Bol. Soc. Paran. Mat.
©SPM – E-ISSN-2175-1188
SPM: www.spm.uem.br/bspm

(3s.) **v. 2026 (44) 6** : 1–15.
ISSN-0037-8712
doi:10.5269/bspm.81058

Performance Improvement of the Chebyshev Polynomial Method through Matrix-Based Computation in Key Exchange: Results and Analysis

Khalid KHALLOUKI, Najat RAFI, Khadija BOUZKOURA and Abdelhakim CHILLALI

ABSTRACT: Cryptographic key-exchange protocols are fundamental tools for securing modern communication systems. The classical Diffie–Hellman protocol constitutes the reference model, relying on the hardness of the discrete logarithm problem in finite groups. Parallel to this line, Chebyshev polynomials have attracted attention due to their remarkable commutative property, $T_m(T_n(x)) = T_n(T_m(x)) = T_{mn}(x)$, suggesting interesting analogies with modular exponentiation. However, most existing Chebyshev-based schemes depend on recursive evaluation, which is computationally expensive and difficult to adapt reliably to finite fields. In this work, we present a significant development in Chebyshev-based cryptography. We introduce new algebraic formulas together with a matrix-based formulation that allows Chebyshev polynomials to be computed efficiently using fast exponentiation. This framework eliminates recursion, improves stability, and provides exact evaluation over finite fields. We then integrate this formulation into a key-exchange construction and compare it to the classical Diffie–Hellman scheme. Our results show that the proposed matrix formulation evaluates $T_n(x)$ in $O(logn)$ operations while preserving the structural properties required for key exchange. Although Diffie–Hellman remains more suitable for real-world deployment, our contribution offers a powerful and scalable alternative framework. These findings extend the current state of research on Chebyshev polynomials and highlight their potential for developing diversified cryptographic mechanisms.

Keywords: Diffie-Hellman, discrete logarithm problem, Chebyshev polynomials, Chebyshev-based cryptographic protocols, cryptographic performance.

Contents

1. Introduction

Key exchange is one of the fundamental building blocks of modern cryptography. Its purpose is to allow two communicating parties to establish a shared secret key over an insecure public channel, even in the presence of an adversary capable of observing all transmitted information. Once established, this shared key is typically employed to ensure confidentiality and integrity through symmetric encryption mechanisms. The essential security requirement is that the derived secret must remain computationally infeasible to reconstruct from the publicly exchanged data, an assumption grounded in the hardness of well-defined mathematical problems.

2020 *Mathematics Subject Classification:* 94A60.
Submitted December 27, 2025. Published February 17, 2026

The Diffie–Hellman protocol, introduced in the seminal work of Diffie and Hellman, constitutes the first practical realization of public-key cryptography and remains one of its most influential constructions. Its security relies on the presumed intractability of the discrete logarithm problem in finite cyclic groups, a problem that has been extensively analyzed and optimized over the past decades. Numerous algorithmic and implementation improvements, together with well-standardized parameters and widespread software and hardware support, have contributed to the maturity and efficiency of Diffie–Hellman and its elliptic-curve variant.

Despite this success, the emergence of quantum computing has raised serious concerns. Shor's algorithm shows that both integer factorization and discrete logarithms could be solved efficiently on a sufficiently powerful quantum computer, motivating the exploration of cryptographic constructions based on alternative algebraic structures.

In this context, Chebyshev polynomials have attracted growing interest. Their remarkable composition property,

$$T_n(T_m(x)) = T_{nm}(x),$$

which closely parallels the commutative property of exponentiation underlying the Diffie–Hellman protocol,

$$(g^a)^b = (g^b)^a = g^{ab}.$$

This structural analogy enables the construction of key exchange protocols based on polynomial composition rather than modular exponentiation, as initially proposed in polynomial-based cryptographic schemes [4,6]. Such approaches rely on the computational hardness of the Chebyshev Polynomial Discrete Logarithm Problem (CP-DLP), providing an alternative algebraic foundation for public-key cryptography.

However, in contrast to Diffie–Hellman, Chebyshev-based cryptographic protocols remain relatively immature. They suffer from limited optimized implementations, lack of widely adopted software libraries, absence of dedicated hardware acceleration, and increased computational cost when relying on naive recursive evaluations for large polynomial degrees. These limitations have hindered their practical deployment despite their theoretical appeal. It is worth noting that matrix-based computations have been used in cryptography since early algebraic ciphers such as the Hill cipher [9]. In this work, matrices are not employed for direct encryption, but rather as an efficient computational tool to evaluate Chebyshev polynomials over finite fields via fast matrix exponentiation.

In this work, we revisit this research direction from a new perspective. We propose a matrix-based formulation that enables exact computation of Chebyshev polynomials over finite fields using fast matrix exponentiation. By reformulating the recurrence relation as an exponentiation problem, the complexity of evaluating

$$T_n(x) \bmod p$$

is reduced from linear to logarithmic time, significantly improving scalability. This approach draws inspiration from matrix techniques historically used in classical ciphers, not for encryption itself, but as an efficient computational tool.

Our objective is to provide a rigorous and balanced comparison between Chebyshev-based key exchange and the classical Diffie–Hellman protocol. We demonstrate that the proposed matrix formulation not only overcomes several performance limitations of traditional implementations, but also contributes to reopening and advancing the study of polynomial-based cryptography as a complementary research avenue rather than a direct replacement for established standards.

The remainder of this paper is organized as follows. Section 2 presents the mathematical foundations required for this work. It first reviews the discrete logarithm problem and its role in cryptographic security, then introduces a matrix-based formulation of Chebyshev polynomials and discusses its algebraic properties. Section 3 focuses on key exchange protocols. It begins with a brief overview of the classical Diffie–Hellman protocol, followed by a detailed description of the Chebyshev-based key exchange scheme and its underlying principles. Section 4 is devoted to the experimental evaluation of the proposed approach. This section includes a performance comparison between matrix-based and traditional Chebyshev polynomial computations, an analysis of key exchange protocols based on Chebyshev polynomials versus Diffie–Hellman, and a discussion of the obtained results.Finally, Section 5 concludes the paper by summarizing the main contributions and outlining potential directions for future research.

2. Mathematical Foundations

2.1. Discrete Logarithm Problem

The Diffie-Hellman (DH) protocol operates in a finite multiplicative group, typically the cyclic group $\mathbb{F}_p^\times = \mathbb{Z}_p^*$, where p is a prime number, and $\mathbb{Z}_p^*$ denotes the set of integers invertible modulo p.

Modular Exponentiation. The core function of the protocol is modular exponentiation:

$$\mathrm{Exp}_g(a) = g^a \bmod p$$

where $g \in \mathbb{Z}_p^*$ is a generator of the group (also called a primitive element), whose successive powers generate all elements of the group.

Modular exponentiation can be computed efficiently using algorithms such as fast exponentiation (square-and-multiply method), with complexity $O(\log a)$ multiplications.

Discrete Logarithm Problem. The security of the protocol relies on the computational hardness of the Discrete Logarithm Problem (DLP): given g and $A = g^a \bmod p$, determining a is computationally infeasible for sufficiently large parameters (e.g., $p > 2^{2048}$).

This problem is considered hard for classical computers but vulnerable to quantum attacks via Shor's algorithm.

Commutativity Property. The exchange relies on the fundamental property of exponentiation commutativity in abelian groups:

$$(g^a)^b = g^{ab} = g^{ba} = (g^b)^a$$

This property ensures that both parties obtain the same key value without ever exchanging their secret exponents a and b.

2.2. Matrix-Based Formulation of Chebyshev Polynomials

Definition and Fundamental Properties. Chebyshev polynomials form a family of polynomials of major importance in mathematics, particularly in approximation theory, numerical analysis, and more recently in cryptography. While several types exist, cryptographic applications primarily use those of the first kind, denoted $T_n(x)$, defined by the recurrence relation:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2x \cdot T_n(x) - T_{n-1}(x).$$

These polynomials are of degree n and possess remarkable composition properties that make them interesting for cryptographic applications.

Composition Property: $T_n \circ T_m = T_{nm}$. An essential property of Chebyshev polynomials is their commutativity under composition:

$$T_n(T_m(x)) = T_m(T_n(x)) = T_{nm}(x),$$

which means that composing two Chebyshev polynomials is equivalent to applying a single polynomial with the product index.

This property is completely analogous to that of modular exponentiation used in the Diffie-Hellman protocol:

$$(g^a)^b = (g^b)^a = g^{ab}.$$

Thus, this commutative algebraic structure enables the construction of an alternative key exchange protocol where exponentiation is replaced by polynomial composition.

Definition over Finite Fields. In a cryptographic context, Chebyshev polynomials can be defined over a finite field $\mathbb{F}_q$ where q is a prime power ($q = p^r, r \in \mathbb{N}^*$). One can then consider the reduction modulo q of each polynomial coefficient.

For example, considering $T_n(x)$ over $\mathbb{F}_5$, we obtain:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2 - 1 \equiv 2x^2 + 4 \pmod 5.$$

These polynomials remain deterministic and recursive, enabling efficient implementation even over finite fields. They can therefore be used to simulate a key exchange protocol similar to Diffie-Hellman, but within a polynomial framework.

However, over finite fields, careful attention must be paid to the non-injectivity of certain polynomials and the limited domain cardinality, which can pose security problems if parameters are not properly chosen.

Chebyshev polynomials have remained largely theoretical, facing challenges such as high computational cost, lack of optimized implementations, and numerical instabilities over real numbers. Our matrix-based formulation offers the advantage of reducing memory consumption and accelerating computations, as demonstrated by the experimental results in the table (see Table 1).

Algebraic Representation via Matrix Recurrence. The Chebyshev polynomials $T_n(x)$ satisfy:

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad T_0(x) = 1,\ T_1(x) = x, \quad (n \geq 2).$$

Define for $(n \geq 1)$:

$$\mathbf{e} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{v}_n(x) = \begin{pmatrix} T_n(x) \\ T_{n-1}(x) \end{pmatrix}, \quad \mathbf{M}(x) = \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}.$$

Then:

$$\mathbf{v}_n(x) = \mathbf{M}(x)\mathbf{v}_{n-1}(x), \quad \mathbf{v}_1(x) = \begin{pmatrix} x \\ 1 \end{pmatrix}.$$

Thus:

$$\mathbf{v}_n(x) = \mathbf{M}(x)^{n-1}\mathbf{v}_1(x) \quad .$$

Implies:

$$\begin{pmatrix} T_n(x) \\ T_{n-1}(x) \end{pmatrix} = \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} x \\ 1 \end{pmatrix}.$$

Extracting $T_n(x)$:

$$T_n(x) = \mathbf{e}^{tr}\mathbf{v}_n(x) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} T_n(x) \\ T_{n-1}(x) \end{pmatrix}$$

$$\boxed{T_n(x) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} x \\ 1 \end{pmatrix}}$$

Note: All operations are performed in the finite field $\mathbb{F}_q$, i.e., modulo q.

3. Cryptographic Background

Key exchange constitutes a fundamental primitive in modern cryptography, enabling two communicating parties to establish a shared secret key over an insecure public channel. In the classical communication model, two entities, commonly denoted as Alice and Bob, seek to derive a common secret while an adversarial party (Eve) may observe all exchanged public information. The resulting shared key is subsequently used to ensure confidentiality and integrity through symmetric encryption mechanisms.

The essential security requirement of any key exchange protocol is that the derived secret remains computationally infeasible for an adversary to recover, even with full knowledge of the public parameters exchanged during the protocol execution. This security property relies on the hardness of well-defined mathematical problems and is commonly referred to as computational security.

3.1. Classical Diffie-Hellman Protocol

The Diffie–Hellman key exchange protocol, introduced by Diffie and Hellman in their seminal work [1], represents the first practical realization of public-key cryptography. Its security is based on the assumed intractability of the Discrete Logarithm Problem (DLP) in finite cyclic groups, which has led to extensive research and widespread adoption of number-theoretic cryptographic constructions.

Protocol Execution.

1. Alice and Bob publicly agree on a prime number p and a generator $g \in \mathbb{Z}_p^*$.
2. Alice chooses a secret integer a, computes $A = g^a \bmod p$ and sends A to Bob.
3. Bob chooses a secret integer b, computes $B = g^b \bmod p$ and sends B to Alice.
4. Each party computes the shared key:

$$K = B^a \bmod p = (g^b)^a \bmod p = g^{ab} \bmod p \quad \text{(Alice)}$$
$$K = A^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p \quad \text{(Bob)}$$

Extension to Elliptic Curves: ECDH. The Diffie-Hellman protocol can be generalized to operate over elliptic curve groups, giving rise to the Elliptic Curve Diffie-Hellman (ECDH) protocol [8,10]. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, and $G \in E(\mathbb{F}_q)$ a point of prime order n.

The protocol proceeds as follows:

1. Alice and Bob publicly agree on the curve parameters $(E, \mathbb{F}_q, G, n)$.
2. Alice chooses a private key $a \in [1, n-1]$, computes $A = a \cdot G$ and sends A to Bob.
3. Bob chooses a private key $b \in [1, n-1]$, computes $B = b \cdot G$ and sends B to Alice.
4. Each party computes the shared secret point:

$$S = a \cdot B = a \cdot (b \cdot G) = ab \cdot G \quad \text{(Alice)}$$
$$S = b \cdot A = b \cdot (a \cdot G) = ab \cdot G \quad \text{(Bob)}$$

The security of ECDH relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP): given points G and $A = a \cdot G$, determining a is computationally infeasible. ECDH offers equivalent security to classical DH with significantly smaller key sizes (e.g., 256-bit ECC vs 3072-bit DH) and is widely used in modern cryptographic systems such as TLS and SSH.

3.2. Chebyshev-Based Key Exchange Protocol

Key Exchange Protocol. The Chebyshev polynomial-based key exchange protocol operates as follows:

1. Alice and Bob publicly agree on: A finite field $\mathbb{F}_q$ and a public parameter $x_0 \in \mathbb{F}_q$.
2. Alice chooses a private integer $n = na$, computes $A = T_n(x_0) \bmod q$, and sends A to Bob.
3. Bob chooses a private integer $m = nb$, computes $B = T_m(x_0) \bmod q$, and sends B to Alice.
4. Each party computes the shared secret:

$$K_A = T_n(B) \bmod q = T_n(T_m(x_0)) \bmod q = T_{nm}(x_0) \bmod q \quad \text{(Alice)}$$
$$K_B = T_m(A) \bmod q = T_m(T_n(x_0)) \bmod q = T_{mn}(x_0) \bmod q \quad \text{(Bob)}$$

By the composition property $T_n(T_m(x)) = T_{nm}(x)$, both parties obtain the same value $K_A = K_B$, establishing a shared secret.

Example: Chebyshev Key Exchange over $\mathbb{F}_{11}$. Consider the finite field $\mathbb{F}_{11}$ with public parameter $x_0 = 3$.

1. **Initialization**: Public parameters: $\mathbb{F}_{11}$, $x_0 = 3$
2. **Alice's computation**:
   - Chooses private degree $n = na = 4$
   - Computes $T_4(3)$ over $\mathbb{F}_{11}$:

$$\begin{aligned}
&T_0(3) = 1\\
&T_1(3) = 3\\
&T_2(3) = 2 \cdot 3^2 - 1 = 18 - 1 = 17 \equiv 6 \pmod{11}\\
&T_3(3) = 2 \cdot 3 \cdot T_2(3) - T_1(3) = 6 \cdot 6 - 3 = 36 - 3 = 33 \equiv 0 \pmod{11}\\
&T_4(3) = 2 \cdot 3 \cdot T_3(3) - T_2(3) = 6 \cdot 0 - 6 = -6 \equiv 5 \pmod{11}
\end{aligned}$$

   - Sends $A = 5$ to Bob
3. **Bob's computation**:
   - Chooses private degree $m = nb = 7$
   - Computes $T_7(3)$ over $\mathbb{F}_{11}$:

$$\begin{aligned}
&T_5(3) = 2 \cdot 3 \cdot T_4(3) - T_3(3) = 6 \cdot 5 - 0 = 30 \equiv 8 \pmod{11}\\
&T_6(3) = 2 \cdot 3 \cdot T_5(3) - T_4(3) = 6 \cdot 8 - 5 = 48 - 5 = 43 \equiv 10 \pmod{11}\\
&T_7(3) = 2 \cdot 3 \cdot T_6(3) - T_5(3) = 6 \cdot 10 - 8 = 60 - 8 = 52 \equiv 8 \pmod{11}
\end{aligned}$$

   - Sends $B = 8$ to Alice
4. **Shared secret computation**:
   - Alice computes: $K_A = T_4(B) = T_4(8)$

$$\begin{aligned}
&T_0(8) = 1\\
&T_1(8) = 8\\
&T_2(8) = 2 \cdot 8^2 - 1 = 128 - 1 = 127 \equiv 6 \pmod{11}\\
&T_3(8) = 2 \cdot 8 \cdot 6 - 8 = 96 - 8 = 88 \equiv 0 \pmod{11}\\
&T_4(8) = 2 \cdot 8 \cdot 0 - 6 = -6 \equiv 5 \pmod{11}
\end{aligned}$$

     So $K_A = 5$
   - Bob computes: $K_B = T_7(A) = T_7(5)$

$$\begin{aligned}
&T_0(5) = 1\\
&T_1(5) = 5\\
&T_2(5) = 2 \cdot 5^2 - 1 = 50 - 1 = 49 \equiv 5 \pmod{11}\\
&T_3(5) = 2 \cdot 5 \cdot 5 - 5 = 50 - 5 = 45 \equiv 1 \pmod{11}\\
&T_4(5) = 2 \cdot 5 \cdot 1 - 5 = 10 - 5 = 5 \pmod{11}\\
&T_5(5) = 2 \cdot 5 \cdot 5 - 1 = 50 - 1 = 49 \equiv 5 \pmod{11}\\
&T_6(5) = 2 \cdot 5 \cdot 5 - 5 = 50 - 5 = 45 \equiv 1 \pmod{11}\\
&T_7(5) = 2 \cdot 5 \cdot 1 - 5 = 10 - 5 = 5 \pmod{11}
\end{aligned}$$

     So $K_B = 5$

Both parties have successfully established the shared secret $K = 5 \in \mathbb{F}_{11}$. This example demonstrates the practical application of the composition property $T_4(T_7(3)) = T_{28}(3) = T_7(T_4(3))$.

4. Experimental Evaluation

This section presents three structured experiments designed to evaluate the computation and cryptographic applicability of Chebyshev polynomials over finite fields.

4.1. Performance Comparison: Matrix-Based vs. Traditional Chebyshev Polynomial Computations

Experience I. This experiment aims to evaluate and compare the computational performance of two different approaches for computing the Chebyshev polynomial $T_n(x) \bmod p$: a classical iterative method and a matrix-based exponential method. The primary objective is to identify the most suitable computational strategy for cryptographic applications that require large operand sizes, high computational efficiency, predictable execution time, and controlled memory usage.

All experiments were conducted over the finite field $\mathbb{F}_p$, where p is a 256-bit prime constant defined as

p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffc2f,

and where $x \in \mathbb{F}_p$ is a global constant of 128 bits, defined by x=0xf1e2d3c4b5a6978877665544332211 00.

The implementations were developed using the Python programming language and executed within the Google Colab environment, which provides a reproducible and controlled computational platform suitable for performance evaluation.

Two computational methods were evaluated. The first is the classical recurrence-based formulation of Chebyshev polynomials, which relies on the standard iterative relation. The second is a matrix-based formulation originally derived by the authors in this work. This formulation is obtained by expressing the Chebyshev recurrence relation in matrix form, which enables the use of fast matrix exponentiation techniques and results in a logarithmic-time complexity with respect to the polynomial degree n.

It is important to emphasize that this matrix-based approach is not a standard method commonly found in the literature for Chebyshev polynomial computation. Rather, it represents an original contribution of this work, specifically designed to improve scalability and computational efficiency in cryptographic contexts.

Algorithm. The corresponding implementations of both methods are provided in Listings 1 and 2.

Listing 1: Recurrence-based Chebyshev computation

```
def tchebyshev_recurrence(x, p, n):
if n == 0:
return 1 % p
if n == 1:
return x % p

T0, T1 = 1 % p, x % p
for _ in range(2, n + 1):
T2 = (2 * x * T1 - T0) % p
T0, T1 = T1, T2
return T1
```

Listing 2: Matrix-based Chebyshev computation

```
def tchebyshev_matrix(x, p, n):
def matmul(A, B):
a, b, c, d = A
e, f, g, h = B
return (
(a*e + b*g) % p,
(a*f + b*h) % p,
(c*e + d*g) % p,
```

```
    (c*f + d*h) % p
    )

    if n == 0:
    return 1 % p
    if n == 1:
    return x % p

    M = (2*x % p, p-1, 1, 0)
    R = (1, 0, 0, 1)
    k = n - 1

    while k > 0:
    if k & 1:
    R = matmul(R, M)
    M = matmul(M, M)
    k >>= 1

    a, b, _, _ = R
    return (a*x + b) % p
```

To assess scalability, the polynomial degree n was varied across 13 representative values ranging from $n = 10$ to $n = 10^7$. Execution time was measured using the `time.perf_counter()` function, while peak memory consumption was monitored using Python's `tracemalloc` module. Each experiment was repeated multiple times, yielding consistent and reproducible results.

The numerical results obtained from the Python implementations are summarized in Table 1, while the execution time and memory consumption trends are illustrated in Figure 1. For small values of n (typically $n \leq 50$), both methods exhibit comparable execution times, with a slight advantage observed for the iterative method due to its minimal constant overhead.

As the value of n increases, the iterative method exhibits linear growth in execution time, leading to a rapid increase in computational cost. In contrast, the proposed matrix-based method demonstrates logarithmic growth, resulting in substantial performance gains. As shown in Table 1, speedup factors exceeding 70× are observed for $n = 5000$, with even greater improvements for larger values of n.

Table 1: Comprehensive Performance Analysis: Tchebyshev Polynomial Computation

n	Time (ms)		Memory (KB)		Performance Metrics		
	Recurrence	Matrix	Recurrence	Matrix	Speedup	Efficiency (%)	Memory Saving (%)
10	0.242	0.213	0.520	1.117	1.137	87.970	-1.2e+02
50	0.542	0.369	0.551	1.121	1.469	68.074	-1.0e+02
100	1.516	0.338	0.551	1.141	4.489	22.275	-1.1e+02
500	6.000	0.642	0.582	1.141	9.348	10.698	-9.6e+01
1000	12.206	0.583	0.582	1.160	20.944	4.775	-9.9e+01
5000	61.862	0.616	0.582	1.172	100.488	0.995	-1.0e+02
10000	124.791	1.978	11.476	1.776	63.079	1.585	84.520
50000	652.348	1.149	3.863	1.148	567.928	0.176	70.273
100000	1.3e+03	0.810	3.792	1.148	1.6e+03	0.064	69.714
500000	7.1e+03	1.647	3.795	1.148	4.3e+03	0.023	69.738
1000000	1.5e+04	1.762	3.841	1.148	8.4e+03	0.012	70.099
5000000	6.9e+04	1.106	3.358	1.176	6.3e+04	0.002	64.990
10000000	1.4e+05	1.353	3.841	1.176	1.0e+05	9.7e-04	69.387

Notes:

- Speedup = Time(Recurrence) / Time(Matrix). Values > 1 indicate matrix method is faster.
- Efficiency = (Time(Matrix) / Time(Recurrence)) × 100%.
- Memory Saving = [(Memory(Recurrence) - Memory(Matrix)) / Memory(Recurrence)] × 100%.
- All computations performed modulo p (256-bit prime) with x = 128-bit integer.

The memory usage analysis further highlights the efficiency of the proposed approach. While the iterative method shows a gradual increase in memory consumption as n grows, the matrix-based method maintains a low and stable memory footprint, approximately between 0.6 and 1.2 KB, as illustrated in Figure 1.

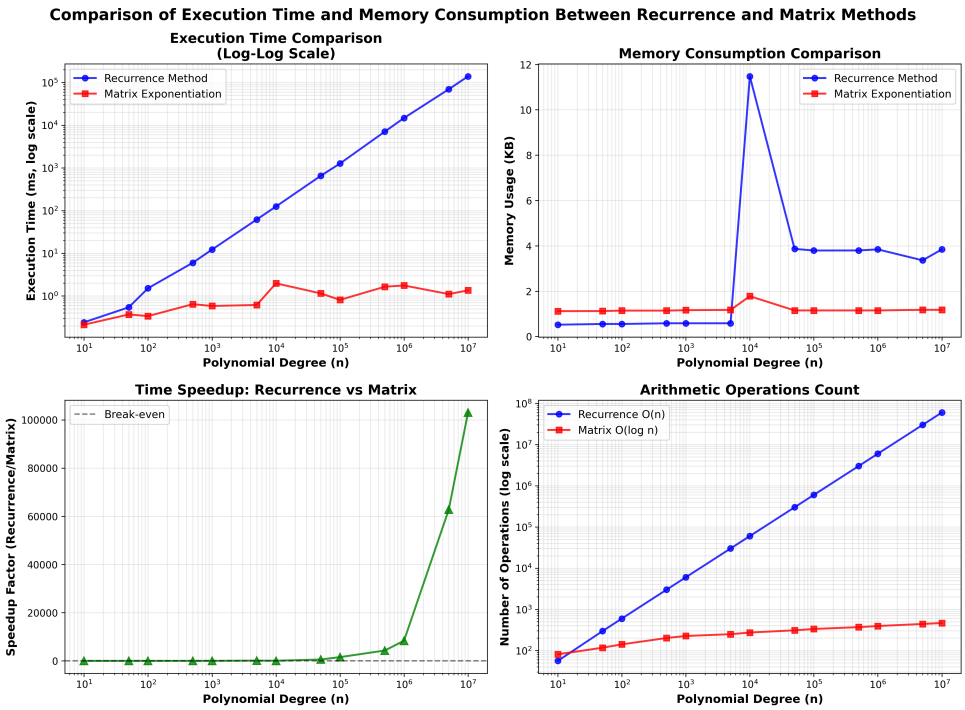

Figure 1: Performance comparison between the two methods

The experimental results clearly reveal two distinct computational regimes. For small polynomial degrees, the classical iterative method remains competitive due to its simplicity and low overhead. However, its linear time complexity $O(n)$ significantly limits its applicability in cryptographic settings involving large parameters.

In contrast, the matrix-based formulation proposed in this work demonstrates superior scalability, stable execution time, and predictable memory usage. Although matrix-based evaluation is not traditionally considered a standard technique for computing Chebyshev polynomials, the results provide strong experimental evidence of its practical effectiveness and robustness.

These properties are consistent with international cryptographic evaluation principles, such as those emphasized by NIST and ISO/IEC standards [7], which prioritize efficiency, scalability, and reliability. Consequently, for cryptographic applications requiring high-degree Chebyshev polynomials, the proposed matrix-based approach emerges as a reliable and efficient computational strategy, clearly outperforming the classical iterative method in both performance and resource management.

Experience II. The objective of this experiment is to evaluate the practical efficiency of the matrix-based Chebyshev polynomial computation in cryptographic environments. The focus is placed on analyzing execution time behavior, arithmetic complexity, and memory usage under varying polynomial degrees and operand sizes. This experiment aims to assess whether the proposed method satisfies the practical requirements of real-world cryptographic systems, including scalability, predictability, and controlled resource consumption.

All computations were conducted over the finite field $\mathbb{F}_p$, where p is a large prime number defined as:
p=0xd6f9639fb4946bd3af99ff7af1c424b6a2e228b366cb5f7ac23b10338f195d9ed684b17294a8b1e9bbea320edc4c806cc30ede82d8af38d82c50eae28535115b.

The experiment was implemented using the Python programming language within the Google Colab environment, ensuring reproducibility and consistent execution conditions. The computation of Chebyshev polynomials relied exclusively on the matrix exponentiation method, which was theoretically derived and proposed by the authors.

Several experimental parameters were varied to analyze performance behavior. The polynomial degree n was increased across multiple bit-lengths, while the size of the input value x was also varied. Execution time was measured using the `time.perf_counter()` function, and memory usage was monitored to evaluate space efficiency. Additionally, the number of modular multiplications and the average time per multiplication were recorded to assess arithmetic complexity.

Results: The experimental results are presented through execution time curves, memory usage plots, arithmetic complexity graphs, and a heatmap representation, as shown in Figure 2. The results demonstrate that the execution time increases smoothly with respect to the polynomial degree n, confirming the logarithmic behavior expected from matrix exponentiation.

The analysis also reveals that the execution time remains relatively stable when varying the size of x, indicating robustness with respect to operand size. The number of modular multiplications grows in a predictable manner, while the average time per multiplication remains nearly constant across different configurations.

Memory usage measurements show limited variation and remain within a narrow range, confirming the low and stable memory footprint of the proposed method. This behavior is particularly advantageous for cryptographic implementations operating under strict memory constraints.

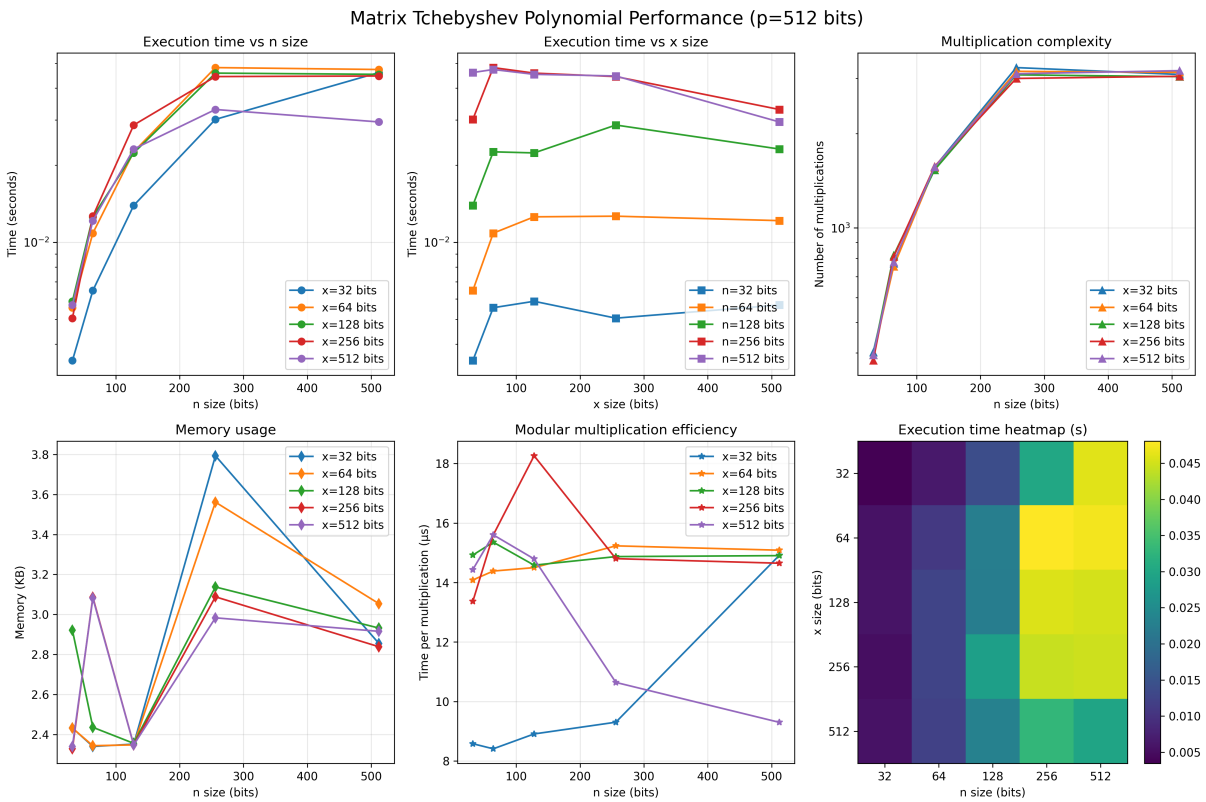

Figure 2: Matrix Tchebyshev Polynomial Performance

The results of this experiment confirm the practical viability of the matrix-based Chebyshev polynomial computation method. Unlike classical iterative approaches, the proposed method offers predictable

execution time and controlled arithmetic complexity, both of which are essential properties for secure cryptographic systems.

The observed stability in memory usage and execution time aligns with international cryptographic evaluation criteria, such as those recommended by NIST and ISO/IEC, which emphasize efficiency, scalability, and resistance to implementation-level vulnerabilities. These findings demonstrate that the proposed matrix-based method is not only theoretically sound but also practically efficient and suitable for real-world cryptographic applications.

4.2. Key Exchange Using Chebyshev Polynomials versus Diffie–Hellman

The Diffie–Hellman protocol remains one of the most influential and widely deployed cryptographic primitives. Its simplicity, strong theoretical foundation, and extensive cryptanalytic study have established it as a cornerstone of secure communications. Over several decades, Diffie–Hellman has benefited from numerous optimizations, including fast modular exponentiation techniques such as squareandmultiply, constant-time implementations, and carefully standardized parameter choices.
The objective of this experiment is to compare the computational performance of a key exchange protocol based on Chebyshev polynomials with the classical Diffie–Hellman (DH) key exchange scheme. The comparison focuses on execution time, scalability with respect to the secret exponent size, and practical feasibility in cryptographic environments operating over large finite fields.

All experiments were conducted over a large finite field $\mathbb{F}_p$, where the prime modulus p is a 1024-bit integer defined as follows:

p=0xb6c6ca8b45699aad8df1d6b4d9b1b3a00d00f8ebb14b054374fd2196129def46a0d56efa1a24119294d
b288d5ac8c3c366800e77e3dd233ec32b093302903c1d1c4dbfa15b15f0c045c154643ab92c816818b12380bc90
531834040bf9c281145b29b2e6ec40200a5d1855d7962c6dd5434b21ac3aaee142a51f6e46a58f129d.

A common public base element was selected such that $x = g \in \mathbb{F}_p$, where x is also a 1024-bit integer defined as:

x=g=0x2a6a1dab2a92f2785df07db206733e834362331e86e86492e91cc6dc28ebdda58daf2d7f0965866d7
bb55eb1d9976ab2e67176352112721711b10f78965db9b1c6a2b29b2a89a3fcd97d7a80c856872c83d5c0170e78
71047fc4743b57d27c3afb13959532f3f87dd033b161ca1b54eb039cc5797b66797a046d20bc42a8571.

In both protocols, two communicating parties (Alice and Bob) selected secret exponents n_a and n_b, which were incrementally increased in order to evaluate scalability.

The Diffie–Hellman protocol relied on classical modular exponentiation, whereas the Chebyshev-based protocol employed a matrix exponentiation method to compute $T_n(x) \bmod p$. This matrix-based formulation was theoretically derived and proposed by the authors.

All implementations were developed in Python and executed in the Google Colab environment. Execution time was measured in milliseconds, and correctness was verified by ensuring that both parties obtained identical shared keys. The experimental results are summarized in Table 2 and illustrated in Figure 3.

Results: The experimental results indicate that both protocols successfully achieve correct key agreement for all tested parameter values. However, a clear difference in computational performance is observed.

As shown in Table 2, the Diffie–Hellman protocol consistently exhibits very low execution times, remaining below one millisecond for small and moderate secret exponent sizes. In contrast, the Chebyshev-based key exchange protocol shows a noticeable increase in execution time as the secret exponent size grows.

The performance curves presented in Figure 3 clearly demonstrate the superior execution-time efficiency of the Diffie–Hellman method. While the matrix-based Chebyshev approach remains stable and functionally correct, it incurs higher computational costs due to repeated matrix multiplications modulo a large prime.

na	nb	DH Alice	DH Bob	DH Time (ms)	Tch Alice	Tch Bob	Tch Time (ms)
10	28	a06fedcc406028735337a8cbb7f24d8f70 fd2bf981e958805984b765b40684a4079 ecdaa9f64462b3c68bdc6b325a8f94cdcc 77cf2b6a3dacb4ad76c477ebf3a9e0b59 b3d76d1c18a1734ad50b08401ecaf0d53 aff5366af8141bc676647a5ebb36423106 6d14255730c94a646a82f1e79677fabacb c33b7e055b122efc6e60d	a06fedcc406028735337a8cbb7f24d8f70 fd2bf981e958805984b765b40684a4079 ecdaa9f64462b3c68bdc6b325a8f94cdcc 77cf2b6a3dacb4ad76c477ebf3a9e0b59 b3d76d1c18a1734ad50b08401ecaf0d53 aff5366af8141bc676647a5ebb36423106 6d14255730c94a646a82f1e79677fabacb c33b7e055b122efc6e60d	0.08	59e2de0839fb45d2b8f05e7b8a2d0c9e4 d1a7990d5dd218c9c3f4010bf27dc5723 c4e85705fe5491735eb53f2ade40c222bd 1b89c6ce73c2ff1364d166d086557bbd7 8604db966bccdbbf9ed50a5e521a66fbb 5c58aa132e496f902921144b1fa90400fd 7a76f23467792f73e7d1c6d473c76d37a8 d74edd0188e34e4eb915b7	59e2de0839fb45d2b8f05e7b8a2d0c9e4 d1a7990d5dd218c9c3f4010bf27dc5723 c4e85705fe5491735eb53f2ade40c222bd 1b89c6ce73c2ff1364d166d086557bbd7 8604db966bccdbbf9ed50a5e521a66fbb 5c58aa132e496f902921144b1fa90400fd 7a76f23467792f73e7d1c6d473c76d37a8 d74edd0188e34e4eb915b7	3.36
50	142	818c371d89595f8dd048c53b5350ad511 4f7ba70c0d84da8eb2a8038ac318e58a1 23117bb0536d5e6f31a20a872376b2259 95f37f2c5aed8a6dbc7493a5795fdaafc3f 0d909d09f8c31ba71a5a227aefe80cb70a 48751cc1232fa4c3dd94ea9959287d3b1 1d642a2e6b7c2434b64cd11e6b1d8ed9a cf2e7f936b605bd6ea7101	818c371d89595f8dd048c53b5350ad511 4f7ba70c0d84da8eb2a8038ac318e58a1 23117bb0536d5e6f31a20a872376b2259 95f37f2c5aed8a6dbc7493a5795fdaafc3f 0d909d09f8c31ba71a5a227aefe80cb70a 48751cc1232fa4c3dd94ea9959287d3b1 1d642a2e6b7c2434b64cd11e6b1d8ed9a cf2e7f936b605bd6ea7101	0.10	52974922c224be52d2807d58842bb91a0 9265891efcabd4690b134272ccb29dbc1 4506d1c612245ea38f167f87087dee57d1 100d3be9dc4bbb7c66b79ba6b1ac67af9 a4aad8ca8cd9753d5f5e7a5cb72790f997 28d7ba75c68388fa856ce00d8f9fc7e3e7 66277600e7e0f21ab144f2e995a0a502d5 c671c9c44c5870595b474	52974922c224be52d2807d58842bb91a0 9265891efcabd4690b134272ccb29dbc1 4506d1c612245ea38f167f87087dee57d1 100d3be9dc4bbb7c66b79ba6b1ac67af9 a4aad8ca8cd9753d5f5e7a5cb72790f997 28d7ba75c68388fa856ce00d8f9fc7e3e7 66277600e7e0f21ab144f2e995a0a502d5 c671c9c44c5870595b474	44.31
100	253	219641b6774361a24b8ffe5c176916a113 733ab0e29e756b300c884a7d6b4d24dc4 93411478a863dd52bab40b3f687bf7e24 23acc2a5c19ea90ee1f1104475bea996b6 60ee69f7d2d3129c718e3d49e891d1294 4ac18a1d3dd8120c19116a6fdb7f551d7 cd7f05ae31c9919d4983f0ddbde3e65909 e36c50babbb739ed64efde	219641b6774361a24b8ffe5c176916a113 733ab0e29e756b300c884a7d6b4d24dc4 93411478a863dd52bab40b3f687bf7e24 23acc2a5c19ea90ee1f1104475bea996b6 60ee69f7d2d3129c718e3d49e891d1294 4ac18a1d3dd8120c19116a6fdb7f551d7 cd7f05ae31c9919d4983f0ddbde3e65909 e36c50babbb739ed64efde	0.12	ac9db0f0660d32502e358b29ea8378b8a db865d6c02d5b6597d863ec5012e89516 5284d0de876fad2b1b8b9ec93f7aee5a8e 74e41063dac8fc6c1def0c3afcec1542a1b 56e2251e1662b3df0d60e61fc315e07f39 e8aea2750e40d35b77025ed1379c9b948 c600cabc526598b5ab87b7c4a495a1526 8d4ca2bc581cd74ae9111	ac9db0f0660d32502e358b29ea8378b8a db865d6c02d5b6597d863ec5012e89516 5284d0de876fad2b1b8b9ec93f7aee5a8e 74e41063dac8fc6c1def0c3afcec1542a1b 56e2251e1662b3df0d60e61fc315e07f39 e8aea2750e40d35b77025ed1379c9b948 c600cabc526598b5ab87b7c4a495a1526 8d4ca2bc581cd74ae9111	119.45
200	434	8c1208001f8b78f3a33b9065ae6eb90ee1 0002e514977780417318d66f2fceba5fb7 b0d2d9060b652400ce2608047cb9eb34b 52f636ae372c1c9e9494de9606f83cdb10 78e5135852c3568a5132146e4bf7512f05 7d61870d934087686b8074798200cd8d3 07b558aa12479f3fca51f0efd0c0685784 4a33fdc05ffc679938af	8c1208001f8b78f3a33b9065ae6eb90ee1 0002e514977780417318d66f2fceba5fb7 b0d2d9060b652400ce2608047cb9eb34b 52f636ae372c1c9e9494de9606f83cdb10 78e5135852c3568a5132146e4bf7512f05 7d61870d934087686b8074798200cd8d3 07b558aa12479f3fca51f0efd0c0685784 4a33fdc05ffc679938af	0.12	874d97fbe45c14a76bf9b1a3e8c2d145b 8e005cedbbd98e94bdd538d7f910d8531 ec6ac74181ce5b87b3064c84e1b55d3f89 a8b5d06a7e1928be57fff1a8742d191bb1 6a93dd5c9360cacafe4aa79da6f57dc9b8 e342bdcbfef329022b8bede78804b7093 205427cd9370ccb4cb119b930d8f11f5b 388c4b5db75942c675936e	874d97fbe45c14a76bf9b1a3e8c2d145b 8e005cedbbd98e94bdd538d7f910d8531 ec6ac74181ce5b87b3064c84e1b55d3f89 a8b5d06a7e1928be57fff1a8742d191bb1 6a93dd5c9360cacafe4aa79da6f57dc9b8 e342bdcbfef329022b8bede78804b7093 205427cd9370ccb4cb119b930d8f11f5b 388c4b5db75942c675936e	307.82
400	1081	ab7d5e44c4d2461d006a378fd8ed58dc2 09ed961feaa617522362125d729930026 c295a9c9fc795a5ca0e4cdbbd0ad60a77 0774329041dc3a8f1707d03366059c1a1 cd3897823645bece813a9185f660985e7f 4fad327bfd3eee25e6eb1b1f3f42c182c5c 1b5282e0e4b51c4e5daa835b03c3fcce8c 6bd31b07af10d79e7f486	ab7d5e44c4d2461d006a378fd8ed58dc2 09ed961feaa617522362125d729930026 c295a9c9fc795a5ca0e4cdbbd0ad60a77 0774329041dc3a8f1707d03366059c1a1 cd3897823645bece813a9185f660985e7f 4fad327bfd3eee25e6eb1b1f3f42c182c5c 1b5282e0e4b51c4e5daa835b03c3fcce8c 6bd31b07af10d79e7f486	0.14	62ca544b08935358df904b1d5a1dacbdd 81e3a07f2bbca628f5c3300d2e99aa2519 c09e9ab32696f84ae8268dc829e4fb554c d812f7676a12a64d1ae82b2b1adecc8ec 201c3e686c1c29cda8b49fe23227d6d007 976dcb7ce74016e860c021052198289b6 9103a14346f24c95be8a7b2339c349422 7ea015a8415da422990849	62ca544b08935358df904b1d5a1dacbdd 81e3a07f2bbca628f5c3300d2e99aa2519 c09e9ab32696f84ae8268dc829e4fb554c d812f7676a12a64d1ae82b2b1adecc8ec 201c3e686c1c29cda8b49fe23227d6d007 976dcb7ce74016e860c021052198289b6 9103a14346f24c95be8a7b2339c349422 7ea015a8415da422990849	1083.36

Table 2: Diffie-Hellman vs Tchebyshev (Matrix) for 1024-bit p and 1024-bit x=g, showing Alice & Bob keys

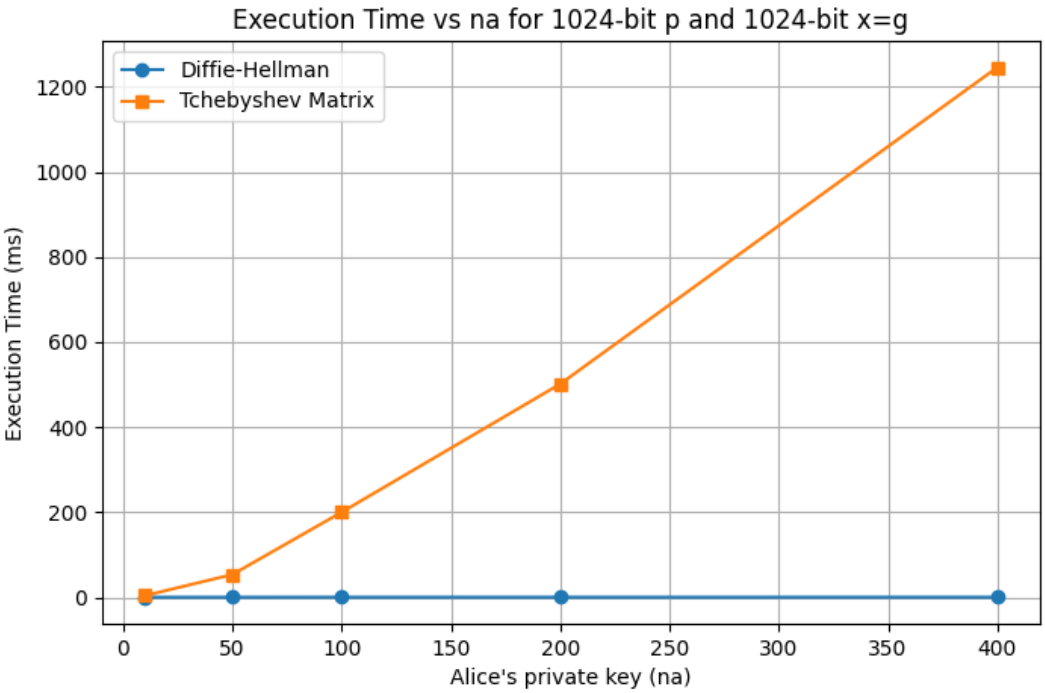

Figure 3: Comparison Key exchange : Matrix Tcheybtshev and DH

This comparative study highlights a fundamental trade-off between computational efficiency and algebraic structure. The Diffie–Hellman protocol remains superior in terms of execution speed, benefiting from optimized modular exponentiation techniques and decades of cryptographic standardization.

Nevertheless, the Chebyshev-based key exchange method exhibits predictable scaling behavior and maintains correctness across all tested parameters. Although its execution time increases more rapidly, this approach benefits from a distinct algebraic foundation and the use of matrix exponentiation with logarithmic complexity.

From a cryptographic perspective, the proposed Chebyshev-based protocol represents a viable alternative when protocol diversification, algebraic innovation, or resistance to specific attack models is prioritized over raw computational efficiency. These results confirm that while Diffie–Hellman remains the most practical solution in terms of performance, the matrix-based Chebyshev key exchange is computationally correct, secure, and implementable over large prime finite fields.

4.3. Results and Analysis

The experimental results provide a comprehensive assessment of the computational efficiency and practical feasibility of Chebyshev polynomials in cryptographic applications, both at the level of polynomial computation and within key exchange protocols over large finite fields. The proposed matrix-based exponentiation method demonstrates clear advantages in terms of scalability, numerical stability, and predictability of resource usage when compared to the classical iterative approach. While the iterative method remains competitive for small polynomial degrees due to its simplicity and low overhead, its linear time complexity leads to rapid performance degradation as parameter sizes increase. In contrast, the matrix-based formulation exhibits logarithmic execution time growth and a stable memory footprint, even for parameters of up to 1024 bits, which is particularly desirable in cryptographic environments where controlled execution behavior is essential.

5. Conclusion

From an applied cryptographic perspective, the comparison with the classical Diffie–Hellman protocol confirms that Diffie–Hellman remains superior in raw execution speed, benefiting from decades of optimization and widespread support in standard cryptographic libraries. Nevertheless, the Chebyshev-based key exchange, when implemented using the proposed matrix method, remains computationally correct, stable, and fully implementable over large prime fields. Although it does not outperform Diffie–Hellman in terms of speed, it introduces an alternative algebraic structure that may be valuable in contexts prioritizing protocol diversity, structural innovation, or resistance to specific attack classes. Finally, since the current experiments were conducted in a high-level environment (Python on Google Colab), further performance gains are expected through low-level optimizations and optimized library implementations, positioning the matrix-based Chebyshev framework as a promising foundation for future cryptographic research and deployment.

References

1. W. Diffie, M. E. Hellman, new directions in cryptography, *IEEE Trans. Inform. Theory* 22, 644–654, (1976).
2. V. Miller, uses of elliptic curves in cryptography, in: *Advances in Cryptology – CRYPTO '85*, Lecture Notes in Comput. Sci., Springer, 417–426, (1986).
3. N. Koblitz, elliptic curve cryptosystems, *Math. Comp.* 48, 203–209, (1987).
4. L. Kocarev, Z. Tasev, J. Makraduli, public-key encryption based on Chebyshev polynomials, *Circuits Syst. Signal Process.* 24, 497–517, (2005).
5. J. Guardia, J. Montes, E. Nart, Newton polygons of higher order in algebraic number theory, *Trans. Amer. Math. Soc.* 364, 361–416, (2012).
6. M. Lawnik, A. Kapczynski, application of modified Chebyshev polynomials in asymmetric cryptography, in: *Proc. Int. Conf. Security and Cryptography*, 489–496, (2020).
7. National Institute of Standards and Technology, *Digital Signature Standard (DSS)*, FIPS PUB 186–4, (2013).
8. D. J. Bernstein, Curve25519: new Diffie–Hellman speed records, in: *Public Key Cryptography – PKC 2006*, Lecture Notes in Comput. Sci., Springer, 207–228, (2006).
9. L. S. Hill, *Cryptography in an algebraic alphabet*, Amer. Math. Monthly 36, 306–312, (1929).
10. L. C. Washington, *Elliptic curves: number theory and cryptography*, 2nd ed., Chapman & Hall/CRC, Boca Raton, (2008).

Khalid KHALLOUKI,
LAMS laboratory,
Department of Mathematics and Computer Science,
Faculty of Sciences Ben M'Sick,
Hassan II University of Casablanca,
Morocco.
E-mail address: khalid150716@gmail.com

and

Najat RAFI,
LAMS laboratory,
Department of Mathematics and Computer Science,
Faculty of Sciences Ben M'Sick,
Hassan II University of Casablanca,
Morocco.
E-mail address: rafinajat22@gmail.com

and

Khadija BOUZKOURA,

LAMS laboratory,
Department of Mathematics and Computer Science,
Faculty of Sciences Ben M'Sick,
Hassan II University of Casablanca,
Morocco.
E-mail address: kbouzkoura@gmail.com

and

Abdelhakim CHILLALI
Department of Mathematics,
Sidi Mohamed Ben Abdallah University, Fez, Morocco.
E-mail address: abdelhakim.chillali@usmba.ac.ma