vector. To match two different sets of biometric data, we use *Normalized Hamming Distance* (NHD).

Definition 7.1.1 (Normalized Hamming distance/fractional Hamming distance [21]). *Let X, M and X', M' be pairs of feature vector and mask of two biometric data then the* normalized Hamming distance *between the two biometric data is defined by*

$$\text{NHD} = \frac{|(X \oplus X') \cdot M \cdot M'|}{|M \cdot M'|} \tag{2}$$

Since we are trying to implement the biometric authentication protocol using IPFE, so first we need to find a method to map NHD into a form that is suitable with the IPFE protocol.

7.2 Transformation of the binary vector and some results

In [23], the authors have given a method to calculate the Hamming distance of two binary vectors using the inner product. We use this method and extend it to calculate the NHD. First, we consider the vector $\widetilde{X} = (\widetilde{x}_1\widetilde{x}_2\cdots\widetilde{x}_n)$, derived from the binary vector $X = (x_1x_2\cdots x_n)$ of length n, using the transformation $\widetilde{x}_i = (-1)^{x_i}, \forall i \in [n]$.

Let us consider the operation "$\odot$" to be the pointwise multiplication operation between two vectors of the same length. Therefore, for any two binary vectors A and B, if $C = A \odot B$ then $\langle A, B\rangle = \sum_{i=1}^{n} c_i$ where, $C = (c_1c_2\cdots c_n)$. Before calculating the NHD using the inner product, we first look at some results.

Result 1. *If $A = (a_1a_2\cdots a_n)$ and $B = (b_1b_2\cdots b_n)$ are two binary vectors of length n then, $\widetilde{A} \odot \widetilde{B} = (\widetilde{A \oplus B})$*

Proof. First, consider two binary bits a, b. For a, b we can write $\widetilde{a} \odot \widetilde{b} = (-1)^{a\oplus b} = \widetilde{a \oplus b}$. Then, $\widetilde{a_i} \odot \widetilde{b_i} = (\widetilde{a_i \oplus b_i})$ for all $i \in [n]$. Therefore, $\widetilde{A} \odot \widetilde{B} = (\widetilde{A \oplus B})$ □

Result 2. *for two binary vectors $A = (a_1a_2\cdots a_n)$ and $B = (b_1b_2\cdots b_n)$ of length n, we can write,*

$$|A \cdot B| = \frac{1}{2}\big(|B| - \langle\widetilde{A}, B\rangle\big)$$

Proof. First, we take a vector $C = (c_1c_2\cdots c_n)$ such that $C = \widetilde{A} \odot B$ then, $\langle\widetilde{A}, B\rangle = \sum_{i=1}^{n} c_i = R_1 - R_2$. The values of R_1 and R_2 are the numbers of 1's and -1's respectively, in the string C. Again we have, $|B| = R_1 + R_2$. Observe that, $R_2 = |A \cdot B|$. Therefore,

$$|B| - \langle\widetilde{A}, B\rangle = 2R_2 = 2\,|A \cdot B|$$

Therefore, $|A \cdot B| = \frac{|B| - \langle\widetilde{A}, B\rangle}{2}$ □

7.3 Calculating NHD using inner product

With the results in Section. 7.2, we are now going to calculate NHD with the inner product. Let X, X' be two binary feature vectors and M, M' be the corresponding binary masks of the biometric data. We know the NHD is defined as it is shown in Eqn. (2). Now, from Result. 2 we can write

$$|(X \oplus X') \cdot M \cdot M'| = \frac{1}{2} \cdot \Big(|M \cdot M'| - \langle\widetilde{X \oplus X'}, M \cdot M'\rangle\Big)$$

We know that, for any pair of binary vectors A and B, we have $|A \cdot B| = \langle A, B\rangle$. Therefore, $|M \cdot M'| = \langle M, M'\rangle$. From Eqn. (2), we get

$$\text{NHD} = \frac{\langle M, M'\rangle - \langle\widetilde{X \oplus X'}, M \cdot M'\rangle}{2 \cdot \langle M, M'\rangle} \tag{3}$$

We are only using inner product to calculate this NHD, so we can use the RLWE-IPFE scheme here to implement the privacy-preserving biometric authentication protocol.

7.4 Proposed protocol

In this part, we will discuss the protocol. We consider three entities here, *a) client*, *b) server* and *c) keyserver*. There are two parts of the protocol, first is the *enrolment* and then comes the *matching*.

Enrolment

First, the *client* has to send its biometric data (X, M) to the *server* in some hidden form, so that it can be used in the matching stage. The *client* generates two one-time pad keys K_x, K_m. These keys are used to initially encrypt the vectors. The *client* computes $(S, T) = (X \oplus K_x, M \oplus K_m)$, and sends the pair (S, T) to the *server's* database at the time of enrolment. Note that the *client* has to send the template (S, T) to the *server* through a *secure channel*[2] so that a third party cannot intersect the data. At the time of matching, the *client* has to come up with the keys K_x, K_m along with a fresh biometric template (X', M') (not necessarily same as X and M).

2. This can be achieved by using any secure encryption scheme. This is important for the sake of security, which we will explain later.

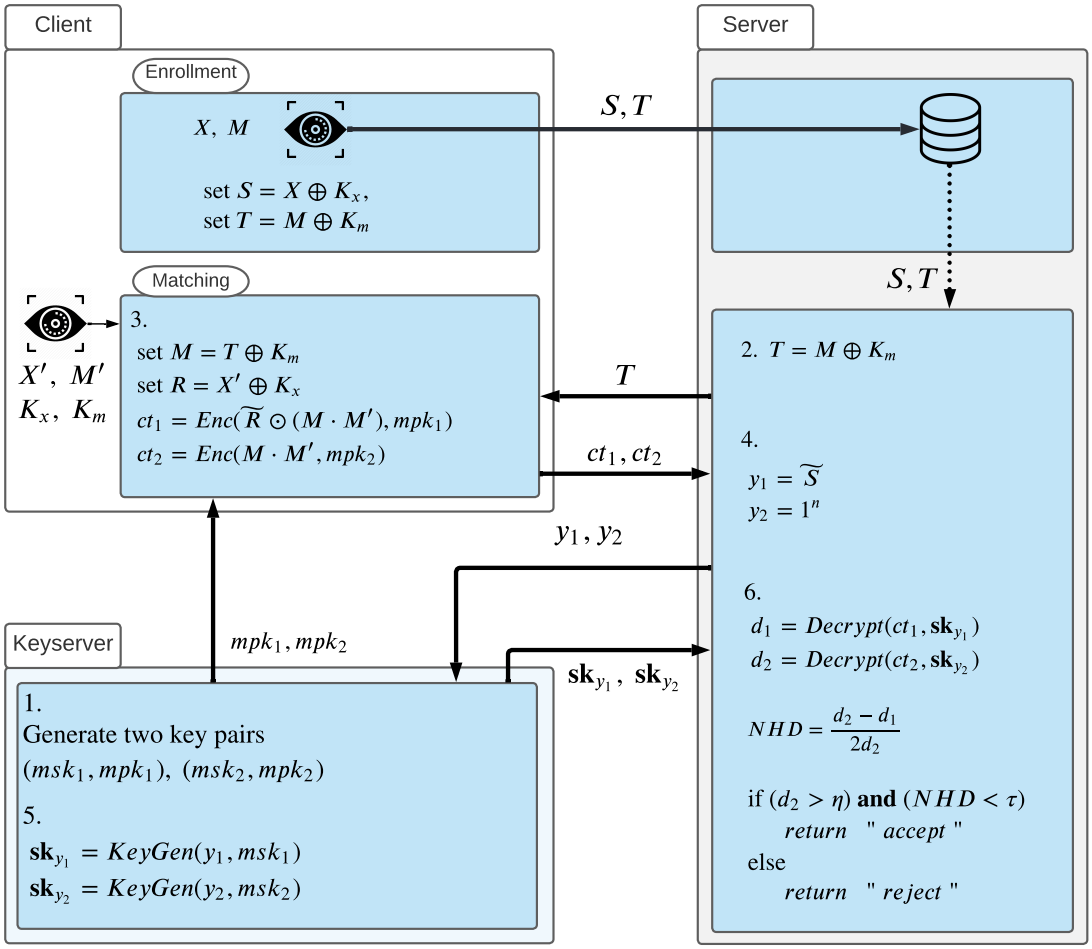


Fig. 8. *Proposed privacy-preserving biometric authentication protocol*

Matching

Before the matching protocol starts, the *client* already holds the keys K_x and K_m and the *server* has $(S, T) = (X \oplus K_x, M \oplus K_m)$ in its database.

(a) The *keyserver* creates two pairs of *master secret key* and *master public key*, namely, (msk_1, mpk_1) and (msk_2, mpk_2).

(b) *Server* sends $T = M \oplus K_m$ to the client.

(c) Since the *client* has correct keys K_m, K_x, it computes $M = T \oplus K_m$, and it also computes $R = X' \oplus K_x$.

(d) The *Client* computes $ct_1 = Enc(\widetilde{R} \odot (M \cdot M'), mpk_1)$ and $ct_2 = Enc(M \cdot M', mpk_2)$ and sends these to the *server*.

(e) The *Server* sets $y_1 = \widetilde{S}$ and $y_2 = 1^n$ and sends to the *keyserver* through a *secure channel*.

(f) *Keyserver* generates *functional keys*
$\mathbf{sk}_{y_1} = \texttt{KeyGen}(y_1, msk_1)$,
$\mathbf{sk}_{y_2} = \texttt{KeyGen}(y_2, msk_2)$
and sends these to the *server*.

(g) *Server* computes $d_1 = Dec(ct_1, \mathbf{sk}_{y_1})$ and
$d_2 = Dec(ct_2, \mathbf{sk}_{y_2})$.

(h) *Server* computes, NHD $= \frac{d_2 - d_1}{2d_2}$.

(i) If NHD $< \tau$ and $d_2 > \eta$ then we *accept* the biometric and the client passes the protocol, Otherwise we *reject* the biometric.

The values τ and η are thresholds for NHD and d_2 respectively, which determines the acceptance rate and security of the

biometric protocol. The threshold $\tau < 1$, as the value of NHD, cannot be greater than 1. The thresholds τ and η are fixed in the protocol.

7.5 Soundness and Completeness

We now show our proposed protocol is *sound* and *complete* if the underlying iris matching algorithm is *sound* and *complete*. To show this, we have to show that our protocol correctly computes the NHD between the biometric template saved in the database of the *server* and the freshly provided template at the *client* side.

In the protocol showed in Fig. 8 we have ciphertexts

$$ct_1 = Enc(\widetilde{R} \odot (M \cdot M'), mpk), \quad ct_2 = Enc(M \cdot M', mpk)$$

Given $y_1 = \widetilde{S}$ and $y_2 = 1^n$ we have the functional keys $\mathbf{sk}_{y_1}$ and $\mathbf{sk}_{y_2}$ using `KeyGen` algorithm. After decryption, we have,

$$d_1 = \langle \widetilde{R} \odot (M \cdot M'),\ \widetilde{S} \rangle, \quad d_2 = \langle M \cdot M',\ 1^n \rangle$$

Clearly, $d_2 = \langle M, M' \rangle$ and $d_1 = \sum_{i=1}^{n} B_i$ where

$$\begin{aligned} B &= \widetilde{R} \odot (M \cdot M') \odot \widetilde{S} \\ &= \widetilde{(R \oplus S)} \odot (M \cdot M') \quad \big[\text{ from Result. 1}\big] \end{aligned}$$

Therefore, $d_1 = \langle \widetilde{R \oplus S}, M \cdot M' \rangle$. Substituting R and S with $X' \oplus K_x$ and $X \oplus K_x$ respectively, we get the value of d_1 as $\langle \widetilde{X \oplus X'}, M \cdot M' \rangle$. Therefore, the *server* calculates

$$\text{NHD} = \frac{d_2 - d_1}{2 \cdot d_2} = \frac{\langle M, M' \rangle - \langle \widetilde{X \oplus X'}, M \cdot M' \rangle}{2 \cdot \langle M, M' \rangle}$$

This is exactly what we have derived in Eqn. (3). Therefore, in the end, the *server* can correctly calculate NHD between the biometric saved in the database and the biometric provided by the *client* at the time of matching.

7.6 Security analysis

The security of the protocol depends on its design and also the underlying cryptographic schemes. In this section, we will analyse the security of this protocol. Here we will assume that any one of *server* or *client* can act as an adversary, and the *keyserver* is honest. Also, we will assume that no two of the *server*, *client* or *keyserver* will act as one entity.

7.6.1 Server security

If an adversary impersonates *server*, then the biometric templates sent to the *server's* database are already encrypted with keys generated by *client*, at the time of enrolment. Therefore, the adversary cannot extract *client's* biometric information.

At the time of matching, the *server* can only receive encrypted data from the *client*. So the adversary does not get any extra information from the *client* other than the normalized Hamming distance between the biometric template at the *server's* database and the biometric template provided by the *client*.

TABLE 7
Updated parameters suitable for biometric authentication scheme

Security Level (PQ Security)	FE Bounds		Gaussian Parameters		Ring Parameters	CRT moduli (q_i)
Standard (125)	B_x	2	σ_1	226	n : 4096	$2^{24} - 2^{14} + 1$
	B_y	2	σ_2	258376413		$2^{29} - 2^{18} + 1$
	ℓ	2048	σ_3	516752823	$\lceil \log \mathbf{q} \rceil$: 83	$2^{30} - 2^{18} + 1$
High (305)	B_x	2	σ_1	2049	n : 8192	$2^{17} - 2^{14} + 1$
	B_y	2	σ_2	5371330561		$2^{20} - 2^{14} + 1$
	ℓ	2048	σ_3	10742661120	$\lceil \log \mathbf{q} \rceil$: 85	$2^{23} - 2^{20} + 1$
						$2^{25} - 2^{14} + 1$

7.6.2 Client security

An adversary impersonating the *client* cannot carry out the authentication without access to both the valid biometric template and the key generated by the actual *client*, at the time of enrolment. Now, if the adversary tries to randomly guess the biometric template to authenticate, then we show that the probability of the adversary being successfully *accepted* by the *server* is at most $\frac{1}{2^{(1-\tau)\eta}}$

Observe that, at the time of *matching*, the contents of the binary strings M, M' are in the hand of the *client*. Therefore, the adversary impersonating the *client* can easily manipulate these strings. We consider that the binary strings M, M' satisfies

$|M \cdot M'| = k$. Since the adversary is randomly guessing the biometric template, then we can say that R is a random binary string. At the end of the protocol, *server* calculates

$$d_1 = \langle \widetilde{R \oplus S}, M \cdot M' \rangle, \quad d_2 = |M \cdot M'|$$

Now, from Result. 2 we have,

$$|(R \oplus S) \cdot M \cdot M'| = \frac{d_2 - d_1}{2}$$

So the calculated NHD is

$$\frac{d_2 - d_1}{2d_2} = \frac{|(R \oplus S) \cdot M \cdot M'|}{|M \cdot M'|} = \frac{|(R \oplus S) \cdot M \cdot M'|}{k}$$

Let us consider, $\rho = |(R \oplus S) \cdot M \cdot M'|$. Then the *server* accepts the biometric provided by the adversary if $\frac{\rho}{k} < \tau$ and $k > \eta$. Since $(R \oplus S)$ is a random binary string, then

$$\Pr[\rho < \tau k] < \frac{1}{2^{(1-\tau)k}} < \frac{1}{2^{(1-\tau)\eta}}$$

Therefore, we need to choose the parameters τ and η such that $2^{(\tau-1)\eta}$ is negligible.

7.6.3 Network security

An adversary eavesdropping over the network can only access the encrypted biometric data and the functional keys. This will not reveal any useful information to the adversary about the *client's* biometric. However, one may use the data gathered by eavesdropping over the network, to successfully impersonate the *client*. Observe that, if the *client* sets, $R = S$ then, that will be accepted by the *server* as authentic biometric information without the knowledge of K_x. Therefore, we must not allow the adversary to get access to the string $S = X \oplus K_x$. This is why the encrypted biometric template (S, T) has to be sent to the *server's* database through a secure channel, at the enrolment stage. Also, at the time of matching, $y_1 = \widetilde{S}$ has to be sent to the *keyserver* through a secure channel for the same reason.

7.7 Practical Implementation

The size of the biometric information is 2048-bits. The original IPFE scheme in [5] has set its parameters for message size less than 2048-bis. So the parameters of the scheme are changed to support the protocol. In Table. 7, we have given two sets of parameters for **Standard** and **High** level of security, respectively. We have calculated the parameters and the security using the LWE estimator in [24], which has also been used for the original work. Table. 8 gives the execution time for each of the operations executed by *keyserver*, *client*, and *server*. We have used the same system that is mentioned in Section. 5.3 to get the experimental results in Table. 8

TABLE 8
Execution time of different operations involved in biometric authentication protocol

Security Level	Execution time (ms)			
	Keyserver		*Client*	*Server*
	`Setup`	`KeyGen`	`Encrypt`	`Decrypt`
Standard	2600.56	183.26	2721.66	154.66
High	5268.51	537.53	9074.00	576.86

8 Conclusion

We have presented an efficient implementation of the RLWE-IPFE library using multi-core architecture and vector instructions. We demonstrated that through several implementation techniques using OpenMP and AVX2, RLWE-IPFE can be accelerated. This makes RLWE-IPFE one of the candidates of the FE family for practical application. In this work, we have presented one such real-world application of IPFE. Our proposed privacy-preserving biometric authentication protocol shows the versatility and usefulness of FE. There are many existing privacy-preserving biometric authentication protocols based on HE as well as FE. The protocols based on HE and FE are fundamentally different and work on two different modes of operation. Therefore, a direct comparison between the HE-based protocols and FE-based protocols is not very fair. However, It will be an interesting work to set a basis of comparison for measuring the practicality of these protocols, which needs further deliberation. Finally, We hope that our work will attract more attention from the cryptographic community toward the design, implementation, and application of FE schemes.

REFERENCES

[1] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Functional encryption with bounded collusions via multi-party computation," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 162–179. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_11

[2] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 40–49. [Online]. Available: https://doi.org/10.1109/FOCS.2013.13

[3] M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *Public-Key Cryptography – PKC 2015*, J. Katz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 733–751.

[4] S. Agrawal, B. Libert, and D. Stehlé, "Fully secure functional encryption for inner products, from standard assumptions," in *Advances in Cryptology – CRYPTO 2016*, M. Robshaw and J. Katz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 333–362.

[5] J. M. B. Mera, A. Karmakar, T. Marc, and A. Soleimanian, "Efficient lattice-based inner-product functional encryption," in *Public-Key Cryptography – PKC 2022*, G. Hanaoka, J. Shikata, and Y. Watanabe, Eds. Cham: Springer International Publishing, 2022, pp. 163–193.

[6] O. Regev, "New lattice-based cryptographic constructions," *J. ACM*, vol. 51, no. 6, p. 899–942, Nov. 2004. [Online]. Available: https://doi.org/10.1145/1039488.1039490

[7] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology – EUROCRYPT 2010*, H. Gilbert, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23.

[8] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[9] W. M. Gentleman and G. Sande, "Fast fourier transforms: For fun and profit," in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, ser. AFIPS '66 (Fall). New York, NY, USA: Association for Computing Machinery, 1966, p. 563–578. [Online]. Available: https://doi.org/10.1145/1464291.1464352

[10] L. Dagum and R. Menon, "openmp: an industry standard api for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[11] "OPENMP API Specification: Version 5.1." [Online]. Available: https://www.openmp.org/spec-html/5.1/openmp.html

[12] G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography," Cryptology ePrint Archive, Report 2018/039, 2018, https://eprint.iacr.org/2018/039.

[13] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[14] G. Pradel and C. Mitchell, "Privacy-preserving biometric matching using homomorphic encryption," 2021. [Online]. Available: https://arxiv.org/abs/2111.12372

[15] W. Yang, S. Wang, K. Yu, J. J. Kang, and M. N. Johnstone, "Secure fingerprint authentication with homomorphic encryption," in *2020 Digital Image Computing: Techniques and Applications (DICTA)*, 2020, pp. 1–6.

[16] R. Kulkarni and A. Namboodiri, "Secure hamming distance based biometric authentication," in *2013 International Conference on Biometrics (ICB)*, 2013, pp. 1–6.

[17] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Packed homomorphic encryption based on ideal lattices and its application to biometrics," in *Security Engineering and Intelligence Informatics*, A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, and L. Xu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 55–74.

[18] K. Zhou and J. Ren, "Passbio: Privacy-preserving user-centric biometric authentication," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 12, pp. 3050–3063, 2018.

[19] J. Lee, D. Kim, D. Kim, Y. Song, J. Shin, and J. H. Cheon, "Instant privacy-preserving biometric authentication for hamming distance," Cryptology ePrint Archive, Paper 2018/1214, 2018, https://eprint.iacr.org/2018/1214. [Online]. Available: https://eprint.iacr.org/2018/1214

[20] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *Security and Cryptography for Networks*, D. Catalano and R. De Prisco, Eds. Cham: Springer International Publishing, 2018, pp. 544–562.

[21] J. Daugman, "How iris recognition works," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 21–30, 2004.

[22] R. Wildes, "Iris recognition: an emerging biometric technology," *Proceedings of the IEEE*, vol. 85, no. 9, pp. 1348–1363, 1997.

[23] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," Cryptology ePrint Archive, Report 2016/440, 2016, https://eprint.iacr.org/2016/440.

[24] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer, "Estimate all the {lwe, ntru} schemes!" in *Security and Cryptography for Networks*, D. Catalano and R. De Prisco, Eds. Cham: Springer International Publishing, 2018, pp. 351–367.