

Performance Overhead of SM2 vs. RSA in Spring Boot

Yangbin Zou *

School of Software, Xinjiang University, Software Engineering Urumqi, Urumqi, Xinjiang, China

* Corresponding author Email: 20232501220@stu.xju.edu.cn

Abstract: As the national strategy for autonomous and controllable information security continues to advance, the process of replacing imported technologies with domestic alternatives in the field of critical information infrastructure is gradually accelerating. Traditional RSA algorithms incur significant computational overhead at high security levels (such as 2048 bits), making them prone to becoming performance bottlenecks in high-concurrency web scenarios. In contrast, the Chinese SM2 algorithm, based on the elliptic curve discrete logarithm problem, requires a key length that is only one-eighth that of RSA at equivalent security levels, offering higher theoretical efficiency. However, existing research has largely focused on comparing the mathematical principles of the algorithms themselves, with a lack of quantitative empirical data from real-world industrial-grade web development frameworks. In particular, the impact of the Spring Boot framework's core mechanisms (such as auto-configuration, bean management and request interceptors) on algorithm execution efficiency has not yet been systematically studied. Consequently, this study utilises the Spring Boot framework and employs a method involving control variables and stratified experiments to quantitatively compare the performance of SM2 and RSA in secure communication scenarios. The aim is to supplement the theoretical data on domestic cryptographic algorithms in the field of web development, thereby providing developers with quantifiable reference criteria and practical engineering guidance for selecting and applying these algorithms in real-world projects.

Keywords: National Cryptographic Algorithms; SM2; RSA; Spring Boot; Performance Comparison.

1. Introduction

As the cornerstone of Public Key Infrastructure (PKI), asymmetric encryption algorithms play an indispensable role in secure communication scenarios such as digital signatures, identity authentication and key agreement. With the rapid development of internet technology and the deepening of digital transformation, web applications have become the core medium for information transmission and business processing; consequently, the performance of the asymmetric encryption algorithms employed directly impacts the security and responsiveness of web services [1]. Therefore, both academia and industry have devoted significant attention to the performance of asymmetric encryption algorithms in real-world engineering environments. Among the many asymmetric encryption algorithms, RSA has long dominated the field of public-key cryptography thanks to its clear mathematical principles and robust engineering support, and is widely used in security protocols such as SSL/TLS and PGP. However, the ever-increasing computational power of information technology has necessitated that RSA must increase key lengths to withstand attacks. As RSA's security relies on the difficulty of factoring large integers, increasing key lengths leads to a sharp rise in the computational load of modular exponentiation, resulting in slower encryption and decryption speeds [2]. Consequently, the use of the RSA algorithm in high-concurrency web application scenarios is highly likely to become a bottleneck for system performance. At the same time, the demand for low-latency, high-throughput services in mobile internet and cloud computing environments has further exacerbated the performance shortcomings of the RSA algorithm.

To address the aforementioned challenges and achieve self-reliance and control over cryptographic technology, China's State Cryptography Administration has released the SM series of commercial cryptographic algorithms. As a public-

key cryptographic algorithm based on the elliptic curve discrete logarithm problem (ECDLP), SM2 relies on a problem whose computational complexity is far greater than that of integer factorisation; consequently, SM2 requires a significantly shorter key length to achieve the same level of security—a 256-bit SM2 key can provide security strength equivalent to that of a 2048-bit RSA key, with a key length of only one-eighth of the latter [3]. Theoretically, a reduction in key length implies lower storage overhead, reduced computational complexity and higher encryption and decryption efficiency. Consequently, based on this theoretical advantage, the SM2 algorithm has attracted increasing attention in the process of domestic substitution within critical information infrastructure.

However, despite decades of in-depth academic research comparing the performance of elliptic curve cryptography (ECC) and RSA, and a general theoretical consensus that 'ECC is more efficient at the same level of security', existing research still suffers from the following significant shortcomings:

(1) Research has largely focused on the mathematical efficiency of the algorithms themselves or on benchmarking individual algorithms, with a lack of end-to-end performance evaluations of SM2 in real-world web development environments.

(2) Although some community-developed Spring Boot Starters support SM2 integration, a standardised integration solution comparable to that of Spring Security for RSA has yet to emerge; as a result, developers still face the dilemma of 'theoretically feasible but difficult to implement' in real-world projects.

In response to the above issues, this paper uses the Spring Boot framework as its experimental platform and compares the Chinese national cryptographic standard SM2 algorithm with the internationally recognised RSA algorithm, focusing on the following three core issues:

(1) When framework interference is ruled out, how significant is the difference in pure algorithmic performance between SM2 and RSA in basic operations such as symmetric and asymmetric encryption and decryption, and digital signature verification?

(2) Once the algorithm has been integrated into the entire request chain of the Spring Boot API (Filter → Interceptor → ArgumentResolver → Controller), what impact will this have on the framework's performance?

(3) In production-level scenarios characterised by high concurrency and long-running connections, how significant are the differences between the two algorithms in terms of throughput (TPS), response time (RT) and system resource usage (CPU/memory)?

The main contributions of this paper are as follows.

(1) This paper adopts a research methodology combining 'control variables, stratified experiments and quantitative analysis', and designs three progressive experimental scenarios: pure algorithm benchmarking, full-link end-to-end API testing, and high-concurrency long-connection stress testing. Compared to previous studies, this stratified testing approach enables more precise identification of the sources of performance bottlenecks, distinguishing between losses inherent to the algorithm itself and those caused by framework interference.

(2) By embedding timestamps in the core components of the Spring Boot request chain, this paper calculates the execution times of SM2 and RSA at each stage, quantifying the specific impact of core Spring Boot mechanisms—such as auto-configuration, bean management and interceptors—on the execution efficiency of asymmetric encryption algorithms. Building on this, the paper explores the potential for improving algorithm performance through framework configuration optimisation by adjusting various framework configuration parameters and their corresponding performance metrics.

(3) This paper proposes a reproducible research methodology for comparing the performance of National Cryptographic Algorithms, producing comprehensive performance benchmark data for SM2 and RSA within the Spring Boot framework. This provides an engineering reference for the standardised integration of National Cryptographic Algorithms within microservice architectures.

2. RELATED WORK

2.1. Comparison of Mathematical Algorithm

The security and efficiency of asymmetric encryption algorithms essentially depend on the computational complexity of the underlying mathematical problems. Both RSA and SM2 are public-key cryptosystems, with their mathematical foundations being the integer factorisation problem and the elliptic curve point set operations, respectively.

2.1.1. The RSA Algorithm and Integer Factorisation Problems (IFP)

The security of the RSA algorithm is based on the computational difficulty of factoring a composite number n into two large prime numbers p and q (the Integer Factorisation Problem, IFP). Its core mathematical logic is as follows:

(1) Key generation: The algorithm first selects two large prime numbers p and q , calculates their product $n = p \times q$, and computes the Euler's totient function $\phi(n) = \phi(p \times$

$q) = (p - 1) \times (q - 1)$. It then selects a public key exponent e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$; that is, e is coprime with $\phi(n)$. This is a necessary condition for the existence of the private key [4].

(2) Algorithm: The private key d is calculated using the congruence equation $e \times d \equiv 1 \pmod{\phi(n)}$. The encryption and decryption processes involve modular exponentiation with large numbers: during encryption, $C \equiv M^e \pmod{n}$; during decryption, $M \equiv C^e \pmod{n}$ [4].

(3) Performance bottleneck: The computational complexity of modular exponentiation increases at an $O(k^3)$ rate as the key length increases. To withstand increasingly powerful computational attacks, RSA must use longer keys, which leads to a significant rise in CPU resource consumption when handling high-concurrency web requests.

2.1.2. The SM2 Algorithm and Elliptic Curves Derived from the Discrete Logarithm Problem (ECDLP)

SM2 is an elliptic curve cryptography (ECC) algorithm based on the prime field $\mathbb{F}_p$, the security of which is founded on the elliptic curve discrete logarithm problem (ECDLP).

(1) Mathematical description: SM2 uses a specific elliptic curve equation, $y^2 = x^3 + ax + b \pmod{p}$ [5]. Given a base point G and a scalar k , it is mathematically straightforward to compute the point $Q = kG$; however, given Q and G , it is computationally infeasible to determine the scalar k [6].

(2) Efficiency advantages: Compared to the IFP problem on which RSA relies, the ECDLP offers greater strength per bit. This means that SM2 requires only a 256-bit key length to provide security equivalent to that of RSA with a 2048-bit key. The reduction in key length, in turn, directly reduces computational effort, storage requirements and network bandwidth usage.

2.1.3. Theoretical Comparison of Key Performance Parameters

A comparison of the theoretical performance parameters of the RSA and SM2 algorithms is shown in **Table 1**:

2.2. Analysis of the Performance Impact of the Spring Boot Framework Mechanism

In industrial-grade web development, the performance of cryptographic algorithms is also affected by additional overheads such as the host framework and component call chains. This section will focus on analysing the marginal performance costs incurred by encryption and decryption operations at each abstraction layer of the Spring Boot framework, and will propose corresponding experimental designs and validation approaches.

2.2.1. Overhead Associated with AOP Proxies and Interceptor Mechanisms

Spring Boot achieves transparent encryption and decryption of API requests through Aspect-Oriented Programming (AOP) or filters to decouple business logic; however, this also introduces additional execution overhead:

(1) Call chain latency: In high-concurrency scenarios, the time taken by the choice of proxy mode and encryption operations within the overall chain may result in a significant cumulative effect of the overhead from dynamic proxies and the time required for encryption calculations.

(2) Context switching: Before processing the encryption logic, the operations performed by interceptors to read and transform the HTTP request context (such as the SecurityContext) also increase CPU utilisation on non-core

computational tasks.

Table 1. Theoretical Performance Comparison of the RSA and SM2 Algorithms

Performance Dimension	RSA Algorithm	SM2 Algorithm
Underlying Mathematical Problem	IFP	ECDLP
Key Parameter Constraints	$\gcd(e, \phi(n)) = 1$	Curve parameters a and b must comply with the National Cryptographic Standard
Standard Key Length	2048 bits	256 bits
Core Operation Type	Modular exponentiation	Elliptic curve dot product
Security/Efficiency Ratio	Low	Extremely high

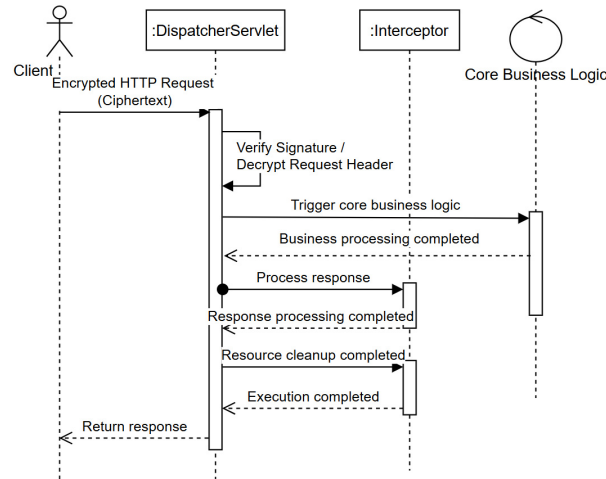


Figure 1. SM2/RSA Request Processing and Interception Flowchart

Based on the Spring Boot framework, the execution flow of SM2 and RSA algorithm-based APIs within the request chain is shown in **Figure 1**. Requests sequentially pass through Filters, Interceptors, and ArgumentResolver for parameter parsing, before finally reaching the Controller to

complete business processing.

The call relationship between the encryption/decryption service and the AOP proxy is shown in **Figure 2**. Whilst the framework weaves in cross-cutting concerns via dynamic proxies, it also introduces additional call overhead.

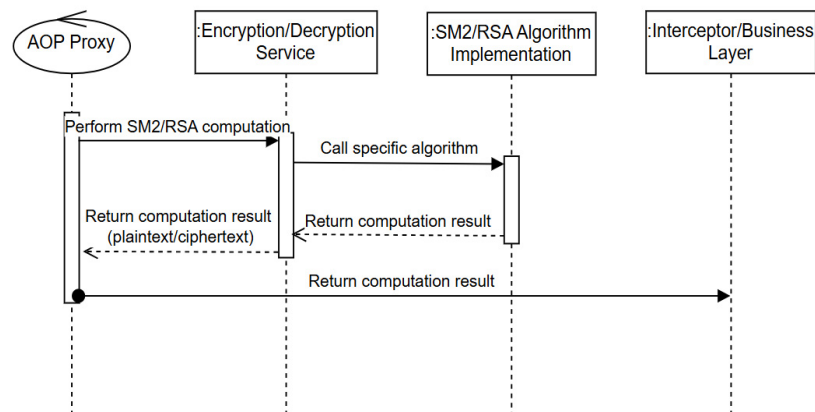


Figure 2. Architectural Integration of Spring AOP Proxying with Cryptographic Services.

2.2.2. Data serialisation and encoding loss

In web services based on the Spring Boot framework, the Jackson library and deserialisation are typically used by default; however, the performance overhead generated by this process can sometimes exceed that of the algorithm itself.

(1) Reflection overhead: When Jackson converts a POJO into the byte array required for encryption, it performs CPU-intensive operations such as using reflection to retrieve field information.

(2) Encoding conversion: To facilitate transmission over the HTTP protocol, the encrypted ciphertext usually needs to be encoded using Base64 or Hex. These additional conversion steps are also a significant cause of end-to-end latency.

2.3. Literature Review of Domestic and International Research Status

Currently, there are differences in focus between the academic and industrial communities, both domestically and internationally, regarding research into the integration of the national cryptographic algorithm (SM2) and the internationally recognised algorithm (RSA):

2.3.1. Domestic Integration Status

(1) Component dependency paths: Domestic Java web developers primarily rely on the Bouncy Castle (BC) library and its derivative open-source projects (such as Hutool-crypto) to implement National Cryptographic Algorithms. Currently, the BC library supports algorithms such as SM2, SM3 and

SM4, and forms the core of national cryptographic integration within the domestic Java environment.

(2) Framework integration bottlenecks: Although some unofficial Spring Boot Starter components have emerged within the community, no official standard framework has yet been established that enjoys widespread consensus and industry recognition, akin to Spring Security for RSA. At present, most integration solutions still lack authoritative, standardised, industrial-grade integration specifications.

(3) Research data gaps: Existing research largely focuses on the mathematical security, cryptographic strength and purely theoretical performance comparisons of the algorithms themselves. There is a lack of systematic, quantified performance benchmark data in end-to-end integration scenarios within mainstream web frameworks such as Spring Boot.

2.3.2. Integration Integration Status

(1) Security assessment: Research into Chinese cryptographic algorithms within the international academic community has primarily focused on mathematical principles and algorithmic security. By comparing the ECDLP-based SM2 with the IFP-based RSA, the reliability and resistance to attacks at the mathematical level are verified.

(2) Limitations of Generalised Integration: There is relatively little research abroad on the engineering integration of Chinese cryptographic algorithms. This paper attributes this to differing demand-driven priorities—foreign researchers tend to focus more on the deep integration of their own national or regional standard algorithms (such as Russia’s GOST or NIST PQC) within similar frameworks.

3. METHODOLOGY

This study quantifies the performance differences between the SM2 and RSA algorithms within the Spring Boot framework by designing a tiered experimental system. This chapter provides a detailed overview of the overall experimental architecture, the hardware and software environment configurations, and the simulation scenarios designed for different application dimensions.

3.1. General Framework of the Experiment

This study employs a ‘control variable method’ evaluation framework to assess factors affecting algorithm performance, primarily through the following two key dimensions:

(1) By eliminating interference from web containers, network communication and framework components, we test the algorithm’s pure computational efficiency within the JVM process.

(2) By placing the algorithm within a full-stack Spring Boot request, we analyse the ‘marginal cost’ imposed on algorithm execution by AOP interception, bean management and data serialisation.

3.2. Experimental Environment Configuration

The following hardware and software configurations were used in this study:

(1) Hardware specifications:

(i) Operating system name: Microsoft Windows 11 Home Chinese Edition

(ii) Version: 10.0.26200, build 26200

(iii) System model: HP Pavilion Plus Laptop 14-eh1xxx

(iv) Processor: 13th Gen Intel® Core™ i5-13500H

(v) Clock Speed: 2600 MHz

(vi) Number of Cores: 12 cores

(vii) Number of Threads: 16 threads

(viii) Installed Physical Memory (RAM): 16.0 GB

(2) Software stack versions:

(i) Development language: JDK 21 (latest JVM performance optimisation features).

(ii) Core framework: Spring Boot 3.2.

(iii) Testing tool: JMeter 5.0.4.

(iv) Cryptography provider: Bouncy Castle (BC) library.

3.3. Scenario Modelling and Experimental Design

This study simulates the transition from theoretical calculations to a production environment through three progressive experimental scenarios:

3.3.1. Scenario A: Benchmark Testing (In-JVM Testing)

(1) Objective: To test the theoretical performance limits of the SM2 and RSA algorithms.

(2) Experimental Method: With interference from the Spring Boot framework eliminated, the algorithm interfaces are directly invoked to perform cyclic encryption, decryption, and signature verification operations.

(3) Test Metrics: Time taken per operation (ms), operations per second (Ops/Sec).

3.3.2. Scenario B: End-to-end testing of the API interface layer

(1) Objective: To identify the primary factors and points where the Spring Boot framework mechanism impacts algorithm execution efficiency.

(2) Business Flow Simulation: Construct a complete end-to-end process covering “Login → Authorisation → Business Processing”.

(3) Experimental Method: Utilise timestamp logging technology to record the time taken at each stage—Filter, Interceptor and Controller—thereby isolating the overhead introduced by the framework.

3.3.3. Scenario C: High-concurrency production-level load testing

(1) Objective: To test throughput stability and resource utilisation in a real production environment.

(2) Implementation Method: Use JMeter to build a high-concurrency testing environment, enabling the Keep-Alive persistent connection property to simulate a real-world web interaction model.

(3) Test Metrics: Average response time (RT), transactions per second (TPS), server CPU fluctuations, and JVM heap memory reclamation frequency.

4. RESULTS AND ANALYSIS

4.1. Distribution Characteristics of Experimental Data and Verification of Frame Stability

4.1.1. Analysis of Framework Overhead Distribution

Figure 3 illustrates the statistical distribution characteristics of framework overhead (T_{frame}) for RSA and SM2 in a full-link encryption scenario.

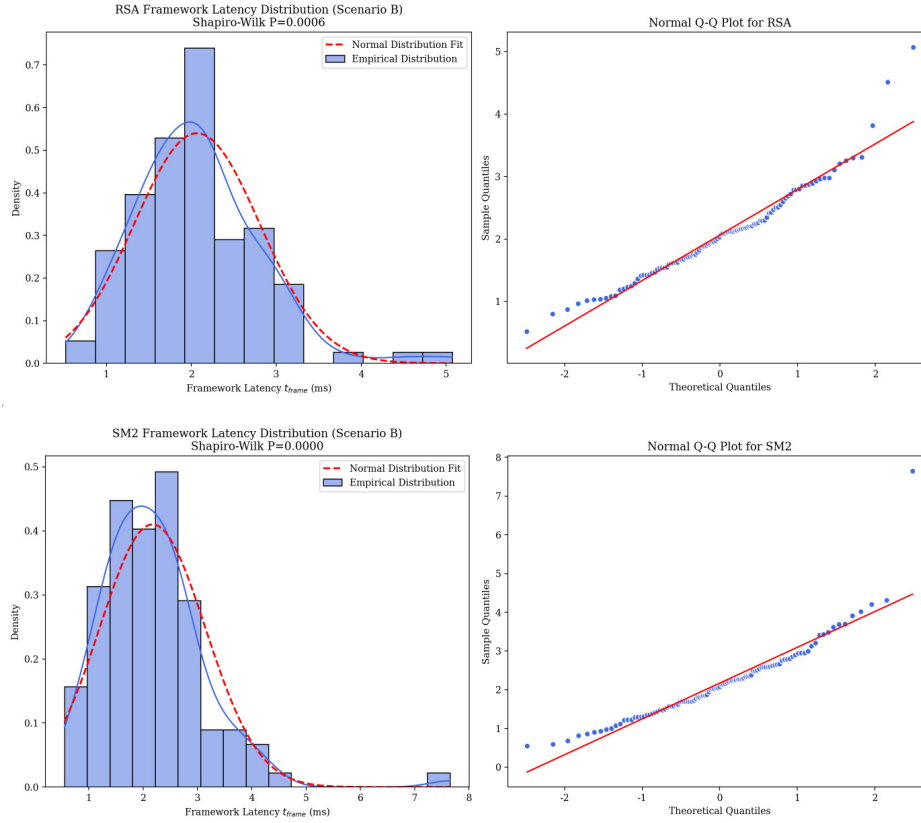


Figure 3. Histograms and Q-Q plots of framework overhead distribution for RSA and SM2

The comparison of the data reveals that T_{frame} in the RSA and SM2 scenarios exhibits a high degree of consistency in terms of statistical distribution:

(1) Consistency in central tendency: The histograms for both are highly concentrated around 2.0 ms, demonstrating that the overhead incurred by the Spring Boot framework when processing encrypted requests is highly predictable.

(2) Consistency in skewness: Although the p-values from the Shapiro-Wilk test are all less than 0.05 (see **Table 2**), in the Q-Q plot, most data points lie close to the theoretical normal line, with upward deviation occurring only at the high-end quantiles.

4.1.2. Summary of Statistical Tests

Table 2. Statistical table of pure algorithm performance experiments (N=10)

Scenario	Algorithm	Sample size (N)	Shapiro-Wilk(P)	Conclusions on the distribution
Encryption	RSA	85	0.0006	Reject the hypothesis of normality
	SM2	97	0.0000	Reject the hypothesis of normality
Decryption	RSA	74	0.0005	Reject the hypothesis of normality
	SM2	100	0.0000	Reject the hypothesis of normality
Signature & verification	RSA	78	0.0000	Reject the hypothesis of normality
	SM2	91	0.0000	Reject the hypothesis of normality

Table 2 summarises the statistical parameters of

framework overhead under encryption scenarios. The experimental data show that the response time of SM2 is slightly higher during the encryption phase, but the framework overhead characteristics of the two are largely consistent.

In summary, the framework losses of RSA and SM2 exhibit a high degree of statistical homogeneity. Consequently, the influence of environmental fluctuations on the evaluation of the algorithms can be ruled out, thereby ensuring the objectivity and validity of the analysis of performance reversals for different algorithms (decryption, signing, and signature verification) in subsequent sections.

4.2. Scenario A: Analysis of Pure Algorithm Benchmark Results

This section examines the theoretical performance ceiling of the algorithm within a JVM process, excluding the influence of the Spring Boot framework.

4.2.1. Statistics on Basic Computational Overhead

Table 3. Statistical summary of pure algorithm performance experiments (N=10)

Experimental Metrics	RSA-2048 (ms)	SM2-256 (ms)	Performance Comparison: SM2 vs RSA
Average encryption time	0.155 ± 0.012	0.413 ± 0.045	Slower (-166%)
Average decryption time	0.902 ± 0.051	0.313 ± 0.025	Significant Optimization (+65%)
Full Encryption-Decryption Pipeline	1.058 ± 0.054	0.727 ± 0.077	Overall Improvement (+31%)
Signature and Verification Time	0.890 ± 0.054	0.698 ± 0.095	Significant Optimization (+21%)

In this experiment, 10 independent replicates were conducted for each of the two algorithms, with each replicate comprising 1,000 sampling cycles, to obtain data on the time taken by the algorithms for their core computational operations, as shown in **Table 3**.

Figure 4 shows a comparison of the average processing times for RSA-2048 and SM2-256 in pure algorithmic benchmark tests. Experimental data indicate that RSA

performs slightly better in encryption operations, whilst SM2 performs significantly better than RSA in decryption operations. When considering the complete encryption and decryption process, SM2 has a total execution time of approximately 0.73 ms, representing a performance improvement of around 31% compared to RSA (1.05 ms), demonstrating a clear overall performance advantage.

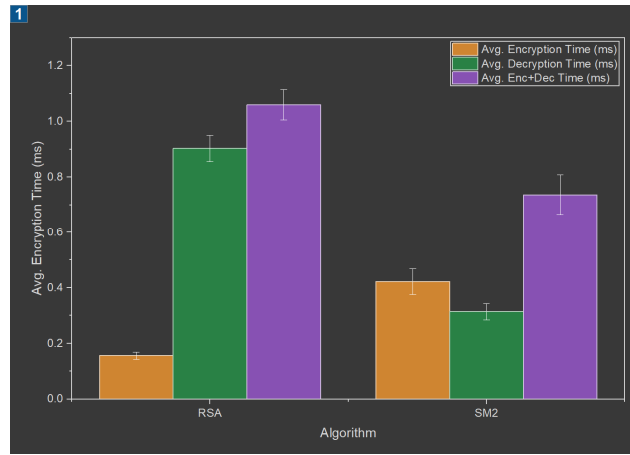


Figure 4. Bar chart comparing the average execution times of RSA and SM2 across various dimensions

4.2.2. Stability and Dispersion Analysis

Experimental data show that the coefficient of variation for the full-link performance of both RSA and SM2 is around 10%, indicating that both algorithms demonstrate good experimental stability. Under these conditions, this study

compared the total encryption and decryption times for both algorithms across 10 rounds of experiments, as shown in **Figure 5**. The results indicate that the mean time for RSA was 1.06 ms, whilst that for SM2 was 0.74 ms; SM2 performed significantly better than RSA in every round of the experiments.

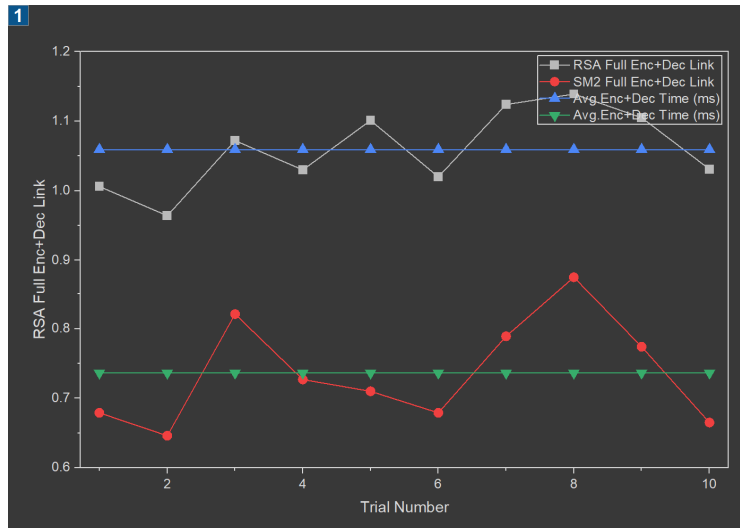


Figure 5. Line graph of latency fluctuations in the full pipeline over 10 experimental trials

4.3. Scenario B: End-to-End Performance Analysis of the API Interface Layer

Within the Spring Boot framework, algorithm execution is no longer an isolated process. This section quantifies the marginal performance cost introduced by the framework's mechanisms, based on the formula $RT_{total} = T_{algo} + T_{frame}$.

4.3.1. Experimental Design and Data Cleaning Strategies

To investigate in depth the relative contributions of low-level encryption algorithms and high-level business frameworks to system response times, this study designed a high-precision end-to-end strip-down experiment based on

nanosecond-level timestamping.

(1) Taking into account the JVM's just-in-time (JIT) compilation mechanism and class loading delays in the Spring container, the experiment was configured with 10 warm-up loops. Sampling data was recorded from the 11th request onwards to eliminate initialisation bias.

(2) During the sampling process, outliers caused by transient system scheduling, interruptions or disk I/O jitter (such as anomalous records where $RT_{total} > 10ms$) were identified and excluded.

(3) `System.nanoTime()` was utilised to obtain nanosecond-level timestamps for the entire request lifecycle. This experiment enhanced JMeter's default millisecond-level

sampling precision to 10^{-4} ms (nanosecond level) to eliminate rounding errors in RT_{total} caused by rounding.

4.3.2. Performance Metrics Calculation Model

Through an in-depth analysis of the HTTP request lifecycle, this experiment establishes the following performance decomposition model:

(1) Total round-trip time (RT_{total}): Refers to the time taken from the dispatch of the request to the receipt of the complete response.

(2) Algorithm Core Duration(T_{algo}): Refers to the pure computation time spent executing encryption/signature algorithms at the server-side Controller layer.

(3) Framework and Scheduling Overhead(T_{frame}): Defined as the residual term after excluding algorithmic computation, as per the following formula:

$$T_{frame} = RT_{total} - T_{algo}$$

This term primarily comprises AOP interception, serialisation/deserialisation, Servlet scheduling, and network transmission overhead.

4.3.3. Statistical Analysis and Comparison of Experimental Results

(1) Performance Analysis of Asymmetric Encryption and Decryption

This section presents the proportions of T_{algo} and T_{frame} for RSA-2048 and SM2-256 when performing encryption tasks.

Key Value: Statistical analysis of the performance overhead of National Cryptographic Algorithms in data protection scenarios.

Table 4. Summary Table of Categorized Encryption Performance (N=99, excluding warm-up data)

Algorithm	T_{algo} Mean(ms)	RT_{total} Mean(ms)	RT_{total} Std(ms)
RSA	0.4852	1.8432	0.3812
SM2	0.5711	2.1276	0.4281

Table 5. Summary Table of Categorized Decryption Performance (N=99, excluding warm-up data)

Algorithm	T_{algo} Mean(ms)	RT_{total} Mean(ms)	RT_{total} Std(ms)
RSA	19.1725	20.7791	1.7942
SM2	0.7395	2.5900	0.7795

(2) Performance Analysis of Digital Signature and Verification

This section presents the latency distribution of the two types of algorithms in identity authentication scenarios.

Core Value: To analyze the performance characteristics and trade-offs between RSA (characterized by slow private key signing and fast public key verification) and SM2 (characterized by more balanced computational overhead).

Table 6. Summary Table of Digital Signature and Verification Performance (N=99, excluding warm-up data)

Algorithm	T_{algo} Mean(ms)	RT_{total} Mean(ms)	RT_{total} Std(ms)
RSA	19.8642	23.1485	3.1254
SM2	1.7654	3.8412	0.9542

Based on the experimental mean values from **Table 4**, **Table 5**, and **Table 6**, the performance latency distribution for RSA and SM2 across various business scenarios was plotted, as illustrated in **Figure 6**.

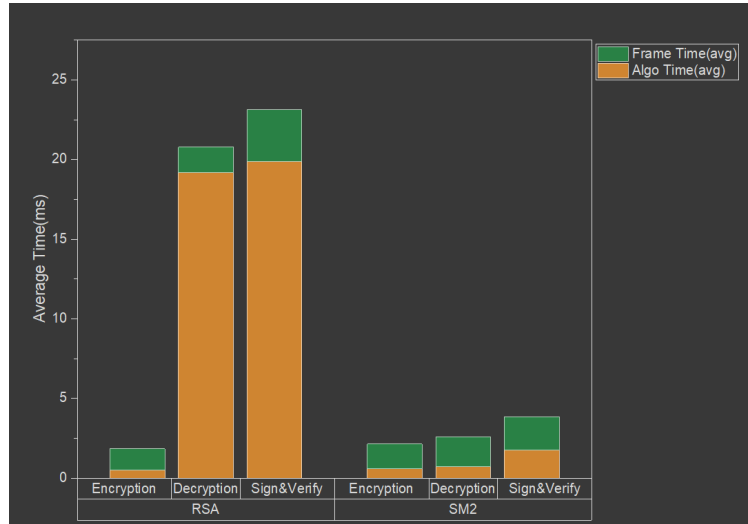


Figure 6. Comparison of core algorithm latency and framework overhead proportions between RSA and SM2

4.3.4. Core Findings and Discussion

(1) Dominance of Framework Performance Overhead:

Experimental results show that T_{frame} remains highly consistent across different algorithms and tasks (encryption/decryption and signing), with latency consistently maintained around 2-3 ms. This indicates that the contextual chain processing of Spring Boot constitutes a "fixed cost" for system response time.

(2) Performance "Inversion" Driven by Task Types:

Experiments prove that in encryption/decryption scenarios, RSA slightly outperforms SM2 due to the simplicity of its public-key operations. Conversely, in digital signature and

verification scenarios, SM2 exhibits a decisive advantage, with latency only approximately 1/6th that of RSA. Consequently, it can be concluded that while RSA's mathematical computation during private-key signing is extremely intensive [7], the Elliptic Curve Cryptography (ECC)-based SM2 offers superior computational efficiency for high-frequency authentication tasks.

(3) Practical Implications for ShangMi (SM) Migration:

In engineering practice, performance optimization efforts should prioritize reducing Web pipeline overhead (e.g., optimizing AOP logic or serialization frameworks) rather than merely tuning algorithm parameters. Furthermore, in financial-grade scenarios requiring frequent signing,

migrating to the SM2 algorithm not only fulfills regulatory compliance but also significantly reduces CPU latency and enhances system throughput.

4.4. Scenario C: Analysis of Production-Level Stress Testing Results under High Concurrency

This section aims to simulate a real-world industrial-grade Web application environment using the JMeter stress testing tool. By employing persistent connections with Keep-Alive enabled, this experiment quantitatively evaluates the stability, throughput, and system resource utilization of RSA-2048 and SM2-256 algorithms across various concurrency gradients.

4.4.1. Experimental Data Preprocessing and Cleaning

Prior to formal analysis, the raw data generated from the stress tests were cleaned as follows to eliminate interference from uncontrolled factors:

(1) Warm-up Phase Slicing: Due to the Just-In-Time (JIT) compilation mechanism of the JVM, significant performance fluctuations occur during the initial phase of system operation. To ensure that the statistical metrics reflect the steady-state performance of the system, the first 5,000 samples (approximately the first 60 seconds of operation) from each concurrency gradient were uniformly excluded.

(2) Outlier Retention Policy: In contrast to baseline benchmarking, this scenario only filtered out null and negative values. Significant latencies observed under high

concurrency (e.g., data points in the 2,000 ms range) were retained, as this study considers them authentic manifestations of system performance bottlenecks. These data points are essential for quantifying the robustness of the algorithms under extreme stress.

4.4.2. Analysis of End-to-End Throughput and Response Time

By varying the number of concurrent threads (100, 500, and 1,000), we observed the performance fluctuations of the system when processing high-frequency asymmetric encryption tasks.

(1) The encryption performance data measured under the three concurrency levels are presented in **Table 7**:

Table 7. Comparison of Encryption Algorithm Performance Metrics Across Different Concurrency Gradients

Algorithm	Threads	T _{algo} Mean(ms)	RT _{total} Mean(ms)	P95	TPS
RSA	100	14.1083	30.4340	65.82	2641.51
	500	40.7252	103.2661	255.62	3153.20
	1000	61.9916	298.7301	511.31	2863.02
SM2	100	15.8388	36.8787	79.83	2794.81
	500	71.2191	186.5084	377.33	2793.51
	1000	108.5994	370.5538	738.03	2849.67

The experimental results illustrating the evolutionary trends of throughput (TPS) and average response time (RT) during high-frequency encryption tasks are depicted in **Figure 7**.

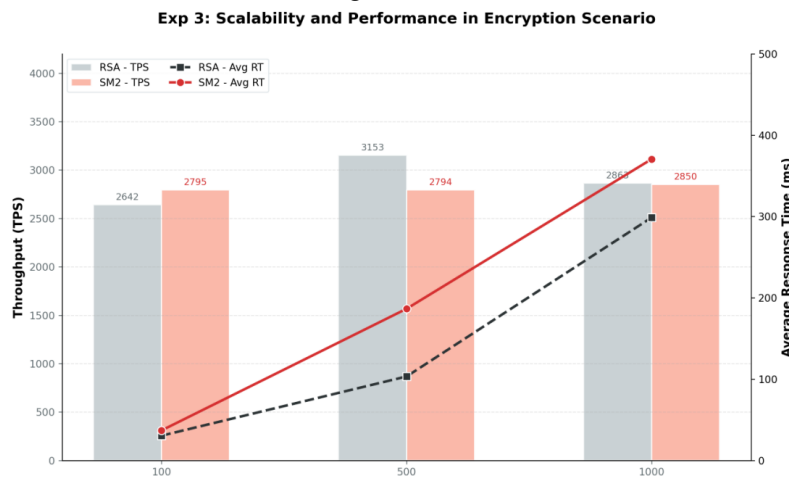


Figure 7. Comparative analysis of encryption performance between RSA and SM2

As illustrated in **Figure 7**, RSA-2048 and SM2-256 exhibit comparable performance in the encryption scenario. The experimental observations are as follows:

a. Linear Growth of Latency: As the number of concurrent threads increases from 100 to 1,000, the average response time (RT) for both algorithms demonstrates a linear growth trend. Notably, at 1,000 concurrency, the growth slope of RSA's RT is slightly lower than that of SM2.

b. Throughput Saturation: The throughput for both RSA and SM2 reaches a saturation threshold at approximately 500

concurrency, peaking at roughly 3,153 TPS and 2,794 TPS, respectively

c. Performance Inversion Phenomenon: Consistent with the baseline testing results, RSA performs slightly better than SM2 in high-concurrency encryption scenarios. This is attributed to the minimal computational overhead of RSA public-key encryption.

(2) The performance data measured in the decryption scenario across the three concurrency levels are presented in **Table 8**:

Table 8. Comparison of Decryption Algorithm Performance Metrics Across Different Concurrency Gradients

Algorithm	Threads	T _{algo} Mean(ms)	RT _{total} Mean(ms)	P95	TPS
RSA	100	298.8585	314.8686	359.02	220.40
	500	1488.3364	1587.8276	2043.03	225.42
	1000	2881.2323	3194.6728	4476.58	231.91
SM2	100	14.3232	33.1507	73.53	3098.31
	500	65.5691	169.6364	360.00	3049.75
	1000	87.9783	331.8421	659.92	3181.55

Decryption performance is a critical metric for evaluating the computational strain on the server side. Based on the empirical data in Table 8, Figure 8 illustrates the "watershed"

performance characteristics of RSA and SM2 when processing decryption tasks in high-concurrency environments.

Exp 3: Scalability Analysis of RSA vs. SM2 (High Concurrency)

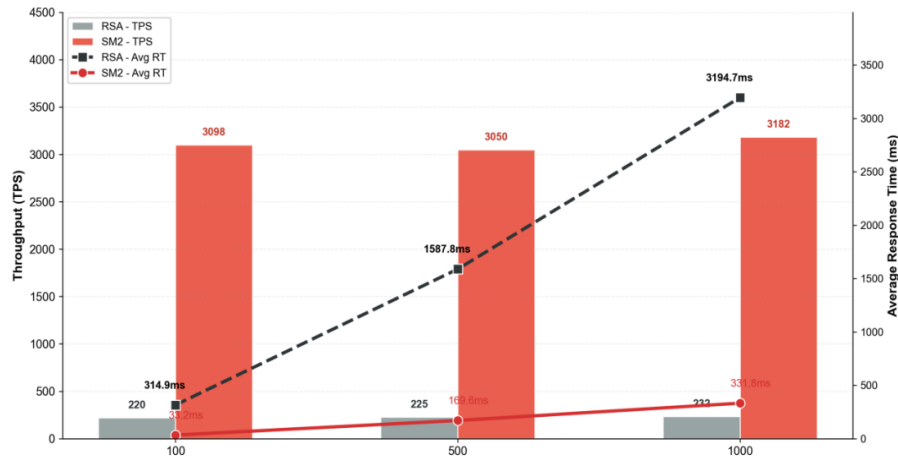


Figure 8. Comparative analysis of decryption performance between RSA and SM2

As illustrated in Figure 8, the performance of the two algorithms in the decryption scenario exhibits a sharp polarization:

a. Performance Degradation of RSA: When concurrency reaches 1,000 threads, the average response time of RSA surges to 3,194.7 ms, while its TPS drops to only 231.9. This indicates that the extreme consumption of CPU power by its modular exponentiation operations has triggered a severe queuing effect.

b. Engineering Resilience of SM2: In contrast, the RT of SM2 remains stable at approximately 331.8 ms under the same load, with a throughput (3,182 TPS) roughly 13.7 times that of RSA. This demonstrates that the SM2 algorithm possesses superior engineering resilience when handling high-concurrency, computationally intensive tasks.

(3) The performance data measured in the digital signature and verification scenario across the three concurrency levels are presented in Table 9:

Table 9. Comparison of Digital Signature and Verification Performance Metrics Across Different Concurrency Gradients

Algorithm	Threads	$T_{algo}Mean(ms)$	$RT_{total}Mean(ms)$	P95	TPS
RSA	100	316.6390	332.9899	377.79	219.78
	500	1579.4887	1677.9099	2184.22	211.51
	1000	3095.4558	3375.0082	4683.52	212.43
SM2	100	35.6914	57.8479	105.91	584.53
	500	178.7378	297.5632	565.13	1540.66
	1000	304.6198	585.9756	1112.02	1668.62

Addressing the identity authentication scenarios in production environments, this study further compares the performance of both algorithms within the digital signature

and verification pipeline. The experimental results are illustrated in Figure 9.

Exp 3: Scalability Analysis of RSA vs. SM2 (High Concurrency)

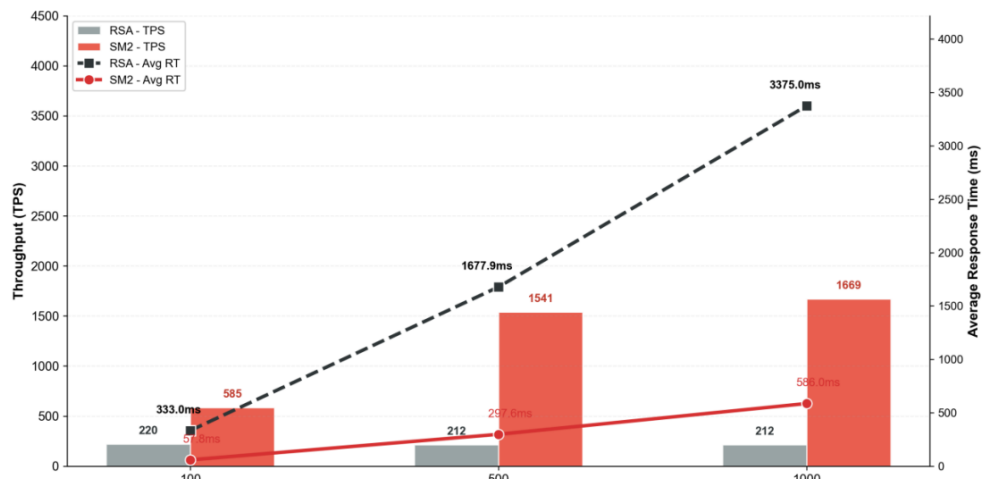


Figure 9. Performance comparison of digital signature and verification algorithms.

Figure 9 reveals that SM2 also demonstrates a significant advantage in identity authentication scenarios. As concurrent pressure increases, the latency curve for RSA rises steeply, whereas the curve for SM2 remains relatively flat. This indicates that within a Spring Boot-based microservices architecture, adopting the SM2 algorithm not only satisfies the compliance requirements of national cryptographic management policies but also significantly reduces CPU

latency during high-frequency authentication.

4.4.3. System Stability and Tail Latency Analysis

Building upon the end-to-end API testing, this section introduces high-percentile metrics (P95 and P99) to provide a deep analysis of response stability and performance inflection points under extreme high-concurrency pressure, as illustrated in **Figure 10**.

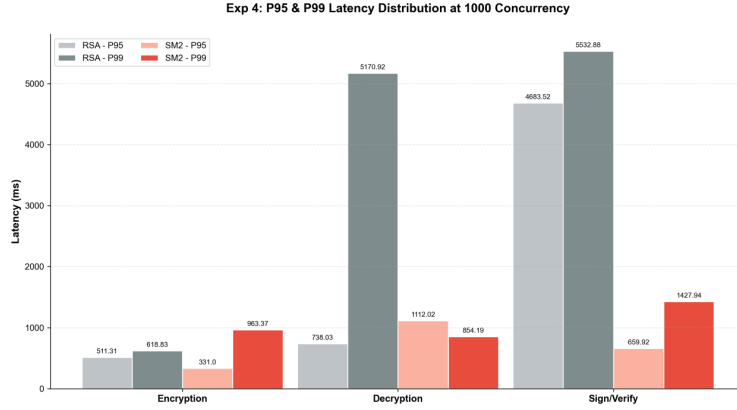


Figure 10. P95 and P99 latency distribution under 1,000 concurrent threads

In summary, the experimental results demonstrate a clear "watershed" in performance between RSA and SM2 under high-concurrency environments:

- (1) Exponential Latency Surge in RSA: When the concurrency reaches 1,000 threads, RSA exhibits a sharp upward trajectory in latency, with P95 and P99 metrics increasing abnormally. This confirms that under the heavy load of 1,000 concurrent threads, the intensive computational overhead of complex large-integer arithmetic during server-side decryption leads to severe queuing effects.
- (2) Operational Stability of SM2: Under the same 1,000-thread pressure, SM2 maintains a higher TPS and minimal latency jitter. This study posits that the short-key

characteristic of SM2 generates fewer temporary BigInteger objects within the JVM heap. Consequently, this significantly reduces the frequency of Minor GC (Garbage Collection) within the JDK 21 environment, ensuring superior performance stability.

4.4.4. Framework Overhead Analysis

To further elucidate the sources of system performance bottlenecks, this study deconstructed the response time under high-concurrency scenarios into two components: algorithm execution time (T_{algo}) and framework overhead (T_{frame}). The proportional distribution of framework overhead for both algorithms is illustrated in **Figure 11**.

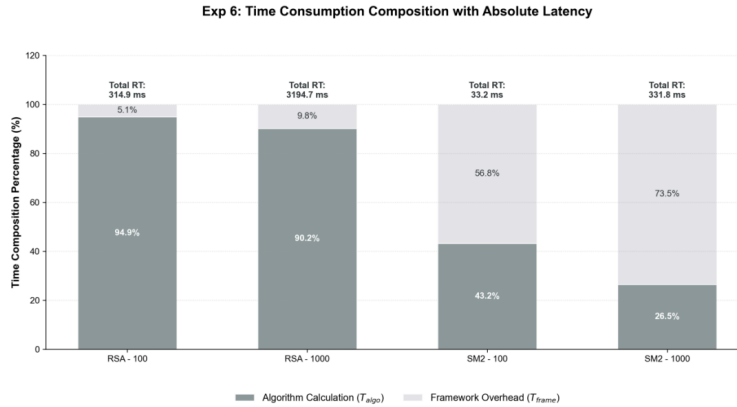


Figure 11. Comparison of framework overhead ratio between the evaluated algorithms

An analysis of **Figure 11** yields the following key insights:

(1) Algorithmic Bottleneck of RSA:

In the RSA-1000 scenario, the total latency reaches 3,194.7 ms, with algorithmic computation (T_{algo}) accounting for 90.2% of the total. This demonstrates that the performance bottleneck of RSA lies entirely within the computational overhead of the algorithm itself. Consequently, optimization at the Spring Boot framework level offers extremely limited potential for overall performance gains.

(2) Framework Sensitivity of SM2:

In stark contrast to RSA, the total latency of SM2 under 1,000 threads is merely 331.8 ms. Notably, the proportion of its framework overhead (T_{frame}) escalates from 56.8% at 100 threads to 73.5%. This shift occurs because the SM2 algorithm executes so rapidly that the majority of system time is diverted to inherent Spring Boot mechanisms, such as Bean management, AOP interception, and serialization/deserialization [6].

(3) Differentiation of Optimization Paths:

Experimental evidence suggests that optimizations for RSA should focus on enhancing computational capacity. Conversely, for SM2-based systems, priority should be given to tuning Spring Boot framework parameters—such as thread pool optimization or reducing interceptor layers—as framework overhead has emerged as the primary bottleneck.

5. Discussion and Conclusion

5.1. Analysis and Discussion of Experimental Results

Through layered experiments, this study identifies distinct performance characteristics between SM2 and RSA within a Spring Boot environment. The underlying engineering logic is analyzed as follows:

(1) Mathematical Drivers of Performance Inversion: The slight edge of RSA-2048 in encryption scenarios stems from its small public exponent ($e = 65537$). Its binary representation contains only two "1"s, allowing the encryption process to be completed with only 16 squarings and 1 multiplication through algorithmic optimization, resulting in extremely low computational overhead [8]. In contrast, SM2 encryption necessitates complex elliptic curve point multiplication, which has a significantly higher mathematical density than RSA's 17 simple operations. However, during decryption and signing, the surge in CPU load caused by large-integer modular exponentiation leads RSA to reach its performance inflection point rapidly under high concurrency [7].

(2) Constancy of Framework Overhead: The experimental model $RT_{total} = T_{algo} + T_{frame}$ demonstrates that T_{frame} remains stable at 2–3 ms across different algorithms. This proves that the overhead from Spring Boot's AOP interception, serialization, and Bean management constitutes a "base environment tax" (infrastructure overhead) of the system, which does not fluctuate drastically with changes in the cryptographic algorithm.

(3) Engineering Resilience under High Concurrency: The mathematical properties of SM2 based on ECDLP (Elliptic Curve Discrete Logarithm Problem) enable it to maintain a stable TPS even under 1,000-thread pressure. Conversely, RSA triggers frequent Minor GC (Garbage Collection) due to the excessive allocation of temporary BigInteger objects, leading to significant queuing effects within the system.

5.2. Overall Summary

Through a quantitative analysis of the data from the preceding scenarios, the following conclusions are established:

(1) Achievement of Research Objectives: This study quantifies the impact of industrial-grade Web frameworks on the execution efficiency of National Cryptographic Algorithms (SM-series). It effectively fills the gap in end-to-end empirical data for such systems.

(2) Core Findings: In scenarios involving decryption and high-frequency authentication (signature and verification), SM2 demonstrates an energy-efficiency ratio far exceeding that of RSA. With response times reduced by approximately 90%, SM2 emerges as the optimal solution for high-performance secure Web communications.

(3) Synergizing Compliance and Performance: Migrating to SM2 achieves a win-win outcome; it not only satisfies the

compliance mandates of national cryptographic management policies but also significantly mitigates server-side CPU strain.

5.3. Engineering Practical Recommendations

(1) Selection Strategy: For high-frequency authentication scenarios in sectors such as finance and government affairs, the adoption of the SM2 algorithm should be prioritized as a mandatory standard.

(2) Optimization Focus: System tuning should concentrate on the T_{frame} component—specifically by optimizing Jackson reflection overhead or streamlining AOP aspect paths—rather than blindly attempting to tune parameters of well-established cryptographic implementations.

(3) Risk Mitigation: If RSA must be maintained within a system, strict rate-limiting mechanisms must be implemented for concurrency gradients exceeding 500 threads to prevent system crashes triggered by CPU exhaustion.

5.4. Limitations and Future Work

(1) Research Limitations: This study is based on the Bouncy Castle software library. It does not yet encompass performance comparisons involving Hardware Security Modules (HSMs) or asynchronous reactive frameworks (such as Spring WebFlux).

(2) Future Directions: Subsequent research may explore the performance enhancement of National Cryptographic Algorithms through hardware acceleration. Additionally, evaluating the performance overhead of mutual TLS (mTLS) using national ciphers within a Service Mesh architecture remains a key area for further investigation [9].

References

- [1] Liu, Y. J. (2025). Application of data encryption technology in computer network information security. *Information Recording Materials*, 26(04), 111–113, 158. (In Chinese)
- [2] Hu, Y. (2025). Application of encryption technology in network communication security. *Information Recording Materials*, 26(02), 129–131. (In Chinese)
- [3] Huo, W., Guo, Q. Q., & Ma, Y. (2020). *Commercial cryptography application and security assessment*. Publishing House of Electronics Industry. (In Chinese)
- [4] Sun, K. Q. (2010). A determination algorithm for factoring large integers in RSA cryptanalysis. *Computer Engineering and Applications*, 46(S1), 142–144. (In Chinese)
- [5] Stallings, W. (2020). *Cryptography and network security: Principles and practice* (8th ed.). Pearson.
- [6] Huang, P. Y., Liao, G. Z., & Ren, J. H. (2024). The application of AES-SM2 hybrid encryption algorithm in big data security and privacy protection. *International Journal of Advanced Computer Science and Applications*, 15(6), 989–995.
- [7] Li, J. L., Mo, Y. N., Su, T., & Chen, D. H. (2022). Hardware design of high-speed hybrid encryption system based on national secret algorithms SM2, SM3, SM4. *Application Research of Computers*, 39(9). (In Chinese)
- [8] Rivest, R. L., & Kaliski, B. (2003). RSA problem. In H. C. A. van Tilborg (Ed.), *Encyclopedia of cryptography and security*. Springer.
- [9] Ma, Z. H. (2024). Research on software-hardware co-design of a hybrid encryption system based on national secret algorithms [Master's thesis, Zhengzhou University]. CNKI. (In Chinese)