Game On: A Performance Comparison of Interpolation Techniques Applied into Shamir's Secret Sharing

Anastassis Voudouris[1], Aristomenis Tressos[1], Apostolis Zarras[2] and Christos Xenakis[1]

[1]*University of Piraeus, Greece*
[2]*Foundation for Research and Technology – Hellas, Greece*
Email: anastassis.voudouris@ssl-unipi.gr

Public-key encryption is typically managed through a Public Key Infrastructure (PKI). However, it relies on a central control point, the Certification Authority (CA), which acts as a single point of failure. Recent technological advancements have led to the need for decentralized cryptographic protocols. This paper presents a comprehensive study on enhancing public-key encryption via threshold cryptography and Multiparty Computation (MPC) to ensure robust security in decentralized systems. The focus lies in exploring various polynomial interpolation techniques within Shamir's Secret Sharing Scheme (SSS), particularly addressing the efficiency and practicality of Newton interpolation, Fast Fourier Transformation (FFT), and advanced versions of Lagrange's method. Utilizing SageMath for a dedicated testing environment, the research investigates the swiftest interpolation methods for secret recovery, introducing new shares into the system, and evaluating the impact of optimizations on performance. The findings highlight FFT as the most effective interpolation method in speed and efficiency, albeit with limitations on the number of shares that can be processed. This paper critically evaluates these interpolation techniques against practical constraints and aims to answer pivotal research questions regarding the optimal approach for large-scale scenarios, challenging existing notions on the efficiency of Newton's method and providing experimental evidence to support the superiority of FFT in specific contexts.

Keywords: Security, Cryptography, Interpolation, Shamir Secret Sharing

1. INTRODUCTION

Public-key encryption in cryptographic schemes is commonly managed through a Public Key Infrastructure (PKI). PKI ensures secure communication across insecure public networks by employing digital signatures to authenticate entities. The PKI relies on the security of a central control point, named the Certification Authority (CA), and is considered trusted by all; however, it also acts as a single point of failure. If this point gets compromised, the entire network's security will fall. Recent technological advancements showed that the research community needs to strengthen current cryptographic protocols in a decentralized manner [1]. New ideas are driven by a well-studied area of cryptography called *threshold cryptography* [2]. A (n, k) threshold cryptography scheme allows n entities to share the ability to perform a cryptographic operation so that any k entities suffice to perform this operation jointly. In contrast, it is infeasible for at most $k-1$ entities to do so, even by collusion. A cryptographic key, which will be used to encrypt the message transmitted from one person to another, is the fundamental tool for both symmetric and asymmetric cryptography and must be protected, from the generation part to the secret sharing among the group's entities.

Secret sharing and threshold cryptography are both branches of the same tree, called, in general terms, Multiparty Computation (MPC). One security application of MPC is to protect cryptographic keys from theft and misuse by never having them exposed at any single point at any single time and by enforcing usage policies at multiple entities. By embracing MPC's ability to protect cryptographic keys in software, an alternative to legacy hardware solutions, such as

Trusted Platform Module's (TPM) root of trust, is provided, mitigating many operational challenges in today's hybrid and multi-cloud environments.

Shamir's Secret Sharing Scheme (SSS) is the main sub-module of threshold cryptographic protocols [3]. Assuming that trust is established, a dealer owning a secret desires to distribute this secret among a group of n shareholders. A shared secret is generated by the trusted dealer, who distributes a share (of a cryptographic key, for example) to each member of the group, such that any subset of size greater than a threshold (t) can reveal or use the shared secret, while smaller subsets do not have any knowledge about it. The core subroutine of SSS is Lagrange's interpolation, through which the shared secret is eventually reconstructed.

It is noteworthy that numerous optimizations for Lagrange's interpolation have been documented in the literature, aimed at accelerating the computation of Lagrange polynomials and obtaining the constant term of the interpolating polynomial, which serves as the secret in the standard SSS scheme [4]. Additionally, there are various other methods available for interpolation. Here are a few notable cases: Multi-secret schemes involve using the secret s as the highest degree coefficient, thereby increasing the efficiency of another interpolation method called Newton's interpolation [5]. The adoption of Newton's interpolation has also been proposed in a recent academic publication where the authors explore the application of this approach in the context of Multi-Factor authentication during the introduction of a new node in a communication protocol [6]. Furthermore, Tomescu et al. [7] aggregate a Boneh-Lynn-Shacham (BLS) signature [8] 3000 times faster in 46 seconds compared to 1.5 days if done naively. This improvement is achieved by utilizing a Fast Lagrange interpolation algorithm, which has a time complexity of $\Theta(t\ log^2\ t)$, where in n total users, the $t > n/2$ must be honest, and incorporates a Fast Fourier interpolation method. This approach outperforms the naive Lagrange algorithm, which has a time complexity of $\Theta(t^2)$. This centers on exploring alternative interpolation techniques instead of the traditional Lagrange method, focusing on enhancing performance.

In response to concerns regarding interpolation methods within the context of SSS, this article presents a comprehensive exploration of various polynomial interpolation techniques. The methodologies scrutinized include Newton interpolation, the Fast Fourier Transformation (FFT), and two refinements of Lagrange's method. The research utilizes a dedicated testing environment, employing SageMath [9], a freely available open-source mathematics software system licensed under the GPL, and utilizing Python-based language. The testing environment accommodates fields with diverse user counts and specific mathematical properties. Additionally, the authors delve into the implications of introducing new shares into the system, akin to a simple communication protocol. Experimental findings demonstrate that, in terms of performance, the FFT emerges as the most efficient interpolation method in both scenarios. However, it is noteworthy that limitations exist regarding the number of shares that can be effectively utilized. Overall, this article poses three main research questions: (*i*) What is the swiftest method and most suitable in large-scale scenarios to interpolate the polynomial in the context of SSS to recover the secret from the constant term? (*ii*) What is the quickest approach to interpolate the polynomial for recovering the secret when adding additional members/shares? (*iii*) Do optimizations (e.g., precomputing values) affect the overall performance of interpolation methods? These research questions form the basis of this work.

In summary, this article makes the following main contributions:

- A comprehensive examination of different interpolation techniques and their implementation in the SSS scheme.
- A specific use case addressing the addition of new shares and the subsequent reconstruction of the secret, challenging the assertion by Bezzateev et al. [6] regarding Newton's method as the swiftest.
- Experimental evidence demonstrating the superior efficiency of the FFT compared to other methods.

The remainder of this paper is organized as follows. In Section 2, we elaborate on several fundamental concepts in cryptography, encompassing finite fields and roots of unity, culminating in an exhaustive exploration of SSS. Moving forward, Section 3 conducts an in-depth analysis of various interpolation methods, including Lagrange's method, its optimizations, Newton's method, and a specialized case involving FFT. Section 4 encapsulates the culmination of all the experiments. In Section 5, a concise discussion addresses the original research questions. Additionally, Section 6 includes a number of research papers that employ various interpolation methods within the context of SSS, following the notable contributions of Shamir and Blakley. Finally, Section 7 wraps up the paper by outlining avenues for future research and posing pertinent questions.

2. BACKGROUND

Before delving into the concepts of secret sharing and polynomial interpolation, it is imperative to establish a foundational understanding. To enhance the clarity of subsequent sections in our work, we aim to lucidly present essential mathematical concepts from various fields, articulating definitions and theorems without proof. Readers are guided to the relevant references for comprehensive insight when further elucidation is required.

2.1. Mathematical Foundations

As a prerequisite for the fundamentals of cryptographic theory, a grasp of finite fields and groups is indispensable. The ensuing paragraphs furnish a concise introduction to vital mathematical concepts, laying the groundwork for a comprehensive understanding of our implementations and approaches to polynomial interpolation.

Finite Fields. The set of real numbers, $\mathbb{R}$, has infinite order, i.e., contains an infinite number of elements, whereas a finite field, or Galois Field (GF), has a finite order. $GF(p)$ (or $\mathbb{F}_p$), where p is prime, is a finite field with finite order p and is mostly used in cryptographic applications due to its excellent properties. Note that the order of a finite field is always a prime p or a prime power, p^n [10]. Finite fields have two operations (i.e., addition and multiplication), which adopt them from the corresponding groups. A fundamental intuition of groups, especially the cyclic ones, is the generator of a group. The order of an element $x \in \mathbb{F}_p$, denoted as $|x|$, is the number of unique elements that x can generate raised to the elements of the group $1, 2, 3, ..., p-1$. The elements that generate all $p-1$ elements of the group are called *generators* or *primitive roots*, and the groups generated by a generator are called *cyclic groups*. The order of generators is always equal to $p-1$. If an element $g \in \mathbb{F}_p$ is a generator, then $g^k \neq 1 \; \forall \; k \neq p-1$.

2.2. Roots of Unity

The following points are crucial for grasping the motivation behind the FFT and the Discrete Fourier Transformation (DFT), which we will demonstrate to be the *fastest* method in terms of time for computing the secret of SSS, under specific conditions.

A complex root of unity is a convoluted number ω such that $\omega^n = 1$. There are exactly n complex $n-th$ roots of unity, $e^{\frac{i2k\pi}{n}} \cdot k = 0.1. \cdots n-1$. Following, according to the cancellation lemma [11] we take that for any integers $n \geq 0, k \geq 0$, and $d \geq 0, \omega_{dn}^{dk} = \omega_n^k$, and that for any even integer $n \geq 0$, $\omega_n^{\frac{n}{2}} = \omega_2 = -1$. Lastly, according to the halving lemma [11], if $n \geq 0$ is even, the squares of the n complex n^{th} roots of unity are the $n/2$ complex $n/2 - th$ roots of unity. The halving lemma is crucial to the divide-and-conquer technique for converting between coefficient and point-value representations of polynomials, which we will analyze in the polynomial interpolation section, since it ensures that the recursive subproblems are half as large as the original problem. However, considering that we are working on a finite field and not on complex numbers, the definition of the roots of unity has a small twist. Let p be a prime and $\mathbb{F}_p$ be the finite field of p elements. Let ω be an element of $\mathbb{F}_p$. We say that ω is an nth root of unity if $\omega^n \equiv 1 \pmod{p}$ but not $\omega^m = 1$, for all $0 < m < n$. Readers are referred to Stewart [12] for in-depth subject analysis. Lastly, a criterion to determine a root of unity is that the finite field $\mathbb{F}_p$ has a primitive n^{th} root of unity if and only if n divides the order of $\mathbb{F}_p$, which is $p-1$. Primitive n^{th} roots of unity can be a convenient way to specify elements of a group that have order n [13].

2.3. Secret Sharing

Secret sharing (SS) is now one of the fundamental elements of cryptographic protocols. Consider a group $S = s_1, s_2, \ldots, s_n$ of parties called *shareholders* or *nodes* that are mutually distrusting of one another and a centralized adversary with the ability to control up to t parties, often referred to as a dealer or data owner. Usually, the designated dealer $D \in P$ may share a secret s by providing each s_i a share σ_i of the secret s via a secret sharing scheme.

On a high level, a secret sharing scheme is secure if any unauthorized subset of participants learns nothing about the message, while any authorized subset of participants can fully reconstruct the secret, also known as *accessibility*. SSS is perfectly secure, i.e., secure in an information-theoretic sense, meaning that unauthorized coalitions cannot deduce any information about the secret.

Shamir's Secret Sharing Scheme. We decided to present SSS under the preliminaries section firstly because it is one of the most well-known and researched secret sharing schemes, and secondly for the readers to have an intuition of which is the end goal of this paper. Our following experiments are entirely based on this scheme.

The SSS scheme is an algorithm first proposed in 1979 by the renowned Israeli cryptographer Adi Shamir [3]. Shamir's (t, n)-threshold scheme, where t is the reconstructing threshold and n is the size of the set $S = s_1, \ldots, s_n$ of shareholders, makes use of Lagrange interpolation. The core components of SSS scheme consist of two key algorithms: the *Share* algorithm and the *Reconstruction* algorithm.

The *Share* algorithm takes as input a message $m \in \mathbb{F}_q$, where q is a prime number. It selects a polynomial $p(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ of degree $\deg(p(x)) = t-1$, where $a_0 := m$ and coefficients $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ are chosen uniformly at random. It computes share $\sigma_i \in \mathbb{F}_q$ for shareholder $s_i \in S$ as a point on polynomial $p(x)$, i.e. $\sigma_i := (i, p(i))$, where i is the ID of shareholder s_i. It distributes share σ_i to shareholder s_i through an information-theoretically secure channel, for $i = 1, \ldots, n$.

Following, the *Reconstruct* algorithm takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \in S$ of shareholders, and with sufficient shares the interpolating polynomial $p(x)$ is reconstructed using Lagrange's interpolation formula and outputs message $m \in \mathbb{F}_q$ as $p(0) = m$.

Other schemes for sharing secrets, such as Blakley's [14] and Mignotte's [15], may offer alternative se-

curity assumptions, complexity, and performance trade-offs. SSS was chosen for this analysis because it strikes a balance between simplicity, efficiency, security, and versatility, which has led to its widespread adoption and preference across a wide range of applications. Due to its sole reliance on polynomial interpolation, SSS is relatively straightforward to comprehend and implement. In addition, it offers information-theoretic security, also known as *perfect security*, which means that the scheme's security is based on mathematical principles rather than computational assumptions and cannot be improved. This makes it exceptionally resistant to various attacks, including those utilizing quantum computers. It must be noted, however, that the fundamental security relies on one's trust in the dealer's honesty. If the dealer is dishonest, a share verification mechanism [16] must be used, which is beyond the scope of this paper and will be the subject of future research. The ability to customize the threshold is a crucial component of SSS. As we have seen, the threshold determines the number of shares necessary to reconstruct the secret, allowing for flexibility in establishing the desired level of security and availability. SSS can accommodate any number of participants. In addition, SSS is resistant to share loss and compromise. If the threshold is not exceeded, the secret cannot be reconstructed even if some shares are lost or compromised. Lastly, the protocol's robustness makes it suitable for situations where the integrity of shares cannot always be guaranteed.

3. POLYNOMIAL INTERPOLATION

The primary focus of this article is on the application of polynomial interpolation in SSS over finite fields, aiming to demonstrate that, in specific scenarios, alternative interpolation methods might offer better performance than the conventional Lagrange method. Following a concise overview of polynomial interpolation and its connection to polynomial evaluation, we analyze four interpolation techniques. Our exploration begins with the standard Lagrange interpolation, which Shamir also advocated in his scheme. Subsequently, we introduce two optimizations of the standard method: the Optimized Lagrange for the constant term and the well-established Barycentric Lagrange formula. While elucidating the Lagrange forms, we scrutinize the Newton interpolation and delineate its trade-offs compared to the Lagrange method. The concluding segment of this section will center on the Fast Fourier method, culminating in a succinct assessment of the advantages and limitations of each technique.

3.1. Two Ways of Polynomial Representation

Polynomial interpolation finds wide application in approximating complex curves, such as evaluating trigonometric functions or the natural logarithm. Its efficient computation makes it a preferred choice for sub-quadratic multiplication and squaring operations, exemplified by techniques like the Toom-Cook multiplication [17] and the Karatsuba multiplication [18]. In Euclidean geometry, two points determine a line, three points a parabola, and so forth. However, given the variety of approaches available, reconstructing a smooth function $f(x)$ from a set of collected sample values lacks a straightforward answer. The forthcoming analysis aims to establish a *mathematical language* that bridges the understanding between the reader and the authors.

Polynomials are commonly represented in two forms: (*i*) *coefficient-form* and (*ii*) *point-value form*. Nevertheless, every polynomial in coefficient form has a unique counterpart in point-value form and vice versa. The coefficient representation of the n-degree polynomial $p(x)$ is the vector of coefficients $p_0, p_1, \ldots, p_n$. Conversely, the point-value representation of the n-degree polynomial $p(x)$ consists of a set of n point-value pairs, denoted as $\{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, (x_n, P(x_n))\}$, where all x_i are distinct. For conciseness, let $y_i = P(x_i)$. Consequently, the shares will be denoted as: $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$.

The process of computing the point-value representation of a polynomial $P(x)$, bounded by degree n, given its coefficient representation $p \in \mathbb{R}^n$, is known as *evaluation*. By selecting n distinct points $x_0, x_1, \ldots, x_{n-1}$, for each x_i, where $i = 0, 1, \ldots, n-1$, $P(x_i)$ can be computed. The evaluation of the value of any arbitrary polynomial $P(x)$ of degree n at a given point x can be achieved in $\Theta(n)$ time, utilizing Horner's method [19]. This method employs the following formula: $P(x) = p_0 + x(p_1 + x(p_2 + \cdots + x(p_{n-2} + x(p_{n-1}))\cdots))$. Conversely, if one were to compute the point-value representation of a polynomial $P(x)$ of degree n, it would require $\Theta(n^2)$ time. On the other hand, the process of computing the coefficient representation of a polynomial $P(x)$, bounded by degree n, given its point-value representation $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, is referred to as *interpolation*.

3.2. Lagrange Interpolation

Our examination of interpolation methods commences with the classical Lagrange interpolation technique [20], widely regarded as the most prevalent and extensively utilized method for data interpolation. This method, initially proposed by Shamir in his seminal work, forms the cornerstone of many interpolation procedures. In the following, we provide a concise overview of its functioning and subsequently introduce two optimizations.

At the outset, given a set of $n+1$ points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, the interpolation problem, as described in Section 3.1, entails defining a unique polynomial $P(x)$ of degree n such that $y_i = P(x_i)$, for $i = 0, 1, \ldots, n$. For the set of points $(x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k)$, where $x_j \neq x_m$ for all

$j \neq m$, the Lagrange basis for polynomials of degree $\leq k$ for those points consists of the basis polynomials $\ell_0(x), \ell_1(x), \ldots, \ell_k(x)$, each of degree k. These polynomials satisfy $\ell_j(x_m) = 0$ if $m \neq j$, or $\ell_j(x_j) = 1$ if $m = j$. Each basis polynomial $\ell_j(x)$ can be expressed as:

$$l_j(x) = \prod_{0 \leq m \leq k} \frac{x - x_m}{x_j - x_m} \tag{1}$$

The Lagrange interpolating polynomial for those points through the corresponding values $y_0, y_1, \ldots, y_k$ is the linear combination:

$$L(x) = \sum_{j=0}^{k} y_j \ell_j(x) \tag{2}$$

In this context, $P(x_i) = y_i$ for all $0 \leq i \leq t$. Each basis polynomial possesses a degree of k, hence the summation $L(x)$ has a degree of $\leq k$, ensuring interpolation of the data as $L(x_m) = \sum_{j=0}^{k} y_j \ell_j(x_m) = \sum_{j=0}^{k} y_j \delta_{mj} = y_m$. The polynomial $P(x)$ satisfying $P(x_i) = y_i$ for all $0 \leq i \leq t$, as derived from the aforementioned expression $L(x) = \sum_{j=0}^{k} y_j \ell_j(x)$, is indeed unique.

3.2.1. Optimizations of Standard Lagrange Method

The primary objective of interpolation within the context of SSS is to ascertain the value of $P(0)$, representing the secret value. In our comprehension of SSS, it is unnecessary to interpolate the entire polynomial up to the maximum degree; solely the constant term necessitates extraction. Hence, by setting the variable x to zero, we introduce our initial optimization, Optimized Lagrange (Opt.L). Leveraging Lagrange polynomials, a simple optimization technique (achieved through substitutions and basic arithmetic operations), can reduce the number of multiplications by nearly 50%:

$$P(0) = \Big(\prod_{i=0}^{n-1} -x_i\Big) \sum_{i=0}^{n-1} \frac{y_i}{-x_i(\Pi_{j=0, j\neq i}^{n-1}(x_i - x_j))}. \tag{3}$$

The previously mentioned optimization lacks clarity in the existing literature. While numerous schemes employing SSS reference the standard Lagrange interpolation method, practical implementations frequently diverge, balancing security with speed. Hence, we contend that providing developers of SSS-based secret-sharing applications with access to and comprehension of this optimization technique would be invaluable, thus warranting its inclusion in our research. Nevertheless, subsequent sections will demonstrate the existence of superior optimizations.

3.2.2. Barycentric Lagrange Formula

Continuing with the Lagrange optimizations, the subsequent one is termed the Barycentric Lagrange (BL) formula. Here, we will provide a concise overview, adhering explicitly to the notation and definitions outlined in Berrut et al. [21]. Commencing with the standard Lagrange formula, the numerator of l_j in Equation 1 can be expressed as:

$$\ell(x) = (x - x_0)\big(x - x_1\big) \cdots \big(x - x_n\big) \tag{4}$$

divided by $x - x_j$. By defining the Barycentric weights:

$$w_j = \frac{1}{\prod_{k \neq i}(x_j - x_k)}, \qquad j = 0, \ldots, n, \tag{5}$$

we can thus write l_j as:

$$\ell_j(x) = \ell(x) \frac{w_j}{x - x_j}. \tag{6}$$

So, in the end, we define the polynomial $P(x)$ as:

$$P(x) = \ell(x) \sum_{j=0}^{n} \frac{w_j}{x - x_j} f_j. \tag{7}$$

The fundamental contrast between the current optimization and the standard Lagrange interpolation formula lies in the methodology employed for constructing the interpolation polynomial. The BL formula entails expressing the numerator of each Lagrange basis function as the quotient of the product of differences between the interpolation point and all other points, and the difference between the interpolation point and the current point. The determination of Barycentric weights is based on these differences, as shown in Equation 5, which are subsequently utilized to compute the Lagrange basis functions. Eventually, the polynomial is obtained by multiplying the Lagrange basis functions with the function values at the interpolation points while accounting for the Barycentric weights. Section 4 will show that the BL method significantly reduces interpolation times.

3.3. Newton Interpolation

Newton's interpolation formula provides an alternative approach to compute $P(x)$, given a set of $n + 1$ data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, where the x_i are distinct. The initial step of this method involves computing Newton's table of divided differences. This table constitutes a triangular array of numbers defined by the recursive formula:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+k}] - f[x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+k} - x_i} \tag{8}$$

with initial conditions $f[x_i] = f(x_i)$. The table has the following form:

$$\begin{array}{lllll}
x_0 & f[x_0] & & & \\
x_1 & f[x_1] & f[x_0, x_1] & & \\
x_2 & f[x_2] & f[x_1, x_2] & f[x_0, x_1, x_2] & \\
\vdots & \vdots & \vdots & & \vdots \\
x_n & f[x_n] & f[x_{n-1}, x_n] & f[x_{n-2}, x_{n-1}, x_n] & \ldots \; f[x_0, x_1, \ldots, x_n]
\end{array}$$

Once the table is computed, the subsequent step involves evaluating $P(x)$ using the following Newton interpolation formula: $\sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0) \cdot \cdots (x - x_{k-1})$, where the coefficients $f[x_0, x_1, \ldots, x_k]$ are the entries in the table. This formula can be interpreted as a weighted sum of terms $(x-x_0)\cdots(x-x_{k-1})$, where each term is weighted by the corresponding coefficient $f[x_0, x_1, \ldots, x_k]$.

It should be emphasized that when the interpolation points $x_0, \ldots, x_n$ are distinct, finding a polynomial passing through the points (x_i, y_i) is equivalent to solving a system of linear equations $A\mathbf{x} = \mathbf{b}$ that has a unique solution. The matrix A is determined by the choice of basis for the space of polynomials of degree n or less. In Newton interpolation, the basis functions are the set $N_j(x)_{j=0}^{n}$ defined recursively as $N_0(x) = 1$, and for $j = 1, \ldots, n$: $N_j(x) = \prod_{k=0}^{j-1}(x - x_k)$. These basis functions are structured such that $N_j(x)$ is a polynomial of degree j, and $N_j(x_k) = 0$ for $k = 0, \ldots, j-1$. Consequently, any polynomial of degree at most n can be expressed as a linear combination of these basis functions, i.e., $P_n(x) = a_0 N_0(x) + a_1 N_1(x) + \cdots + a_n N_n(x)$.

Updating the interpolating polynomial becomes straightforward as interpolation points are added because the basis functions $\{N_j(x)\}$, $j = 0, \ldots, n$, remain unchanged. New interpolation points can be added to update the interpolating polynomial using Newton's method, and then the divided differences table is updated accordingly. For instance, suppose we have a set of $n+1$ distinct points x_i, y_i, and we have computed the divided differences table for these points. If we wish to add a new point (x_{n+1}, y_{n+1}), we update the divided differences table by adding a new column to the right. The first entry in this new column will be the divided difference of the new point, which is simply the function value y_{n+1}. Subsequent entries in the column can be recursively computed using divided differences from the previous column. Specifically, the $(k+1)$-th entry in the new column is determined as follows: $f[x_{n+1}, x_0, \cdots, x_k] = \frac{f[x_{n+1}, x_1, \ldots, x_k] - f[x_n, x_0, \ldots, x_{k-1}]}{x_{n+1} - x_n}$. Here, $f[x_{n+1}, x_1, \ldots, x_k]$ represents the divided difference computed in the previous column, while $f[x_n, x_0, \ldots, x_{k-1}]$ denotes the divided difference from the same row but in the preceding column.

Once the divided differences table is updated, the updated interpolating polynomial can be computed using the same formula as before $P_n(x) = \sum_{k=0}^{n} f[x_0, \cdots, x_k] N_k(x)$, where $N_k(x)$ signifies the k-th Newton basis function.

3.4. Fourier Interpolation

As demonstrated earlier, polynomials can be defined over a field denoted as $\mathbb{F}$. However, in the context of cryptography, we have chosen to define them specifically on $GF(p)$, where p is a prime. The Fourier transform (FT) is typically defined on the infinite set $\mathbb{C}$, which is the field of complex numbers. For the intended scope of our research, it is necessary to establish its definition on $GF(p)$ as well. Subsequently, within this section, we will examine the methodology for achieving this objective through the utilization of the roots of unity on finite fields. The current research involves the utilization of the FFT to alter the representation of a polynomial $P(x)$, which has a degree less than n and consists of coefficients belonging to the field $GF(p)$, from a coefficients-based representation to an evaluation-based representation (and vice versa). The evaluation is conducted exclusively over n distinct field elements, which are the nth roots of unity. As a result, sufficient knowledge of the preliminaries that were described at the beginning is required.

Following, we analyze the DFT, the FFT, and lastly the inverses of both which will lead us to the interpolation.

DFT over Finite Fields. Consider a polynomial $P(x)$ of degree n expressed in coefficient form, denoted as $p \in \mathbb{F}^n$, where p is a column vector given by $p = [\, p_0 \;\; p_1 \ldots p_{n-1}]^T$. We aim to represent $P(x)$ in its point-value form by computing its values at the n distinct n^{th} roots of unity, namely $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$. Let y_j denote $P(\omega_n^j)$ for j ranging from 0 to $n-1$. The n-th order DFT is a mathematical operation that takes the coefficients of a polynomial P of degree $n-1$ as input. Its output is an n-dimensional vector, where each component represents the evaluation of P at one of the n-th roots of unity. Conventionally, we denote this operation as $y = DFT_n(p)$, where p signifies the polynomial of degree $n-1$. A visual depiction of this problem can be seen as a matrix multiplication:

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \tag{9}$$

What the DFT essentially provides is a transformation:

$$\begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{n-1} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{bmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \tag{10}$$

In the transformed relation, observe that x_i corresponds to ω^i. We designate the Vandermonde matrix of $1, \omega, \omega^2, \ldots, \omega^{n-1}$ as the *DFT matrix*. Alternatively, we can express this algebraically as:

$$F_k = \sum_{i=0}^{n-1} a_i \omega^{ki} \tag{11}$$

After rearranging, we derive the Inverse Discrete Fourier Transformation (IDFT) formula, designed to recompute the coefficients a_i.

$$a_i = \frac{1}{n} \sum_{k=0}^{n-1} F_k \omega^{-ik} \tag{12}$$

This formula arises from the symmetric properties inherent in the primitive n^{th} roots of unity. Similarly, the IDFT can also be expressed through matrix multiplication.

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)^2} \end{bmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{n-1} \end{pmatrix} \tag{13}$$

In line with the DFT matrix, we denote the matrix containing the inverse powers of ω as the *IDFT matrix*.

3.5. Fast Fourier Transform over Finite Fields

We know that Horner's algorithm [22] facilitates the computation of DFT_n in $\Theta(n^2)$ steps. However, when dealing with significantly large values of n (e.g., hundreds of thousands or millions), executing DFT_n may demand excessive time. An alternative algorithm for computing the DFT of size n in $\Theta(n \log n)$ operations, known as the FFT, addresses this challenge. This approach leverages the symmetric properties of the nth roots of unity and is rooted in the technique introduced by Cooley and Tukey [23]. We adhere explicitly to the mathematical notations and definitions outlined by Fateman [24], as they offer a more algebraic perspective and are utilized in our implementations.

Fast Fourier Transform. The FFT algorithm leverages the symmetry and periodicity properties inherent in roots of unity to compute the Fourier transform more efficiently. By employing the Cooley and Tukey algorithm [23], the complexity is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ operations. The fundamental concept involves breaking down the initial DFT problem into smaller DFT problems of half the size, employing a recursive divide-and-conquer approach. This characteristic makes the FFT particularly valuable for handling large datasets and real-time applications. For demonstration purposes and to showcase the algorithm's optimal performance, we opt for the number of shares n to be a power of two, i.e., $n = 2^k$. While the same methodology can be extended to other values of n, such as $n = p^k$ or $n = p_1 p_2 p_3 \dots p_k$, these choices may not underscore the significant advantage of the Cooley and Tukey algorithm as effectively.

Consider the polynomial P defined as $P(x) = \sum_{i=0}^{n-1} a_i x^i$. We can partition P into two polynomials: one comprising the even-indexed coefficients of P, denoted as P_0, and another consisting of the odd-indexed coefficients, denoted as P_1.

$$P(x) = \underbrace{(a_0 + a_2 x^2 + \ \dots \ a_{n-2} x^{n-2})}_{P_0} + \underbrace{(a_1 x + a_3 x^3 + \ \dots \ + a_{n-1} x^{n-1})}_{P_1} \tag{14}$$

$$P(x) = \underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{2i} x^{2i}}_{P_0} + \underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} x^{2i+1}}_{P_1} \tag{15}$$

$$P(x) = \underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{2i} (x^2)^i}_{P_0} + x \underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} (x^2)^i}_{P_1} \tag{16}$$

Notice that the first term in the summation represents the evaluation of the $(n/2)$-degree polynomial P_0 at x^2, while the second term represents the evaluation of the $(n/2)$-degree polynomial P_1 at x^2. Consequently, we can express P as:

$$P(x) = P_0(x^2) + xP_1(x^2) \tag{17}$$

We iterate this procedure for each of P_0 and P_1 until the size of the FFT reduces to 1 and can no longer be split further. By recursively evaluating the smaller-sized FFTs and substituting these evaluations into the polynomial of the higher layer, we eventually derive the shares of the original polynomial.

The Inverse Fourier Transform is the problem of determining the left-hand side coefficient vector of the following relation:

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)^2} \end{bmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{n-1} \end{pmatrix} \tag{18}$$

The algorithm resembles the standard Fourier Transform, but in this case, the required matrix comprises negative powers of ω. Thus, the objective is to apply the FFT as described above, but this time evaluating at negative powers. As a final step, the Inverse Fourier Transform (IFFT) divides the inner product of each multiplied vector by n to obtain the coefficients.

Section 4 presents our testbed environment and the rationale behind selecting the respective parameters.

4. EVALUATION

After establishing the essential mathematical groundwork and conducting a thorough analysis of each interpolation method, this section encapsulates the culmination of our experiments. The experimental results demonstrate how various parameters influence the efficiency and performance of each interpolation method.

This contribution is motivated by the lack of related work on this topic, particularly when combined with secret sharing. To underscore the feasibility and practicality of our approach, we implemented a prototype using Python 3.11.18 and SageMath 10.3 [9] on a MacBook running macOS Sonoma 14.4.

4.1. Domain Parameters Configuration

Considering that our interpolation techniques will be employed for SSS across finite fields, defining the finite field within the specified domain we will be working with is crucial. The domain parameters encompass a set of predefined characteristics that hold universal applicability and will be consistently utilized across all experimental procedures. These parameters are influenced by the constraints of the FFT in the following manner.

The initial step involves constructing the finite field $\mathbb{F}_p$. Since the FFT does not function with a randomly chosen p, we must manually construct it. To compute the primitive n^{th} roots of unity, the division $\frac{p-1}{n}$ must yield an integer, indicating that n divides $p-1$. We opt for p to be a safe prime following the format $p = n_{max}q + 1$, where q represents a random 28-bit prime number. Here, we set n_{max} to 2^{100} as an upper bound since, in our specific use cases, n will never reach such magnitudes. Hence, the value of p is determined as:

$$p = 2^{100} * 135783853 + 1 = 172126482756751667503106328023403593729 \quad (19)$$

The size of p is arbitrary as it does not affect the performance of the interpolation methods.

Another important domain parameter crucial mostly for FFT is the group generator g. We can find g using any classical method for determining generators, but for simplicity, as stated by Lynn et al. [25], we let $g = 3$.

For each specific scenario, a 128-bit AES key will be consistently designated as the constant term of the polynomial and treated as the secret value s. The selection of a 128-bit key aligns with the length of the prime number p (Equation (19)). While the algorithm could support longer keys, such as those with 256 or 512 bits, maintaining equivalent security necessitates proportionally increasing the length of the prime number. Deviating from this balance by selecting a shorter prime number or extending the key length beyond what is proportional compromises security. Shorter primes risk insecurity due to modular arithmetic properties and loss of information, while longer keys necessitate padding. The objective is to distribute this confidential information among participants of the secret-sharing protocol to securely reach a consensus on the same key.

TABLE 1. Performance of unoptimized Newton, Barycentric and FFT.

Methods	128	256	512	1024	2048	4096	8192
Newton	0.05	0.12	0.6445	1.9967	8.8905	31.8654	320.3828
Barycentric	0.1483	0.3906	0.9381	2.998	12.5994	38.0322	114.7491
FFT	0.0025	0.0081	0.0112	0.0344	0.1144	0.1393	0.2699

4.2. A Comparative Analysis of Different Interpolation Methods

Regarding our initial experiment, certain adjustments are necessary to accommodate the constraints of the FFT algorithm.

1. The number of shares n should be a power of two. Specifically, we set $n_{min} = 2^7$ and $n_{max} = 2^{13}$ as these selections effectively highlight performance differences.
2. Maintaining consistency in the value of the primitive root, ω, across various n necessitates minor adjustments, which are not aligned with our objectives. While precomputing the roots of unity could potentially resolve this issue, our current approach utilizes unoptimized versions of each algorithm.
3. Due to the considerations above, it is imperative that the degree of the interpolating polynomial remains $n-1$ for all values of n.

Consequently, it is not possible to determine a fixed interpolating polynomial, necessitating the generation of an arbitrary polynomial of degree $(n-1)$ for each value of n.

We will now analyze the results obtained from the unoptimized implementations of the Newton, Barycentric, and FFT algorithms. Each cell within the Table 1 denotes the duration, measured in seconds, required to complete the interpolation process. It is notable, based on the FFT algorithm's computational complexity of $\mathcal{O}(n \log_2 n)$, its performance outperforms by far all the other interpolation methods.

We notice that for values of n up to 4096, both Newton and Barycentric methods demonstrate nearly comparable performance. However, Newton's time complexity undergoes a notable and rapid increase beyond this threshold. The disparity between the computational complexities of $\mathcal{O}(n \log_2(n))$ and $\mathcal{O}(n^2)$ becomes evident in Figure 1, as demonstrated by the logarithmic nature of the FFT graph, which converges towards a constant $\mathcal{O}(1)$ with an infinitesimally small rate of change.

In the second segment of our initial experiment (i.e., the standard Lagrange method), we set the lower bound n_{min} to 100 and the upper bound n_{max} to 1000, incrementing the value of n by 100 in each successive iteration. Due to the absence of constraints associated with the FFT, it becomes feasible to establish a deterministic solution for a randomly selected polynomial of degree five. The outcomes are

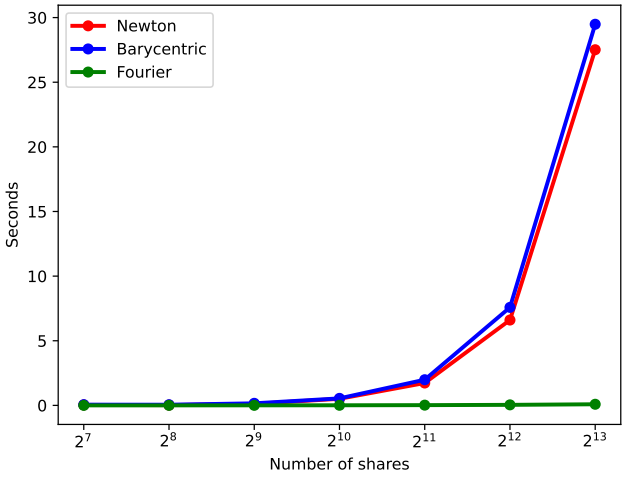

FIGURE 1. Interpolation times for Unoptimized Newton, Barycentric and FFT diagram.

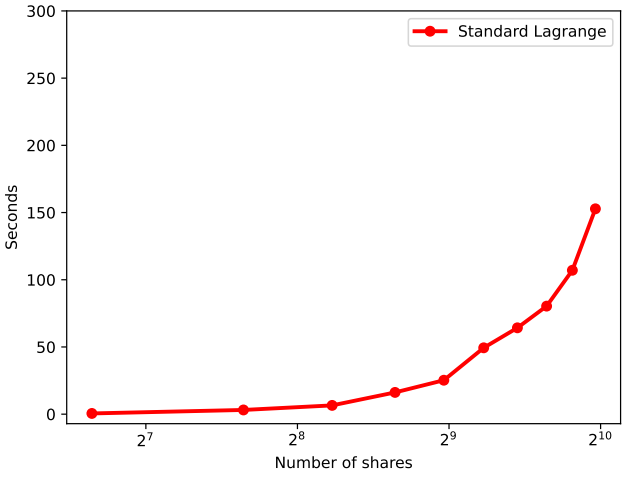

FIGURE 2. Naive Lagrange method for increasing number of members.

TABLE 2. Performance of unoptimized naive Lagrange.

Number of users	Interpolation Time (s)
100	0.5433
200	3.1385
300	6.5299
400	16.1861
500	25.2504
600	49.3655
700	64.2438
800	80.3796
900	107.0505
1000	152.7803

depicted in Table 2.

It is evident that for $n = 1000$, the Lagrange algorithm necessitates approximately two minutes to complete, as depicted in Figure 2. Naive Lagrange interpolation time increases quickly even with a small number of shares ($n = 1000$), as anticipated due to its quadratic time complexity $O(n^2)$.

4.3. Enhancing Newton and Barycentric Methods with New Shares

This experiment aims to explore an application employing secret sharing for key exchange. Both the enhanced Newton and Barycentric methods utilize precomputed divided differences and weights. These precomputed values are reused for each new user added to the system, eliminating the need for recalculations from scratch.

To illustrate this concept, we set the variable representing the number of shares to $n = 5000$. The precomputed values are stored and gradually expanded with each new share until the value of n reaches 5010. Let us first examine the scenario where no values have been precomputed. Table 3 exhibits the results obtained from the Newton and Barycentric methods when the precomputation of divided differences and weights is not conducted. It is crucial to note that the utilized versions of these approaches are optimized, focusing solely on recovering the constant term a_0 rather than the entire polynomial. Each cell denotes the duration in seconds required to complete the interpolation process. The advantage of Barycentric over Newton is evident from the very beginning with the $n = 5000$ shares. The number of shares is selected arbitrarily and is intentionally set to a large value to represent a real-life scenario. This is in contrast to the $n = 1000$ used in the naive Lagrange method, which is not suitable for high values of n and has no impact on the result. Moreover, the time advantage persists even when incorporating additional shares. Bezzateev et al. [6] proposed that using Newton interpolation for adding a single new share in the multifactor authentication setting is beneficial, compared to naive Lagrange. While this is correct, we bolster this assumption by demonstrating that the Barycentric technique is more advantageous, even for a larger number of new shares. In the subsequent experiments, we showcase that the FFT outperforms all methods with the correct setup.

Now, let us examine the outcomes of the Newton and Barycentric methods when utilizing precomputed values with the same number of shares. Table 4 highlights the significant difference in time complexity when divided differences and weights are precomputed. For the initial value $n = 5000$, the time in the first column is nearly identical to Table 3, as the experiment is repeated to establish the initial divided differences and weights. However, subsequent interpolations with $n + 1$ shares show notable changes in interpolation time. This is because interpolations with $n + 1$ shares utilize the precomputed n divided differences and weight values, while interpolations with $n + 2$ shares utilize the precomputed $n + 1$ values, and so forth. Both methods demonstrate substantial improvements in interpolation times compared to Table 3, with the Barycentric method still maintaining a computational

TABLE 3. Performance of Newton and Barycentric without precomputed values.

Methods	5000	5001	5002	5003	5004	5005	5006	5007	5008	5009	5010
Newton	8.5294	9.376	8.7036	9.3542	9.3054	9.5331	8.9018	8.8316	10.1682	9.0897	8.8205
Barycentric	5.6065	5.5197	5.7283	5.6771	5.6474	5.6487	5.7032	5.5013	5.6746	5.7882	5.524

TABLE 4. Performance of Newton and Barycentric with precomputed values.

Methods	5000	5001	5002	5003	5004	5005	5006	5007	5008	5009	5010
Newton	8.557	0.0085	0.0086	0.0087	0.3562	0.0082	0.008	0.0083	0.008	0.0083	0.0083
Barycentric	5.7163	0.0091	0.0092	0.0088	0.0086	0.0092	0.009	0.009	0.0095	0.0093	0.0098

TABLE 5. Performance of FFT and optimized standard Lagrange, Newton and Barycentric.

Methods	128	256	512	1024	2048	4096	8192
Standard Lagrange	0.0112	0.0465	0.1719	0.6817	2.7803	10.9333	43.7564
Newton	0.0322	0.0194	0.1042	0.4909	1.6527	6.1578	27.8844
Barycentric	0.004	0.0157	0.0615	0.246	0.9626	3.7881	15.1438
FFT	0.001	0.0019	0.0044	0.0092	0.019	0.0405	0.0864

advantage over Newton.

It can be concluded that irrespective of the application, employing Barycentric interpolation is more advantageous for a secret-sharing application experiencing a progressive increase in the number of users.

4.4. Comparison of All Optimized Interpolation Methods

In our final experiment, we compare all the aforementioned interpolation methods. It should be noted that the restrictions of the FFT method are considered, such as the requirement for the number of shares n to be a power of 2. All approaches are optimized by precomputing values for divided differences in Newton's method, weights in Barycentric Lagrange's method, and roots of unity in the FFT method. This experiment aims to determine the most effective interpolation method for standard SSS based only on performance metrics such as interpolation time and shared secret retrieval a_0 among n users. Table 5 presents interpolation times in seconds for varying numbers of shares. It is notable, that the FFT method does not even surpass 1 second for $n = 16384$ shares.

Furthermore, Figure 3 illustrates all optimal variations of the methodologies. Computational bounds are set at $n_{min} = 2^7$ and $n_{max} = 2^{13}$. The FFT approach demonstrates almost a straight line on the $x - y$ plane due to its time complexity of $O(nlog_2(n))$, which is significantly lower than the $O(n^2)$ complexity of other methods, except Barycentric, which has an $O(n)$ retrieval time with precomputed weights.

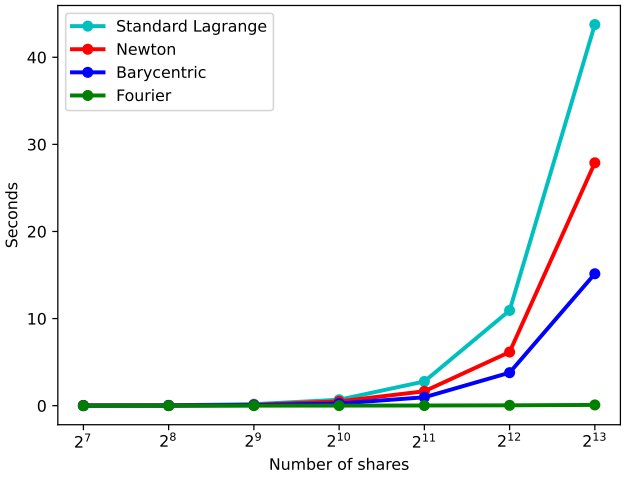

FIGURE 3. Optimized methods performance.

5. DISCUSSION

Through an examination of the research questions, we conducted an in-depth investigation into the practicalities and relative efficiencies of different polynomial interpolation techniques as they apply to secret-sharing schemes and more extensive real-world implementations. This research has been thorough, encompassing both theoretical frameworks and empirical validations through carefully designed experiments. Our journey through this analytical process illuminated the distinct capabilities, limitations, and situational advantages of the FFT, the BL formula, and Newton's interpolation method. Every technique was evaluated in terms of its computational complexity, practicability in finite fields, and ease of implementation in dynamic environments. Our main objective while analyzing these methodologies was to unequivocally respond to the research questions that were introduced at the beginning of this paper. Having reached the final phase of our investigation, equipped with a comprehensive body of evidence and insights derived from our rigorous analysis, we confidently conclude and respond to the initial research questions. This serves as concrete proof of our dedication to expanding the knowledge of the research community into applying polynomial interpolation methods into practical, large-scale applications, in addition to being an endorsement of the depth of our investigation.

RQ1:What is the swiftest method and most suitable in large-scale scenarios to interpolate the polynomial in the context of SSS to recover the secret from the constant term?

The FFT emerges as the premier choice for poly-

nomial evaluation and interpolation in SSS scheme, thanks to its unparalleled computational efficiency and rapid execution times, particularly when handling large datasets. Shamir's scheme relies on constructing a polynomial of degree $k-1$ for a k-threshold scheme, where the secret is embedded in the constant term. For reconstruction, traditionally, Lagrange interpolation is used, which has a computational complexity of $O(n^2)$ for n shares. In contrast, FFT can perform the necessary polynomial evaluations or interpolations in $O(n \log n)$ time. This efficiency is largely due to the FFT's ability to perform polynomial operations in $O(nlogn)$ time, leveraging divide-and-conquer strategies [23] and the discrete Fourier transform's symmetry properties. Such a significant reduction in computational complexity positions the FFT as exceptionally advantageous for secret reconstruction within large-scale operations. Nonetheless, the FFT requires the number of participants and the polynomial's degree to adhere to powers of two, a constraint that can be circumvented through algorithmic enhancements designed to broaden its applicability [26].

On the other hand, the Lagrange interpolation method and its evolution through the BL formula offer a different set of advantages and limitations. While the Lagrange method is renowned for its versatility, it faces practical difficulties in dynamically changing scenarios, such as the addition of new participants to a sharing scheme. This limitation arises from the need to recalculate Lagrange polynomials from scratch with each new addition, rendering the process cumbersome in rapidly evolving settings.

The BL formula, in contrast, addresses some of the practical challenges inherent in the standard Lagrange approach by eliminating the need for complete polynomial recalculations for every new interpolation point. This refinement significantly enhances efficiency, particularly in large datasets, by reducing the overall computational demand to $O(n)$ for evaluating interpolating polynomials once Barycentric weights are computed. The primary challenge of BL method lies in managing small datasets where the overhead of computing Barycentric weights may outweigh its computational benefits.

Furthermore, Newton's interpolation method, while sharing foundational similarities with the Lagrange approach in terms of employing a specific set of basis functions, distinguishes itself through the utilization of divided difference polynomials. This method offers advantages in computational efficiency for iterative polynomial evaluations, albeit at the cost of increased memory complexity due to the storage requirements of the divided difference table. Despite these advantages, Newton's method is not the primary choice compared to the FFT and BL methods, especially when quick interpolation of polynomials at new points is required.

In delving deeper into the comparative analysis of these methods, it becomes evident that the FFT stands out for its robust applicability and superior efficiency in large-scale secret-sharing schemes, assuming its specific constraints are effectively managed. The BL formula, with its focus on overcoming the limitations of the Lagrange method in dynamic and large datasets, provides a complementary approach that excels under certain conditions, particularly in environments not restricted by the size of the dataset. Consequently, when assessing the most suitable interpolation method for large-scale scenarios, the FFT unequivocally offers the most promising solution, subject to further refinements and adaptations to extend its utility beyond the confines of power-of-two constraints, reinforcing its theoretical and experimental preeminence over alternative interpolation techniques.

RQ2: What is the quickest approach to interpolate the polynomial for recovering the secret when adding additional members/shares?

This question lacks a clear and direct response and is more complex to be answered. While one might anticipate that the FFT remains the swiftest technique, its constraints in this particular context, such as accommodating the addition of new shares, are pivotal. To effectively employ the FFT approach, it becomes imperative to incorporate additional shares in batches that align with powers of 2. For instance, if 16 users intend to integrate additional members and allocate shares accordingly, the subsequent feasible limit would be 32. In our experimentation concerning the addition of new shares, we specifically evaluated the BL and Newton methods. The findings revealed that the BL method exhibited superior speed when integrating a small number of new shares, irrespective of the application or scenario. This observation contradicts the assertions made by Bezzateev [6], which suggested that the Newton method is more suitable for accommodating new members in a multifactor authentication protocol. Future endeavors entail integrating the FFT method and delving into incorporating new shares in batches. This exploration may encompass examining various adaptations of the FFT method, such as radix forms [27] or the Bluestein algorithm [28, 29], to mitigate any constraints encountered.

RQ3: Do optimizations (e.g., precomputing values) affect the overall performance of interpolation methods?

Precomputing values is a crucial optimization step for all the previously discussed methods. It is noteworthy that, even with a substantial value of n, these precomputed values do not significantly affect the memory bandwidth of our testing setup. Nonetheless, in alternative experimental contexts, such as IoT devices, memory utilization emerges as a critical factor demanding meticulous attention. Implementing our approaches and experiments on constrained devices poses a challenge we intend to address in future

endeavors.

6. RELATED WORK

The problem of secret sharing was firstly introduced by A. Shamir [3], along with the so-called Shamir threshold secret sharing schemes, based on Lagrange's interpolation. At around the same time, Blakley proposed his secret sharing scheme [14]. Blakley's secret sharing scheme tackles the secret sharing problem from the scope of hyper plane geometry. To implement a (t, n) threshold scheme, each of the n participants is given a hyperplane equation in a t dimensional space over a finite field such that each hyper plane passes through a certain point. The intersection point of the hyper planes is the original secret. When t participants come together, they can solve the system of equations to find the original secret. The secret is a point in a t dimensional space and n shares are affine hyper planes that pass through this point. The secret and the shares can be summarized as a linear system $Cx = y$, where matrix C and the vector y correspond to hyperplane equations. However, Blakley's scheme has a few weaknesses. First, there are no actual implementations. In his original paper, Blakley presented only a guideline for outlining a matrix of linear systems for perfect secrecy; no actual matrix was presented. In addition, each shareholder is aware that the point (which is the secret) lies in his share (hyperplane), so the method is not perfect like Shamir's. Finally, it is not effective with space.

However, researchers started to use Blakley's geometry-based secret-sharing approach in the area of secret image sharing [30, 31]. Ulutas et al. [32] introduced an improved scheme for secret image sharing, which adopts Blakley's secret sharing method with another method to share the secret and create meaningful shares. Moreover, Bozkurt et al. [33] explained the first threshold RSA signature approach using Blakley's secret scheme.

Since the pioneer publications of Shamir's and Blakley's, several other methods have been proposed. Another mathematical category from which many ideas have been driven is the Chinese remainder theorem. Mignotte [15] and Asmuth and Bloom [34] are both based on the Chinese remainder theorem for coprime modules over the ring of integers $\mathbb{Z}$. Chinese remainder theorem uses a well known property of integers, the so-called Euclidean division, which is not exclusive to $\mathbb{Z}$. Other rings that admit it are called Euclidean Domains. For this work, only two of them are highlighted , more specifically the ring of univariate polynomials over a field $\mathbb{K}[X]$, and the ring of Gaussian Integers $\mathbb{Z}[i]$.

Mignotte's threshold secret sharing scheme uses special sequences of integers, referred to as the *Mignotte sequences*. A generalization of Mignotte's scheme by allowing modules that are not necessarily pairwise coprime was proposed in [35], by introducing *generalized Mignotte sequences*. More generalizations of Mignotte's scheme are of Galibus and Matveev [36], who provided a version over polynomial rings of one variable and showed the remarkable property that any access structure can be realized by a Mignotte polynomial SSS. Finally, in [37] an extension of Mignotte threshold scheme to the ring of Gaussian Integers is proposed. Gaussian Integers play a role in digital signal processing, cryptography, coding, and many other fields of science and technology [38]. Mignotte's threshold scheme has already been proposed for image sharing [39], which will be very useful for future work, if biometrics are considered as images and try to merge biometric authentication with threshold secret sharing. Authors in [40], improved the extension of Mignotte SSS to Gaussian Integers proposed in [37], because in the Euclidean division over $\mathbb{Z}[i]$, quotient and remainder are not unique, so they proposed a new version of Mignotte's SSS over $\mathbb{Z}[i]$ that works properly, unlike [37].

Regarding the Asmuth and Bloom [34] scheme, the main idea is to create coprime integers $(m_0, m_1, \ldots, m_n)$, and have a secret s. Later, $s_i = S + \alpha m_0 (mod m_i)$ needs to be computed, and with the CRT after collecting enough shares, the secret is retrieved successfully. Extensive research work exists, studying the perfectness or the information rate of Asmuth-Bloom's scheme. Goldreich [41]recommended to choose $m_0, m_1, \ldots, m_n$ being primes as close as possible and proved that $t-2$ shares or less give no information on the secret for a $(t; n)$-threshold scheme. Lastly, Quisquater [42] showed that Asmuth-Bloom's scheme with moduli being consecutive primes is asymptotically ideal. However, for fixed values of moduli, the scheme always has a lower information rate (less than 1), especially for not-too-large moduli. Moreover, Asmuth and Bloom's scheme does leak some probabilistic information. Any particular value of the shared secret is possible; however different values of the shared secret have different numbers of possible secret value α. If it is assumed that the attacker knows the probability distribution that α was drawn from, he can then assign different probabilities to the values of the shared secret, should he choose to. However, besides the information leakage, with some modifications to overcome this problem, a few schemes have been proposed based on Asmuth-Bloom's idea, like [43, 44, 16, 45]

We will now examine a group of research papers that examine various interpolation methods in SSS, but it must be noted that the literature is limited. First, the paper by Bezzateev [6] investigates the use of Newton interpolation as an alternative to Lagrange and proposes it as a suitable method for organizing Multi-Factor-Authentication schemes. Nevertheless, their approach is more theoretical and lacks implementation results and comparisons. In addition, the same theoretical approach is utilized in [46], where one of the authors of the current work proposed Newton

interpolation for the SSS scheme in the context of distributed key generation protocols, however also lacking implementation results.

As already mentioned, Kogan [5] presents a novel modification of Shamir's polynomial secret sharing, where the secret equals the leading coefficient of the polynomial (as opposed to the free coefficient as in the original scheme) and the polynomial is reconstructed by Newton interpolation (rather than Lagrange interpolation). An additional proposition for the utilization of Newton's interpolation was put out by one of the authors of the present study, although in a theoretical manner and within the framework of distributed key generation (DKG) [46].

Lastly, Tomescu [7] notes, for his DKG implementation for scalable threshold cryptosystems, that he makes use of FFT instead of Lagrange interpolation for the reconstruction of the secret sharing scheme, but does not elaborate on how this is achieved in greater detail.

7. CONCLUSION

In this paper, we presented an in-depth theoretical analysis of different interpolation methods used in SSS. More precisely, we conducted experiments with different user numbers to examine the performance and interpolation times required for reconstructing a shared secret, specifically a 128-bit AES key. Our experiments have conclusively displayed the superior efficiency of the FFT method. This outcome aligns seamlessly with theoretical expectations, transitioning from the quadratic time complexity of naive Lagrance and Newton methods, $O(n^2)$, to the highly efficient logarithmic time complexity of $O(n \log n)$ using Barycentric and FFT. The empirical evidence presented robustly supports this assertion. Integrating these optimized techniques into real-world applications, particularly emphasizing verifiable secret sharing, such as zero-knowledge-proof communication protocols, holds promise for enhancing overall performance. Overall, our findings underscore the significance of leveraging FFT methods in cryptographic protocols and pave the way for further advancements in optimizing secure computation processes.

ACKNOWLEDGMENTS

This research has received funding from European Commission's Horizon Europe and Digital Europe research and innovation programs under grant agreements No. 101082440 (CHRISS), No. 101128024 (CYBERUNITY), and No. 101120962 (RESCALE).

DECLARATIONS

Competing Interest Statement: The authors declare no conflict of interest.

Research Data/Code Availability: Data/code will be made available on request.

Compliance with Ethical Standards: This article does not contain any studies with human participants or animals performed by any of the authors.

REFERENCES

[1] I. T. L. C. S. Division, "Multi-party threshold cryptography: Csrc." `https://csrc.nist.gov/Projects/threshold-cryptography`, 2010 (accessed May 2, 2022).

[2] NIST, "Multi-party threshold cryptography." `https://csrc.nist.gov/projects/threshold-cryptography`, 2023. Accessed: 2023-12-26.

[3] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[4] O. Catrina, "Efficient secure floating-point arithmetic using shamir secret sharing.," in *ICETE (2)*, pp. 49–60, 2019.

[5] N. Kogan and T. Tassa, "Improved efficiency for revocation schemes via newton interpolation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 4, pp. 461–486, 2006.

[6] S. Bezzateev, V. Davydov, and A. Ometov, "On secret sharing with newton's polynomial for multi-factor authentication," *Cryptography*, vol. 4, no. 4, p. 34, 2020.

[7] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. G. Gueta, and S. Devadas, "Towards scalable threshold cryptosystems," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 877–893, IEEE, 2020.

[8] D. Boneh, S. Gorbunov, R. S. Wahby, H. Wee, C. A. Wood, and Z. Zhang, "BLS Signatures," Internet-Draft draft-irtf-cfrg-bls-signature-05, Internet Engineering Task Force, June 2022. Work in Progress.

[9] The Sage Developers, *SageMath, the Sage Mathematics Software System (Version x.y.z)*, YYYY. `https://www.sagemath.org`.

[10] K. Conrad, "Finite fields," *order*, vol. 73, p. 343, 2013.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[12] I. Stewart, *Galois theory*. CRC press, 2022.

[13] T. Brazelton, J. Harrington, S. Kannan, and M. Litman, "On consecutive primitive nth roots of unity modulo q," *Journal of Number Theory*, vol. 174, pp. 494–504, 2017.

[14] G. R. Blakley, "Safeguarding cryptographic keys," in *Managing Requirements Knowledge, International Workshop on*, pp. 313–313, IEEE Computer Society, 1979.

[15] M. Mignotte, "How to share a secret," in *Workshop on cryptography*, pp. 371–375, Springer, 1982.

[16] L. Harn, M. Fuyou, and C.-C. Chang, "Verifiable secret sharing based on the chinese remainder theorem," *Security and Communication Networks*, vol. 7, no. 6, pp. 950–957, 2014.

[17] Wikipedia contributors, "Toom–cook multiplication — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Toom%E2%`

80%93Cook_multiplication&oldid=1176614118, 2023. [Online; accessed 8-December-2023].

[18] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Doklady Akademii Nauk*, vol. 145, pp. 293–294, Russian Academy of Sciences, 1962.

[19] Wikipedia contributors, "Horner's method — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Horner%27s_method&oldid=1178819274, 2023. [Online; accessed 8-December-2023].

[20] Wikipedia contributors, "Lagrange polynomial — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Lagrange_polynomial&oldid=1183619771, 2023. [Online; accessed 8-December-2023].

[21] J.-P. Berrut and L. N. Trefethen, "Barycentric lagrange interpolation," *SIAM review*, vol. 46, no. 3, pp. 501–517, 2004.

[22] W. G. Horner, "Xxi. a new method of solving numerical equations of all orders, by continuous approximation," *Philosophical Transactions of the Royal Society of London*, pp. 308–335, 1819.

[23] 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.

[24] P. R. Fateman, "Cs282- algebraic algorithms," July 2002-2003.

[25] B. Lynn, "Notes on number theory." https://crypto.stanford.edu/pbc/notes/numbertheory/gen.html, 2024. 7/3/2024.

[26] R. Larrieu, *Arithmétique rapide pour des corps finis*. PhD thesis, Université Paris-Saclay (ComUE), 2019.

[27] M. Soni and P. Kunthe, "A general comparison of fft algorithms," *Pioneer Journal Of IT & Management*, 2011.

[28] L. Rabiner, R. W. Schafer, and C. Rader, "The chirp z-transform algorithm," *IEEE transactions on audio and electroacoustics*, vol. 17, no. 2, pp. 86–92, 1969.

[29] V. Sukhoy and A. Stoytchev, "Generalizing the inverse fft off the unit circle. sci rep 9 (1): 14443," 2019.

[30] C.-C. Chen, W.-Y. Fu, and C.-C. Chen, "A geometry-based secret image sharing approach.," *J. Inf. Sci. Eng.*, vol. 24, no. 5, pp. 1567–1577, 2008.

[31] H.-K. Tso, "Sharing secret images using blakley's concept," *Optical Engineering*, vol. 47, no. 7, p. 077001, 2008.

[32] M. Ulutas, V. V. Nabiyev, and G. Ulutas, "Improvements in geometry-based secret image sharing approach with steganography," *Mathematical Problems in Engineering*, vol. 2009, 2009.

[33] I. N. Bozkurt, K. Kaya, A. A. Selcuk, and A. M. Güloglu, "Threshold cryptography based on blakley secret sharing," *Information Sciences*, pp. 1–4, 2008.

[34] C. Asmuth and J. Bloom, "A modular approach to key safeguarding," *IEEE transactions on information theory*, vol. 29, no. 2, pp. 208–210, 1983.

[35] S. Iftene, "A generalization of mignotte's secret sharing scheme," in *Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania*, pp. 196–201, 2004.

[36] T. Galibus and G. Matveev, "Generalized mignotte's sequences over polynomial rings," *Electronic Notes in Theoretical Computer Science*, vol. 186, pp. 43–48, 2007.

[37] I. Ozbek, F. Temiz, and S. İrfan, "A generalization of the mignotte's scheme over euclidean domains and applications to secret image sharing," *Journal of Algebra Combinatorics Discrete Structures and Applications*, vol. 6, no. 3, pp. 147–161, 2019.

[38] R. E. Blahut, *Algebraic methods for signal processing and communications coding*. Springer Science & Business Media, 2012.

[39] G. Ulutaş, M. Ulutaş, and V. Nabiyev, "Secret sharing scheme based on mignotte's scheme," in *2011 IEEE 19th Signal Processing and Communications Applications Conference (SIU)*, pp. 291–294, IEEE, 2011.

[40] D. Munuera-Merayo, "On mignotte secret sharing schemes over gaussian integers," *arXiv preprint arXiv:2104.06361*, 2021.

[41] O. Goldreich, D. Ron, and M. Sudan, "Chinese remaindering with errors," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pp. 225–234, 1999.

[42] M. Quisquater, B. Preneel, and J. Vandewalle, "On the security of the threshold scheme based on the chinese remainder theorem," in *International Workshop on Public Key Cryptography*, pp. 199–210, Springer, 2002.

[43] K. Kaya and A. A. Selçuk, "Threshold cryptography based on asmuth–bloom secret sharing," *Information sciences*, vol. 177, no. 19, pp. 4148–4160, 2007.

[44] L. Harn and M. Fuyou, "Multilevel threshold secret sharing based on the chinese remainder theorem," *Information processing letters*, vol. 114, no. 9, pp. 504–509, 2014.

[45] Y. Liu, L. Harn, and C.-C. Chang, "A novel verifiable secret sharing mechanism using theory of numbers and a method for sharing secrets," *International Journal of Communication Systems*, vol. 28, no. 7, pp. 1282–1292, 2015.

[46] A. Voudouris, I. Politis, and C. Xenakis, "Secret sharing a key in a distributed way, lagrange vs newton," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pp. 1–7, 2022.