

DeV-IP: A k -out- n Decentralized and verifiable BFV for Inner Product evaluation*

Jose Contreras¹ and Hardik Gajera²

¹ Humanode,
tanogedler@humanode.io

² Humanode,
hgajera@humanode.io

Abstract

The biometric system has become the desired alternative to a knowledge-based authentication system. An authentication system does not provide uniqueness, as a single user can create multiple registrations with different identities for authentication. Biometric authentication identifies users based on physical traits (fingerprint, iris, face, voice), which allows the system to detect multiple authentications from the same user. The biometric templates must be encrypted or hidden to preserve users' privacy. Moreover, we need a system to perform the matching process over encrypted data without decrypting templates to preserve the users' privacy. For the euclidean distance-based matching process, centralized server-based authentication leads to possible privacy violations of biometric templates since the power of computing inner product value over any two encrypted templates allows the server to retrieve the plain biometric template by computing a few inner products. To prevent this, we considered a decentralized system called collective authority, which is a part of a public network. The collective authority computes the collective public key with contributions from all nodes in the collective authority. It also performs a matching process over encrypted biometric templates in a decentralized manner where each node performs partial matching. Then the leader of the collective authority combines it to get the final value. We further provide a lattice-based verification system for each operation. Every time a node performs some computations, it needs to provide proof of the correctness of the computation, which is publicly verifiable. We finally make the system dynamics using Shamir's secret sharing scheme. In dynamic collective authority, only k nodes out of the total n nodes are required to perform the matching process. We further show that the security of the proposed system relies on the security of the underlying encryption scheme and the secret sharing scheme.

1 Introduction

Due to advancements in the Internet of Things (IoT) and digitization of systems, there has been an increase in interaction with mobile devices in our day-to-day life. There are several authentication methods to prevent any unauthorized access to any personal device knowledge-based authentication (PIN, password), smart-card-based authentication, context-aware (location, IP address), and biometric-based authentication (fingerprint, iris, voice, face), among others. [13]. In a service-based system, a user interacts with a service provider through a personal device like a laptop, smartphone, tablet, or smart device. In this scenario, user authentication can be performed either locally on the user's device or remotely on the service provider platform [17].

In a system where the uniqueness of a user is necessary, biometric authentication is used over traditional password-based authentication. Biometric authentication has several advantages over password-based authentication. There is no requirement to keep the secret key

*Reviewed by Humanode Core team, MOZGIII, and Dmitry Lavrenov.

safe and secure; challenging to forge (Sybil resistant) and hard to guess. However, biometric authentication has an intrinsic limitation as well. Password-based authentication requires an exact match in the input string and stored password, while biometric authentication relies on the closeness between two biometric feature vectors. This closeness can be based on either euclidean distance, cosine similarity, or hamming weight [20]. Here, we consider euclidean distance to compare feature vectors.

In remote biometric authentication, a server must store the biometric templates of all the users. In database leakage, an adversary may obtain a valid user’s template. The adversary can use the template for some other biometric authentication-based application or try to obtain the user’s biometric data by reverse engineering the template. It is essential to keep the biometric templates secure so that no one other than the user can access their plaintext form. On the other hand, we also want to check the closeness between two biometric templates for authentication purposes, leading to a system with the following requirements:

- The biometric template must not leave the user’s device plain, and no other party should be able to retrieve the template.
- System should be able to check the closeness of two encrypted templates without decrypting any of the templates.

In this paper, we are considering the euclidean distance between two templates as a closeness property. Euclidean distance computation is a non-linear function, but we can write it as an inner product. Therefore, it is enough to have a system to compute the inner product over encrypted templates. Let $\mathbf{u} = (u_1, u_2, \dots, u_n)$ and $\mathbf{v} = (v_1, v_2, \dots, v_n)$ be two n-dimensional vectors. Then we have

$$\begin{aligned} (d(\mathbf{u}, \mathbf{v}))^2 &= (u_1 - v_1)^2 + (u_2 - v_2)^2 + \dots + (u_n - v_n)^2 \\ &= (u_1^2 + u_2^2 + \dots + u_n^2) + (v_1^2 + v_2^2 + \dots + v_n^2) - 2(u_1v_1 + u_2v_2 + \dots + u_nv_n) \\ &= \mathbf{u} \cdot \mathbf{u} + \mathbf{v} \cdot \mathbf{v} - 2\mathbf{u} \cdot \mathbf{v} \end{aligned}$$

Assume we have a centralized system where a server takes two encrypted templates and computes inner product value. In this scenario, the power of computing the inner product value over any two encrypted templates allows the server to retrieve the plain biometric template by computing a few inner products. For example, to retrieve the vector $\mathbf{u} = (u_1, u_2, \dots, u_n)$, the server can get the individual component u_i by computing inner product $\mathbf{u} \cdot \mathbf{e}_i$ where \mathbf{e}_i is the vector with 1 at i -th position and 0 on all other positions. For this reason, we considered a decentralized system where the inner product is computed collectively by several parties, and no single party can compute the inner product independently. In a decentralized system, when some operation involves several parties, everyone has to trust that other parties have done their task correctly and as required. However, in a public network, you can not trust others blindly, as there will always be a malicious party who may try to affect the system’s operations. If a malicious party does not provide correct input to the computation of the inner product between two templates, that might lead to data compromise. In the proposed scheme, we verify computation. Whenever a party computes something and shares it with another party in the network, it has to prove the computation’s correctness. The other party verifies the proof before proceeding further.

We considered a public network of nodes where a user controls each node through biometric authentication. A node becomes active and performs tasks in the network only after the corresponding user passes through biometric authentication. For the authentication part, we considered a random subset of the network. We call it collective authority (CA). It includes

a CA leader and several other nodes. Initially, the CA collectively computes the public key for some homomorphic encryption scheme. The corresponding collective secret key remains distributed among the nodes of the collective authority. This collective public key is made available to all the nodes in the system.

During the registration process, a user encrypts its biometric template using the collective public key and sends it to the network. The collective authority performs the inner product with an existing encrypted biometric template (this is possible because of the homomorphic property of the underlying encryption scheme). Then it decrypts it collectively using all nodes of the collective authority. Finally, it uses the inner product value to check the euclidean distance. If no match is found, then all nodes of the collective authority store the encrypted template. Note that no node can view the template in plaintext form.

For authentication, a user sends an encrypted biometric template along with the node id to the collective authority. Here, we assume a liveness detection check is in place to prevent any replay or spoof attack. The collective authority fetches the corresponding stored biometric template for the particular node id and then computes the inner product with the help of nodes in the collective authority. For this kind of system, the essential requirement is a decentralized encryption scheme where collective authority can perform operations over encrypted values and decrypt them later with the help of all nodes of the collective authority. For this purpose, we proposed a (k, n) decentralized and verifiable BFV scheme for inner product evaluation.

1.1 Our Contribution

- In this paper, we proposed a decentralized version of BFV encryption using the idea of collective authority where the collective public key is computed collectively in such a way that the corresponding collective secret key remains in distributed form.
- We provide verifiable BFV public key generation, which allows anyone to verify that the public key pk is in the form $-(p_1s + e)$ where s, e are sufficiently small, and p_1 is a public value.
- To make the system dynamic, we propose (k, n) decentralized system where any k parties out of n parties can perform decryption. We use the idea of Shamir’s secret-sharing scheme to distribute shares of a collective secret without any trusted setup. We further provide proof for the correctness of the computation of each of the shares.
- Finally, we present verifiable decentralized decryption where each participating node provides proof of correct computation for its partial decryption.

2 Related Work

Homomorphic Encryption (HE) is a cryptographic primitive that allows performing computations on encrypted data. It is a fundamental building block for privacy-preserving applications. The first homomorphic encryption scheme was proposed by Gentry in 2009 [10]. Since then, many homomorphic encryption schemes have been proposed, each with advantages and disadvantages. The most popular homomorphic encryption schemes are based on the learning with errors (LWE) problem [11, 6].

There has been a natural application of homomorphic Encryption to privacy-preserving machine learning. Suppose we can perform computations on encrypted data. In that case, we can train a machine-learning model on encrypted data, which can then be used to predict encrypted

data. A machine learning model that allows performing calculations is a compelling application of homomorphic Encryption. However, training machine learning models is a computationally intensive task that relies on identifying features in the data. Since the encryption scheme does not reveal any feature of the actual data, it is difficult to train a machine-learning model on encrypted data.

Inner product encryption is a cryptographic primitive that allows users to compute the inner product of two vectors without revealing the vectors. The inner product of two vectors is the sum of the products of the corresponding elements of the vectors. The inner product of two vectors is used in many applications, such as secure computation, secure multi-party computation, and secure machine learning, including biometric authentication systems. Function-hiding Encryption is a cryptographic primitive that allows a user to compute a function of a vector without revealing the vector. The function of a vector is the result of applying a function to the elements of the vector, including biometric authentication systems [14].

A unique property of inner product calculation is that it can be extracted from the product of two ciphertexts without revealing the plaintexts. The inner product of two vectors is the sum of the products of the corresponding elements of the vectors after a suitable encoding procedure of the vectors before the encryption [7]. For instance, in a biometric authentication system that uses a neural Network, the matching process can use inner product calculation to compare the encrypted features of the enrolled and the probe without revealing the plain vectors, allowing to perform the authentication process in a privacy-preserving manner.

For instance, in a biometric authentication system that uses a Neural Network, the matching process can use inner product calculation to compare the encrypted features of the enrolled and the probe without revealing the plain vectors, allowing to perform the authentication process in a privacy-preserving manner.

Privacy-preserving cryptographic schemes have been studied for a long time. The first one was proposed by Chaum in 1981 [5]. The first practical scheme was proposed by Goldwasser, Micali, and Rackoff in 1984 [12]. In every case, centralization is a critical aspect of the scheme. The main idea of the scheme is to use a trusted third party (TTP) to perform cryptographic operations. The TTP is a central authority trusted by all the users and responsible for generating the keys, encrypting, and decrypting the data. The TTP is also responsible for performing cryptographic operations. There are several security concerns when using a centralized, trusted third party in privacy-preserving authentication systems [1]. The TTP can be a single point of failure. If the TTP is compromised, all the user's keys are compromised. The TTP can also be a single point of attack and control of the cryptographic operations. The main idea of the decentralized scheme is to use a decentralized network of users to perform cryptographic operations.

However, the server can also be compromised. Using a decentralized homomorphic encryption scheme is a way to protect the server. In a decentralized homomorphic encryption scheme, the server is replaced by a set of nodes. The nodes are connected using a network. The nodes can communicate with each other using the network. The nodes can also communicate with the user. The user can send the biometric feature to its node via a secure channel. The owner's node can then encrypt the biometric feature vector using a homomorphic encryption scheme and send the encrypted biometric feature to the other nodes in the network. The nodes can then decrypt the encrypted biometric feature and compare it with the stored biometric feature. If the two features match, the user is authenticated.

The idea of a decentralized trusted third party could be found in [19], where the authors explore the concept of Collective Authority. In general, the collective authority serves the purpose of a trusted third party that uses a secret sharing protocol and a verification system

to ensure that the collective authority is not compromised. The collective authority can be used to perform cryptographic operations. An excellent example of a general-purpose collective authority that uses the ElGamal encryption scheme is UnLynx [9].

In terms of verifying the operations in a Neural Network, a series of works explores how to implement an efficient Zero-knowledge Proof System that verifies the operations and the output of a Neural Network [16, 8, 4, 15]. However, these works consider a scenario where the Neural Network is private, and the input/output data are public. Some of the work even uses a trusted third party to set up a zero-knowledge common reference string. The trusted third party is a central authority trusted by all the users and is responsible for generating the keys and encrypting and decrypting the data.

3 Cryptographic preliminaries

We begin with a brief introduction to the cryptographic primitives used in this paper. We then describe the primary cryptographic operations used in the protocol and the security properties of the scheme.

3.1 Lattices

In group theory, a lattice in \mathbb{R}^n is an algebraic subgroup of \mathbb{R}^n that spans the vector space \mathbb{R}^n with integer coefficients in its basis.

Formally, let $n \in \mathbb{N}$, $B \in \mathbb{R}^{n \times n}$ a matrix, and $b_i \in \mathbb{R}^n$ the i -th row of B , with $1 \leq i \leq n$. The linear combinations of b_i , defined as

$$\mathcal{L}(B) = \left\{ \sum_{i=1}^n m_i b^i : m_i \in \mathbb{Z}, 1 \leq i \leq n \right\}$$

is a subgroup of \mathbb{R}^n . If the b_i are linearly independent, we say that $\mathcal{L}(B)$ is a *Lattice* in \mathbb{R}^n of dimension n .

3.2 BFV scheme

The BFV scheme is a fully homomorphic encryption scheme that allows the execution of computations on encrypted data. The scheme is based on the learning with errors problem, which is a problem in the computational learning theory consists of finding a function f that maps a set of inputs X to a set of outputs Y such that the error between the output of f and the actual output is small. The BFV scheme is based on the hardness of the learning with errors problem in the context of lattices. The scheme comprises three algorithms: the key generation algorithm, the encryption algorithm, and the decryption algorithm.

A decentralized BFV scheme is a BFV scheme where the public key is distributed among the users. The key is distributed in a way that allows anyone to perform encrypted data computations without a central authority. This paper presents a decentralized BFV scheme where the public key is distributed among n users, and k of them are needed to perform computations on encrypted data.

Here, we describe the BFV scheme and in the following section, we describe the decentralized version based on Collective Authority.

3.2.1 Set-up parameters

The parameters of the BFV scheme are the following:

- m degree parameter
- q prime, defining the ring $R_q = R/qR = \mathbb{Z}_q[x]/f_m(x)$. This ring is the ciphertext space.
- t an integer, with $t \ll q$, defining the ring $R_t = R/tR = \mathbb{Z}_t[x]/f_m(x)$. This ring is the plaintext space.
- σ the parameter for the discrete Gaussian distribution $\chi = D_{\mathbb{Z}^m, \sigma}$

3.2.2 Key generation

1. Sample s as polynomial of degree $(m - 1)$ with coefficients from $\{0, 1\}$.
2. Take a random $p_1 \in R_q$ and the error $e \leftarrow \chi$.

Then the public-key is defined as $\mathbf{pk} = (p_0, p_1)$, where $p_0 = -(p_1 s + e)$, and the secret-key is $\mathbf{sk} = s$.

3.2.3 Encryption

After encoding the plaintext v as an element in R_t and giving the public-key $\mathbf{pk} = (p_0, p_1)$, we sample $u, f, g \leftarrow \chi$ and compute

$$\text{Enc}(v, \mathbf{pk}) = (c_0, c_1) = (p_0 u + g + \Delta v, p_1 u + f)$$

where, $\Delta = \left\lfloor \frac{q}{t} \right\rfloor$.

3.2.4 Decryption

Given the secret-key $\mathbf{sk} = s$ and the ciphertext $\text{Enc}(v, \mathbf{pk}) = (c_0, c_1) = c$, we compute

$$\text{Dec}(c, \mathbf{sk}) = \left\lfloor \frac{t(c_0 + c_1 s)}{q} \right\rfloor \pmod{t} \in R_t$$

Homomorphic Operations:

The BFV scheme allows the execution of homomorphic operations on encrypted data. The homomorphic operations are the following:

$$c = (c_0, c_1), c' = (c'_0, c'_1)$$

$$c + c' = (c_0 + c'_0, c_1 + c'_1)$$

$$c * c' = (c_0 * c'_0, c_0 * c'_1 + c_1 * c'_0, c_1 * c'_1)$$

3.3 Inner product extraction

Let $P = (p_0, \dots, p_{l-1})$ and $Q = (q_0, \dots, q_{l-1})$ bit sequence representations of vectors, with $m \geq l$ over the ring $R = \mathbb{Z}[x]/(x^m + 1)$.

We can define a transformation F onto the ring R such that.

$$f_P = \sum_{i=0}^{l-1} p_i x^i q \quad f_Q = \sum_{j=0}^{l-1} q_j x^{m-j}$$

If we multiply $f_P * f_Q$, then

$$\begin{aligned} f_P * f_Q &= \sum_{i=0}^{l-1} p_i q_i x^m + \dots \\ &= \langle P, Q \rangle x^m + \dots \end{aligned}$$

Thus, if we encrypt f_P and f_Q , thanks to the homomorphic properties of the encryption scheme, we can extract the inner product as a constant term from the encrypted result:

$$\text{Dec}(\text{Enc}(f_P) * \text{Enc}(f_Q)) = \langle P, Q \rangle x^m + \dots$$

3.4 Collective Authority

One of the most critical problems to solve when defining encryption schemes in decentralized environments is handling cryptographic keys. In addition, the calculations are performed and verified by peers through Multi-party Computation. In this sense, we will consider a subgroup of the network, whom we will call Collective Authority, whose objective is to generate the collective keys for homomorphic Encryption and also verify the calculations performed by each node.

The collective authority works as a trusted third party for key generation and verification but is composed of several network nodes. Here, the trust in the collective authority is derived from verifying each communication within the network. During the Setup process, the collective authority is the one who defines the generic parameters for the establishment of the cryptographic protocols. Each node in the collective authority takes these generic parameters and locally generates its public and private keys.

The node keeps its private key secured locally but sends the public key to the collective authority leader. After collecting the public keys from each node, the collective constructs a public key and distributes it back to all nodes. This collective public key is the one used to encrypt the feature vectors. If a malicious user intercepts the public key in a traditional cryptosystem, obtaining the private key is computationally challenging. Since the collective public key is the sum of the individual public keys, it does not reveal any additional information. Therefore, the collective public key generation process is as secure as the underlying public key generation.

3.5 Shamir's Secret Sharing

Shamir's secret sharing scheme is a famous secret sharing scheme for sharing a secret among n parties [18]. The party with secret $s \in \mathbb{Z}_p$ creates shares s_1, s_2, \dots, s_n in such a way that any k shares are enough to retrieve the secret s but $k - 1$ shares do not reveal any information

about the secret. For this, the party randomly picks a $k - 1$ degree polynomial $f(x)$ such that $f(0) = s$ and computes the shares $\{s_i = f(i)\}_{i=1}^n$. Note that the shares are points on the curve defined by $k - 1$ degree polynomial $f(x)$. Since $f(x)$ is of degree $k - 1$, any k points on the curve are sufficient to compute the polynomial $f(x)$. Given any $k - 1$ points on the curve, there are p many possibilities for $k - 1$ degree polynomial which pass through those $k - 1$ points, each of them gives different value for $f(0)$.

Using Lagrange interpolation, we can reconstruct the secret from k shares. Let $I = \{i_1, i_2, \dots, i_k\}$ be set of indices corresponding to the selected k shares. For the index $i_j \in I$, we define Lagrange polynomial as follows:

$$L_{i_j, I} = \prod_{i \in I, i \neq i_j} \frac{x - i}{i_j - i} \text{ mod } p$$

Note that $L_{i_j, I}(i_j) = 1$ and $L_{i_j, I}(i_t) = 0$ for $i_t \neq i_j$. Using shares and corresponding Lagrange polynomial, we construct the polynomial $f(x)$ as follows:

$$f(x) = \sum_{i_j \in I} L_{i_j, I}(x) s_{i_j} \text{ mod } p$$

Since the secret s is a point on the curve at $x = 0$, we have

$$s = \sum_{i_j \in I} L_{i_j, I}(0) s_{i_j} \text{ mod } p.$$

4 Protocols

This section provides concrete constructions of each algorithm in the (k, n) – decentralized and verifiable BFV encryption scheme.

4.1 Setup

The leader of the collective authority generates the setup parameters used for the key generation.

- σ : the standard deviation.
- T : Relinearization parameter
- m : degree of polynomial
- t : size of the plaintext space, R_t
- q : size of the ciphertext space, R_q
- p_1 : public polynomial sampled from R_q

4.2 Collective Key-Generation Process

The collective authority will generate the encryption key, which we call the collective public key. Once the leader publishes the setup parameters, each node in the collective authority runs the key generation algorithm of BFV to generate their key pair.

1. Each node will have a key pair $(\text{sk}_i, \text{pk}_i)$ where,

- Each node sample $\mathbf{sk}_i = s_i \leftarrow \{0, 1\}^m$
 - Sample an error $e_i \leftarrow \chi$
2. Calculate $p_{0i} = -(p_1 s_i + e_i)$
 3. Send public key $\mathbf{pk}_i = (p_{0i}, p_1)$ to the leader.

The leader will take the public keys of all the nodes and add them to compute the collective public key as follows:

$$\mathbf{K} = \left(\sum_i p_{0i}, p_1 \right)$$

The corresponding private key is never constructed and stays distributed among the collective authority members. The leader publishes the collective public key, \mathbf{K} , in the network so that other nodes can use it to encrypt their plaintexts.

4.2.1 Verification of Key Generation:

When a node sends the public key \mathbf{pk}_i to the leader, the leader must be sure whether the public key is correct. If the noise in the public key is too large, then that will result in incorrect message decryption. Therefore, it is necessary to ensure that the corresponding secret key s_i and the noise e_i are selected from the required distributions. For this, we use an LWE-based zero-knowledge proof scheme [3].

Prover: The node picks r_s randomly from $\{0, 1\}^m$ and r_e randomly from χ . It then computes $t = -(p_1 r_s + r_e)$. It then use a collision-resistant and cryptographically secure hash function $H : 0, 1^* \rightarrow \{0, 1, 2, \dots, 2m - 1\}$ and computes $c = H(t || \mathbf{pk}_i)$.

It further computes

$$\mathbf{s}_s = r_s + X^c s_i \quad \mathbf{s}_e = r_e + X^c e_i.$$

It sends $(t, \mathbf{s}_s, \mathbf{s}_e)$ to the leader along with the public key \mathbf{pk}_i .

Verifier: The leader verifies the following equation:

$$X^c \mathbf{pk}_i + t = -(p_1 \mathbf{s}_s + \mathbf{s}_e)$$

It also checks whether $\|\mathbf{s}_s\| \leq 2n$ and $\|\mathbf{s}_e\| \leq 2\sqrt{n}\sigma$ where $\|\cdot\|$ is Euclidean norm.

4.3 Encoding and Encryption

If the plaintext is not an integer, it needs to be encoded in a suitable form to extract the inner product. We assume that the plaintext is already in a quantized form, i.e., all entries are integers only.

Given such quantized plaintext $v = (a_0, a_1, \dots, a_{m-1})$, the encoding produces two polynomials v_a and v_b in the ring $R_t = \mathbb{Z}_t[X]/(x^m + 1)$ with the forms:

$$v_a = a_0 + a_1 x + \dots + a_{m-1} x^{m-1}$$

$$v_b = a_0 x^{m-1} + a_1 x^{m-2} + \dots + a_{m-1}$$

The coefficient of x^{m-1} in $v_a * v_b$ will be the inner product, $\langle \mathbf{v}, \mathbf{v} \rangle$.

The Encryption of the plaintext will be done using the collective public key $\mathbf{K} = (p_0, p_1)$.

1. Encode the plaintext v as v_a and v_b
2. Encrypt v_a and v_b using the collective public key K as follows:

$$c_a = \text{Enc}(v_a, K) \quad c_b = \text{Enc}(v_b, K)$$

After encrypting the plaintext, the node can send it to the network either for registration or authentication.

4.4 Sub-share and Master share generation

Given two encrypted feature vectors, any node can perform a matching process over it. However, we must decrypt the final output to get the matching score. Since the collective secret key is in distributed form, we need to perform decryption in a decentralized manner. This requires participation from each node in the collective authority, and if any one node goes offline for any reason, then the system will not be able to perform decryption. Ideally, the system should work even if a few collective authority nodes are unavailable.

To ensure this, we convert our system into a k -out-of- n system where the collective authority can perform the decryption process as long as at least k out of the total n nodes of the collective authority are available. For this, we use Shamir's secret-sharing method to create shares of the collective secret in a distributed manner.

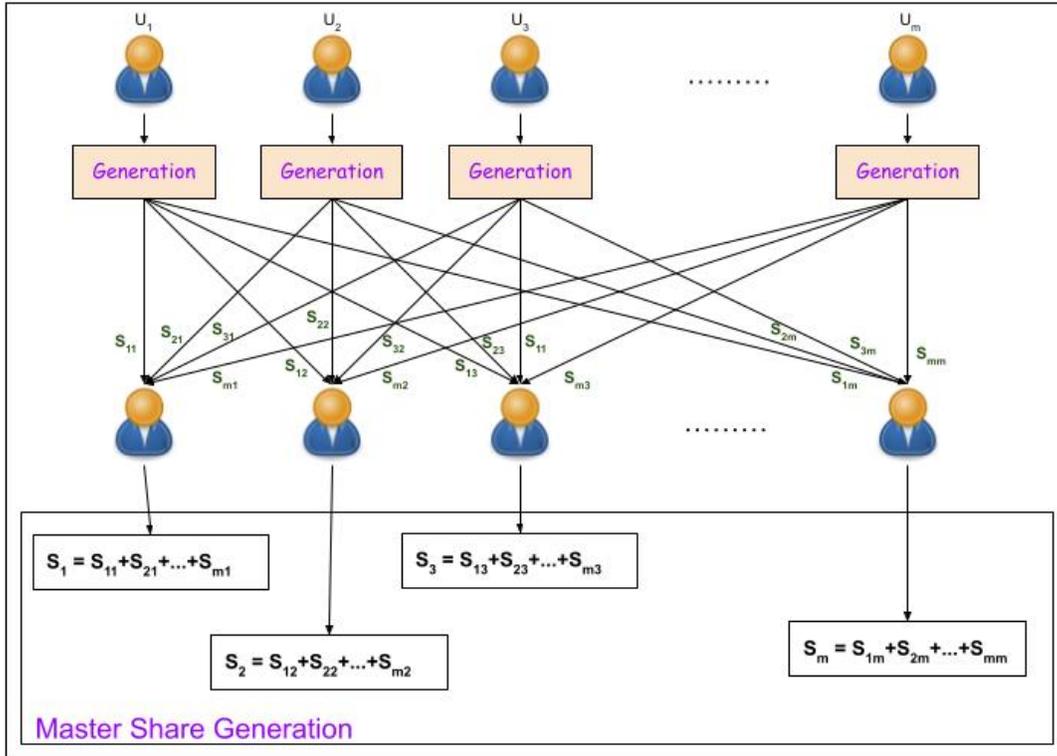


Figure 1: Sub-share and Master share Generation

- **Sub share generation:** Let the collective authority has n nodes, $\{P_1, P_2, \dots, P_n\}$. Each node has their secret key $s_i \in R_q$. The collective secret key is $s = \sum s_i$, which is unknown to anyone in the system.

The node P_i randomly picks $\{g_{ij} \in R_q\}_{j=1}^{k-1}$ and set $g_i(y) = \sum_{j=0}^{k-1} g_{ij}y^j$ where $g_{i0} = s_i$.

In other words, the node P_i randomly picks a $k - 1$ degree polynomial whose coefficients are in R_q , and the constant term is s_i . This polynomial will generate sub-shares for other participant nodes. Let

$$g(y) = \sum_{i=1}^n g_i(y).$$

For each participant node P_j , the node P_i computes $s_{ij} = g_i(j)$. The node P_i publishes $\{s_{ij}\}_{i \neq j, 1 \leq j \leq m}$

- **Master share generation:** Using s_{ij} , each node P_i computes their own master share as follows:

$$S_i = g_i(i) + \sum_{j=1, j \neq i}^n s_{ji} = g_i(i) + \sum_{j=1, j \neq i}^n g_j(i) = \sum_{j=1}^n g_j(i) = g(i)$$

At the end, each participant has master secret $S_i = g(i)$ and $g(y)$ is a $k - 1$ degree polynomial over R_q with $g(0) = \sum_{i=1}^n g_i(0) = \sum_{i=1}^n s_i = s$ (collective secret).

This setting becomes similar to Shamir's secret-sharing scheme but over polynomials without a trusted third party.

4.4.1 Verification of sub-shares

When the node **a** sends the sub-share s_{ab} to the node **b**, it is necessary for the node **b** verify that s_{ab} is computed correctly.

The node **a** has secret key $\text{sk}_a = s_a$, public key $\text{pk}_a = -(p_1 s_a + e_a)$ and secret polynomial $g_a(y)$ with $g_a(0) = s_a$.

It then computes public polynomial $h_a = (h_{a0}, h_{a1}, \dots, h_{a(k-1)})$ where

$$h_{a0} = \text{pk}_a \quad h_{ai} = -(p_1 g_{ai} + e'_{ai})$$

with e_i is noise from the distribution χ .

The node computes the sub-share s_{ab} for the node **b**. Let $r_a = (r_{a0}, r_{a1}, \dots, r_{a(k-1)})$ where r_{ai} is randomly selected from the distribution χ . It further computes

- $y_{ab} = \sum_{j=0}^{k-1} e'_{aj} b^j$
- $t_{ab} = \sum_{j=0}^{k-1} r_{aj} b^j$
- $c_{ab} = H(t_{ab} || y_{ab} || s_{ab})$

The node **a** publishes c_{ab} and waits for c_{ib} from other nodes. Once it receives all $\{c_{ib}\}_{i \neq a, b}$, it further computes

- $c_b = H(c_{1b} || \dots || c_{nb})$
- For $0 \leq i \leq k - 1$, $u_{ai} = r_{ai} + X^{c_b} e'_{ai}$

let $u'_{ab} = (u_{a0}, u_{a1}, \dots, u_{a(k-1)})$ It sends $\{s_{ab}, h_a, t_{ab}, u'_{ab}\}$ to the node **b**.

The node **b** first computes $y'_{ab} = h_a(b) + p_1 s_{ab}$ and $c_b = H(c_{1b} || \dots || c_{nb})$ where $c_{ib} = H(t_{ab} || y'_{ab} || s_{ab})$. It then checks

$$X^{c_b} y'_{ab} + t_{ab} = \sum_{j=0}^{k-1} u_{aj} b^j$$

and $\|u_{ai}\| \leq 2\sqrt{n}\sigma$.

4.5 Decentralized Decryption using Master shares

The normal decryption of a ciphertext $c = (c_0, c_1)$ is as follows:

$$\text{msg} = \left\lfloor \frac{t(c_0 + c_1 * s)}{q} \right\rfloor \pmod{t}.$$

Using Lagrange interpolation method, we have

$$s = \sum_{j=0}^{k-1} \left(g(x_j) \prod_{i=0, i \neq j}^{k-1} \frac{x_i}{x_i - x_j} \right)$$

where $\{(x_j, g(x_j))\}_{j=0}^{k-1}$ are k many distinct points on the polynomial $g(y)$. Each node P_i has a point $(i, S_i = g(i))$. Using any k of these points we can compute s .

Also, we have

$$c_1 * s = c_1 * \sum_{j=0}^{k-1} \left(g(x_j) \prod_{i=0, i \neq j}^{k-1} \frac{x_i}{x_i - x_j} \right) = \sum_{j=0}^{k-1} \left(c_1 * g(x_j) \prod_{i=0, i \neq j}^{k-1} \frac{x_i}{x_i - x_j} \right).$$

At the start of the decryption process, the leader of the collective authority will first find online nodes and pick any k of them. Assume that $\{P_1, P_2, \dots, P_k\}$ are the selected nodes. The leader publishes this list and asks each to participate in decryption.

Then each of them will compute

$$d_j = c_1 * S_j \prod_{i=1, i \neq j}^k \frac{i}{i - j} + p_1 \zeta_{1j} + \zeta_{2j} \quad \text{and} \quad z_j = p_1 \zeta_{1j} + \zeta_{3j}$$

where ζ_{1j}, ζ_{2j} , and ζ_{3j} are noise polynomial from distribution χ . The node shares (d_j, z_j) with the leader.

Note that

$$\sum_{j=1}^k (d_j - z_j) = \sum_{j=1}^k c_1 * S_j \prod_{i=1, i \neq j}^k \frac{i}{i - j} + \zeta_{2j} - \zeta_{3j} = \sum_{j=1}^k c_1 * S_j \prod_{i=1, i \neq j}^k \frac{i}{i - j} + \zeta'_j$$

where $\zeta'_j = \zeta_{2j} - \zeta_{3j}$.

The leader then computes

$$\text{msg} = \left\lfloor \frac{t \left(c_0 + \sum_{j=1}^k (d_j - z_j) \right)}{q} \right\rfloor \pmod{t} = \left\lfloor \frac{t (c_0 + c_1 * s + \zeta)}{q} \right\rfloor \pmod{t}$$

where $\zeta = \sum_{j=0}^k \zeta'_j$ and each coefficients of ζ is less than $\frac{q}{2t}$.

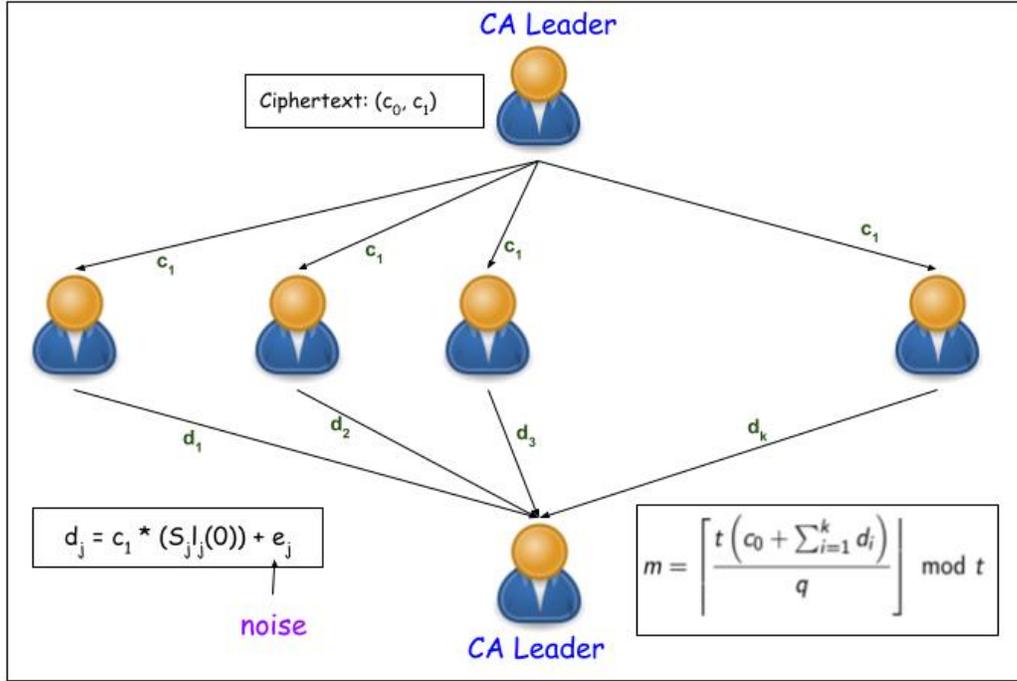


Figure 2: Decentralized Decryption

4.5.1 Verification of Decentralized Decryption

In decentralized decryption, a participating node a performs partial decryption using its master share and outputs d_a . If the node a is malicious and does not compute d_a correctly, it will lead to incorrect final decryption. By checking d_a , it is impossible to say whether the node a has computed it correctly. To ensure the correctness of d_a , we use a verification system based on a zero-knowledge proof for LWE secrets.

Assume that the node a is participating in the decentralized decryption of the ciphertext $c = (c_0, c_1)$. The node a will compute (d_a, z_a) as described in the section 4.5. It further picks $r_{01}, r_{02}, r_{11}, r_{12}$ randomly from the distribution χ . It then computes

- $t_0 = p_1^2 r_{01} + p_1 r_{02}$
- $t_1 = p_1 r_{11} + r_{12}$
- $v_{01} = r_{01} + X^{c_a} \zeta_{1a}$
- $v_{02} = r_{02} + X^{c_a} \zeta_{2a}$

- $v_{11} = r_{11} + X^{c'_a} \zeta_{1a}$
- $v_{12} = r_{12} + X^{c'_a} \zeta_{3a}$

where c_a is the same value that was computed by the node a during the verification of the subshares s_{ia} and $c'_a = H(t_0 || t_1 || c_a || d_a || z_a)$.

It then sends $(t_0, t_1, v_{01}, v_{02}, v_{11}, v_{12})$ along with (d_a, z_a) to the CA leader.

The leader computes

$$y_a = p_1 d_a + \pi_a c_1 h(a)$$

where $\pi_a = \prod_{i=1, i \neq a}^k \frac{i}{i-a}$ and $h = \sum_{i=1}^n h_i$.

It then uses $\{t_{ia}\}_{i=1}^n$ and $\{u'_{ia}\}_{i=1}^n$ from the sub share verification and checks the correctness of the following equalities:

$$\begin{aligned} X^{c_a} y_a + \pi_a c_1 \sum_{i=1}^n t_{ia} + t_0 &\stackrel{?}{=} \pi_a c_1 \sum_{i=1}^n \langle u'_{ia}, \vec{a} \rangle + p_1^2 v_{01} + p_1 v_{02} \\ X^{c'_a} + t_1 &\stackrel{?}{=} p_1 v_{11} + v_{12} \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ represents inner product function and $\vec{a} = (1, a, a^2, \dots, a^{k-1})$.

The leader also checks whether $\|v_{01}\|, \|v_{02}\|, \|v_{11}\|, \|v_{12}\| \leq 2\sqrt{m}\sigma$. If all verification steps are valid, then the leader proceeds to the final decryption step described in the section 4.5, else it aborts.

5 Security Analysis

We analyze the security of each protocol by going through each step of our decentralized protocols presented in section 4. We separately analyze the security of the corresponding verification protocols.

Collective Key-Generation:In this protocol, each node in the system generates a BFV key pair (sk_i, pk_i) and keeps the secret key sk_i hidden. Only public key pk_i is shared with the CA leader, who aggregates all individual public keys to compute the collective public key K . The corresponding collective secret key, $\sum_i sk_i$ remains unknown as long as at least one (sk_i, pk_i) is safe.

Encoding and Encryption:A vector is encoded as a polynomial and then encrypted using BFV encryption. As long as the parameters for BFV are selected for 128 bit security as recommended in [2], the encoding and Encryption of the vector remain secure.

Sub-share Generation:In this protocol, each node in the collective authority uses Shamir's secret sharing scheme to create secret shares (called sub-share) for the individual secret key sk_i . The node i sends the sub share s_{ij} to the node j via a private channel to keep it confidential.

For k out of n secret sharing scheme, the secret is hidden as long as the attacker cannot get at least k of the corresponding sub-shares. Overall, the security of the sub-share generation is equivalent to that of Shamir's secret-sharing scheme.

Decentralized Decryption:In this protocol, a selected node performs partial decryption of the ciphertext (c_0, c_1) using its master share. The noise noise_j in the partial decryption d_j hides information about the master share.

Assume that the leader has multiple partial decryptions of different ciphertexts from the same node j . That means the leader has list of partial decryptions, $\{S_j * x_i + \text{noise}_i\}$, and tries to learn about the master share S_j . However, this is equivalent to the learning with errors (LWE) problem, which is assumed to be a hard problem for selected parameters.

The CA leader gets d_j from each participating node and performs the final step to retrieve the message. However, the leader cannot learn anything about the corresponding master share from the partial decryption d_j .

5.1 Security Analysis of Verification

Now, we analyze the security of each of the verification protocols. In all the verification protocols, we use the idea from Zero-Knowledge proof for LWE-secrets s, e such that $y = as + e$ for some public value a [3]. The relation is as follows:

$$\mathcal{R} = \{((a, y), (s, e)) : y = as + e \wedge \|s\|, \|e\| \leq \mathcal{O}(\sqrt{n}\alpha)\}$$

Given (a, y) , the protocol provides zero-knowledge proof that the prover knows short elements s, e such that $y = as + e$, that is, y is a linear combination of short elements for some publicly known linear coefficients.

We consider a more generalized version of the ZKP for LWE secrets in verifying sub-share generation and decentralized decryption. The generalized relation is as follows:

$$\mathcal{R} = \left\{ ((a_1, a_2, \dots, a_k, y), (e_1, e_2, \dots, e_k)) : y = \sum_{i=1}^k a_i e_i \wedge \|e_1\|, \dots, \|e_k\| \leq \mathcal{O}(\sqrt{m}\alpha) \right\}$$

- **Verification of Key Generation:** Here, we used ZKPoK of LWE secrets to prove that the public key pk_i generated by the node i is in correct form with the corresponding secret and the noise having a small norm. The corresponding relation is as follows:

$$\mathcal{R} = \{((-p_1, -1, \text{pk}_i), (s_i, e_i)) : \text{pk}_i = (-p_1) * s_i + (-1) * e_i \wedge \|e_1\|, \dots, \|e_k\| \leq \mathcal{O}(\sqrt{m}\sigma)\}$$

where σ is the parameter of the noise distribution χ .

- **Verification of sub-shares:** Here, we used a generalized version of ZKPoK of LWE secrets to prove that the sub share $s_{ab} = g_a(b)$ indeed is generated for the secret s_a and $\{s_{ab}\}_{b=1}^n$ are all generated using the same polynomial g_a .

Since the polynomial g_a must be kept secret, the node a computes $h_a = -(p_1 * g_a + e'_a)$ and makes h_a public. Finding g_a from h_a is as hard as solving the LWE problem. We use this h_a to verify the claim about the sub share s_{ab} . Note that

$$h_a(b) + p_1 s_{ab} = -(p_1 g_a(b) + e'_a(b)) + p_1 g_a(b) = e'_a(b)$$

where $e'_a(b) = \sum_{j=0}^{k-1} e'_{aj} b^{j-1}$ and e'_{aj} are selected from the noise distribution χ . Therefore, we prove that $y = h_a(b) + p_1 s_{ab}$ is indeed a linear combination of small elements. The corresponding relation is as follows:

$$\mathcal{R} = \left\{ ((1, b, \dots, b^{k-1}, y), (e'_{a0}, e'_{a1}, \dots, e'_{a(k-1)})) : \right. \\ \left. y = \sum_{i=0}^{k-1} e'_{ai} b^i \wedge \|e'_{a0}\|, \dots, \|e'_{a(k-1)}\| \leq \mathcal{O}(\sqrt{m}\sigma) \right\}$$

- **Verification of Decentralized Decryption:** The participating node performs partial decryption and computes (d_a, z_a) . The verification process ensures that the d_a is computed correctly using a valid master share. We use ZKPoK twice to prove that $y_a = p_1 d_a + \pi_a c_1 h(a)$ and z_a are both linear combinations of small elements.

Note that the master share for the node a is $g(a) = \sum_{i=1}^n g_i(a)$. Using the public polynomials $\{h_i = -(p_1 g_i + e'_i)\}_{i=1}^n$, one can compute

$$h(a) = \sum_{i=1}^n h_i(a) = \sum_{i=1}^n -(p_1 g_i(a) + e'_i(a)) = - \left(p_1 g(a) + \sum_{i=1}^n e'_i(a) \right).$$

We use this h_a to remove the term involving master share in d_a and then use ZKPoK of LWE secrets for the remaining part.

$$y_a = p_1 d_a + \pi_a c_1 h(a) = p_1^2 \zeta_{1a} + p_1 \zeta_{2a} - \pi_a c_1 \sum_{i=1}^m e'_i(a)$$

Note that y_a is a linear combination of small elements; therefore, we can verify it using ZKPoK for LWE-secrets. Similarly, we verify that z_a is a linear combination of small elements.

Now, assume that $d'_a = \pi_a c_1 S'_a + p_1 \zeta_{1a} + \zeta_{2a}$ where $S'_a \neq S_a$. That means an incorrect master share is used for the computation of d'_a . In this case, $y'_a = y_a + \pi_a c_1 (S'_a - S_a)$. Then d'_a passes through the verification if and only if an attacker can write $\pi_a c_1 (S'_a - S_a)$ as a linear combination of short elements in the ring \mathcal{R}_q . This is equivalent to a short vector solution problem in the ring \mathcal{R}_q .

6 Implementation

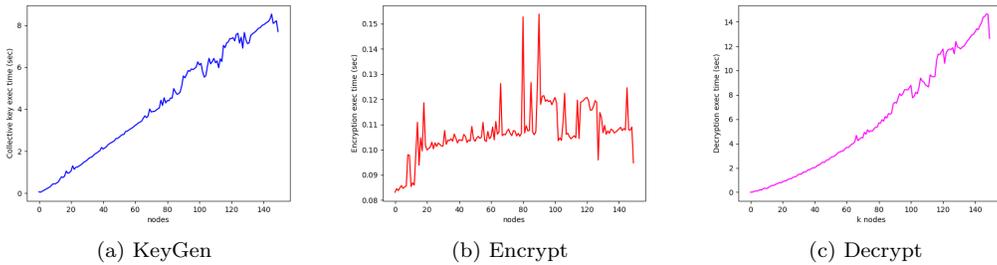


Figure 3: Execution time for the different operations in the protocol

6.1 Parameter Generation

The selection of the parameter for classic (no decentralized) homomorphic encryption was standardized in the document [2]. Most researchers agree that although it is essential to consider the Standard suggestions, the process for selecting them is mostly heuristic. In our case, we have several constraints that need to be satisfied. The basic parameter is the number m , that is a j -th power of 2, with $j = 10, 11, 12, \dots$

The following constraint is the security level l , the number of bits of the ciphertext. When generating the parameters for the encryption scheme, we need to consider the number of nodes in the Collective Authority (CA) and the value σ , which is the parameter for the discrete Gaussian distribution. It is important to note that the number of nodes in the CA differs from the number in the network. The number of nodes in the CA is the number of nodes that can participate in the two-round decryption, inner-product calculation, and the matching protocol.

An open question is whether the number of nodes in the CA impacts the encryption scheme's security level. In summary, the parameters for the encryption scheme are two prime numbers, q , and t . The number q is the size of the encrypted space, and t is for the plaintext space. The constraints for q and t are as follows:

1. q and t are prime numbers
2. $q | (2 * m - 1)$
3. $(q - 1) | t$
4. $3 * m * n * \sigma < \frac{q}{2 * t}$

where n is the number of nodes in the Collective Authority, and σ is the parameter needed for the random distributions and usually is set to $\sigma = 3.2$

The pseudocode for the parameter generation is as follows:

Algorithm 1 Parameter Generation

```

1: procedure GENERATEPARAMETERS
2:    $m \leftarrow$  base param
3:    $l \leftarrow$  security level
4:    $n \leftarrow$  number of nodes in CA
5:    $\sigma \leftarrow$  parameter for random distributions
6: top:
7:   if  $l = 128$  and  $m = 2^{10}$  then
8:     size of encrypted space  $q$  is 29 bits.
9:   else
10:    if  $l = 128$  and  $m = 2^{11}$  then
11:      size of encrypted space  $q$  is 56 bits.
12: loop:
13:   GeneratePrimes  $q$  and  $t$ 
14:   while  $(q | (2 * m - 1) \wedge (q - 1) | t \wedge 3 * m * m * \sigma < \frac{q}{2 * t})$  do
15:     GeneratePrimes  $q$  and  $t$ 

```

6.2 Complexity analysis

Let n be the number of nodes in the network and k be the number of nodes in the CA. Let m be the degree of the polynomials. We now present the complexity of the different operations in the protocol.

Key Generation: In this case, we have two operations, the generation of each node’s private key and the generation of the collective public key. The complexity of each node’s public key generation is determined by a multiplication of a polynomial of degree m by a constant number and an addition of two polynomials of degree m . the complexity in this case is $O(m)$. The complexity of the collective public key generation is determined by adding k polynomials of degree m . The complexity, in this case, is $O(k)$. The total complexity of the key generation is $O(m + k)$.

Encryption: The complexity of the Encryption is determined by the product of two polynomials of degree m . The complexity will be $O(m^2)$. If the polynomial multiplication is done using the FFT algorithm, the complexity will be $O(m \log m)$. Then there is an addition of two polynomials of degree m . Thus, the total complexity of the Encryption is $O(m \log m)$. Each node in the network performs Encryption, so it does not depend on the number of nodes in the CA.

Two-round decryption: The two-round decryption is done in two phases. The first phase is the calculation of the master shares. The complexity of the master shares calculation is determined by sub-share generation for each node. The sub-share is the sum of $k - 1$ products of polynomials of degree m . This gives us a complexity of $O(km \log m)$. Then the master share is the sum of n sub shares from the rest of the nodes and its sub share. This gives us a complexity of $O(nm \log m)$.

Then we use the Lagrange interpolation to calculate the secret using at least k master shares. The complexity of the Lagrange interpolation is $O(km^2)$. Then we multiply the secret by one of the components of the ciphertext. The complexity of the multiplication $O(m \log m)$. Then each node in the network calculates the partial decryption using its master share and the previous multiplication. Taking into account the Lagrange interpolation, the multiplication of the Lagrange interpolation by the first polynomial of degree m , and the multiplication of the secret by the second polynomial of degree m , the total complexity of the two-round decryption is $O(km^2 \log m + nm \log m)$.

The next table summarizes the complexity of the different operations in the protocol:

Operation	Complexity
Key Generation	$O(m + k)$
Encryption	$O(m \log m)$
Two-round decryption	$O(km^2 \log m + nm \log m)$

Table 1: complexity of the different operations in the protocol

6.3 Execution time

Now we will present the execution time for the different operations in the protocol. Figure 3 shows the execution time for the different operations in the protocol.

We tested the protocol with 150 nodes in the network and a variation of two third of the nodes in the CA. The execution time for key generation is presented in figure 3a. As expected, the execution time increases linearly as we increase the node. This become constant time when we consider each node generating their public keys simultaneously.

The execution time for the Encryption is presented in figure 3b. The encryption time is more or less constant as increasing the number of nodes does not affect the encryption key and the encryption algorithm. The time cost for two-round decryption is presented in 3c. The decryption time increases as we increase the number of nodes due to the decentralized nature of the decryption. We considered two-thirds of the total nodes as the threshold number. As the number of nodes increases, the threshold number also increases, which results in a linear increase in the decryption time.

7 Conclusion

In this paper, we presented a new collective public key encryption scheme. The scheme uses a CA to generate the collective public key. The CA is a group of nodes in the network that generates the collective public key and distributes it to the rest of the nodes in the network. The nodes in the network use the collective public key to encrypt their data. The CA uses a two-round decryption protocol to decrypt the data. Using this CA, we developed a new decentralized version of BFV that uses a (k,n) secret sharing scheme to create shares of the collective secret in a distributed manner. Also, we studied this new scheme's security properties and presented a complexity analysis and execution time for the different operations in the protocol. This work can be applied to biometric authentication in a decentralized network.

References

- [1] Aysajan Abidin and Aikaterini Mitrokotsa. Security aspects of privacy-preserving biometric authentication based on ideal lattices and ring-lwe. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 60–65, 2014.
- [2] Martin Albrecht, Melissa Chase, Hao Chen, and et al. Homomorphic encryption standard. <http://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf>, 2018.
- [3] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In *ASIACRYPT*, 2014.
- [4] Sean Bove, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zeze: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964, 2020.
- [5] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, feb 1981.
- [6] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. <https://eprint.iacr.org/2012/144>.
- [7] Shisen Fang, Shaojun Yang, and Yuexin Zhang. Inner product encryption from ring learning with errors. *Cybersecurity*, 3, 12 2020.
- [8] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. Zen: Efficient zero-knowledge proofs for neural networks. *IACR Cryptol. ePrint Arch.*, 2021:87, 2021.

- [9] David Froelicher, Patricia Egger, João Sá Sousa, Jean Louis Raisaro, Zhicong Huang, Christian Mouchet, Bryan Ford, and Jean-Pierre Hubaux. Unlynx: A decentralized system for privacy-conscious data sharing. *Proceedings on Privacy Enhancing Technologies*, 2017:232 – 250, 2017.
- [10] Craig Gentry. Fully homomorphic encryption using ideal lattices. STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.
- [11] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [12] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Symposium on the Theory of Computing*, 1985.
- [13] Anil K. Jain and Karthik Nandkumar. Biometric authentication: System security and user privacy. *Computer*, 45 (11):87 – 92, 11 2012.
- [14] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. Cryptology ePrint Archive, Paper 2016/440, 2016. <https://eprint.iacr.org/2016/440>.
- [15] Seunghwan Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vcnn: Verifiable convolutional neural network. *IACR Cryptol. ePrint Arch.*, 2020:584, 2020.
- [16] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkenn: Zero knowledge proofs for convolutional neural network predictions and accuracy. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.
- [17] Dhananjay Nigam, Shilp Patel, P. M. Durai Raj Vincent, Kathivaran Srinivasan, and Sinouvassane ArunMozhi. Biometric authentication for intelligent and privacy-preserving healthcare systems. *Journal of Healthcare Engineering*, 2022:1 – 15, 03 2022.
- [18] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [19] Ewa Syta, Iulia Tamas, Dylan Visher, David Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities ”honest or bust” with decentralized witness cosigning. *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545, 2016.
- [20] Kai Zhou and Jian Ren. Passbio: Privacy-preserving user-centric biometric authentication. *IEEE Transactions on Information Forensics and Security*, 13(12):3050–3063, 2018.