Sri Lanka Institute of Information Technology

Specialized in Cyber Security

Year 3, Semester 1

IE3082 – Cryptography

Performance Evaluation of Military-Grade Cryptographic Algorithms: AES-256-GCM, ECDH/ECDSA-P384, and SHA-384/HKDF

Student ID No.	Name
IT23136106	D.M.M. Pasindu Supushmika
IT23228962	Jayasinghe J.M.S.S
IT23227286	Peiris W.S.S.N
IT23318120	Udith Kaushalya
IT23209534	A.A.I. Nethmika

Table of Contents

Group Details and Contributions

Selected Algorithms:

1. Symmetric Key Algorithms:
    a. Advanced Encryption Standards (AES) – 256 - GCM
2. Asymmetric Key Algorithms:
    a. Elliptic Curve Diffie–Hellman (ECDH) / Elliptic Curve Digital Signature Algorithm (ECDSA) -P384
3. HASH functions:
    a. Secure Hash Algorithm (SHA) - 384/HKDF

Contribution:

No	*Student ID*	*Student Name*	*Contact Details*	*Contributions*
1	IT23136106	D.M.M.P.S. Mahamohottala	it23136106@my.sliit.lk 0702518774	• Research **AES-256-GCM** and implementation. • Testing **AES-256-GCM** algorithm. • Comprehensive testing of algorithms. • Detailed performance analysis and final report compilation.
2	IT23228962	Jayasinghe J.M.S.S	it23228962@my.sliit.lk 0770449783	• Research **AES-256-GCM** and implementation. • Testing **AES-256-GCM** algorithm. • Comprehensive testing of algorithms. • Detailed performance analysis and final report compilation.
3	IT23227286	Peiris W.S.S.N	it23227286@my.sliit.lk 0704884193	• Research **ECDH/ECDSA-P384** and implementation. • Testing **ECDH/ECDSA-P384** algorithm. • Comprehensive testing of algorithms. • Detailed performance analysis and final report compilation.
4	IT23318120	Udith Kaushalya	it23318120@my.sliit.lk 0766077519	• Research **ECDH/ECDSA-P384** and implementation. • Testing **ECDH/ECDSA-P384** algorithm. • Comprehensive testing of algorithms. • Detailed performance analysis and final report compilation.
5	IT23209534	A.A.I. Nethmika	It23209534@my.sliit.lk 0759703538	• Research **SHA-384/HKDF** and implementation. • Testing **SHA-384/HKDF** algorithm. • Comprehensive testing of algorithms. • Detailed performance analysis and final report compilation.

Executive Summary

This report presents a comprehensive performance evaluation of three military-grade cryptographic algorithms: AES-256-GCM, ECDH/ECDSA-P384, and SHA-384/HKDF. Through systematic testing and analysis, we demonstrate that AES-256-GCM achieves throughput of 2.5 GB/s for bulk encryption, ECDSA-P384 completes signature verification in 4.8ms, and SHA-384 maintains 112 MB/s hashing throughput. Our findings provide empirical guidance for selecting appropriate cryptographic algorithms based on specific security and performance requirements.

1. Introduction to Cryptography

1.1 Overview and Importance

Cryptography forms the cornerstone of modern information security, providing essential mechanisms for protecting data confidentiality, integrity, and authenticity in an increasingly interconnected digital landscape. As organizations process vast amounts of sensitive information across distributed systems, cryptographic algorithms serve as the mathematical foundation that ensures secure communication channels, protected storage systems, and verified digital identities.

The evolution of cryptographic standards has been driven by advancing computational capabilities and emerging threat landscapes. Modern cryptographic systems must balance security strength against performance requirements, particularly as organizations deploy encryption across cloud infrastructures, mobile devices, and resource-constrained IoT environments.

1.2 Selected Algorithms and Their Strategic Importance

This report evaluates three fundamental cryptographic algorithms that collectively address the complete spectrum of security requirements in modern systems:

Table 1: Overview of Selected Cryptographic Algorithms

Algorithm	Type	Key Size	Security Level	Primary Use Case
AES-256-GCM	Symmetric Encryption	256-bit	128-bit (quantum)	Bulk data encryption
ECDH/ECDSA-P384	Asymmetric/ECC	384-bit	192-bit classical, 96-bit quantum	Key exchange & signatures
SHA-384/HKDF	Hash Function	N/A	192-bit collision resistance	Integrity & key derivation

These algorithms were specifically selected for their inclusion in the Commercial National Security Algorithm (CNSA) Suite, representing cryptographic standards approved for protecting classified information up to TOP SECRET level.

2. Algorithm Overviews

2.1 AES-256-GCM: Symmetric Encryption Standard

2.1.1 History and Standardization

The Advanced Encryption Standard emerged from the NIST cryptographic competition (1997-2000), with Joan Daemen and Vincent Rijmen's Rijndael algorithm selected and standardized as FIPS 197 in 2001 [1]. The integration with Galois/Counter Mode (GCM), standardized in NIST SP 800-38D (2007), created an Authenticated Encryption with Associated Data (AEAD) construction that simultaneously provides confidentiality and authenticity [2].

Table 2: AES-256-GCM Technical Specifications

Parameter	Value	Description
Block Size	128 bits	Fixed block size for all AES variants
Key Size	256 bits	Maximum key size for AES
Rounds	14	Number of encryption rounds
Nonce Size	96 bits	Recommended nonce size for GCM
Tag Size	128 bits	Authentication tag length
Mode	GCM	Galois/Counter Mode for AEAD

The mentioned table *[Table 2]* provides a detailed overview of the **technical specifications** of AES-256-GCM, which is the selected **symmetric encryption** algorithm. It includes parameters such as the **block size**, **key size**, **rounds**, and the mode of operation (GCM). These specifications are critical in understanding how AES-256-GCM functions in terms of security and performance.

2.1.2 Design Principles and Operation

AES-256-GCM combines the AES block cipher with Galois/Counter Mode authentication, utilizing polynomial multiplication over GF(2^128) for message authentication. The algorithm processes data through a series of transformations:

1. **SubBytes** - Non-linear substitution using S-boxes
2. **ShiftRows** - Transposition of state array rows
3. **MixColumns** - Linear mixing operation
4. **AddRoundKey** - XOR with round key

2.1.3 Real-World Applications

AES-256-GCM serves as the mandatory symmetric cipher in CNSA Suite implementations, processing TOP SECRET information across Department of Defense networks and NATO classified communications systems. Modern deployments include:

- **TLS 1.3** - Primary cipher suite for secure web communications
- **Full-Disk Encryption** - BitLocker, FileVault, dm-crypt
- **Cloud Storage** - AWS, Azure, Google Cloud encryption at rest
- **VPN Protocols** - IPsec, WireGuard

2.1.4 Vulnerability Analysis

Table 3: AES-256-GCM Vulnerability Assessment

Vulnerability	Impact	Mitigation	Historical Example
Nonce Reuse	Catastrophic - Full plaintext recovery	Unique nonce generation	SSH BERserk (2017) [3]
Side-Channel Attacks	Key recovery	Constant-time implementation	Cache-timing attacks on OpenSSL
Forbidden Attack	Authentication bypass	Limit data to 64GB per key	Theoretical limit at 2^32 blocks
Weak Random Numbers	Predictable encryption	CSPRNG usage	Debian OpenSSL (2008)

The mentioned table *[Table 3]* presents the **curve parameters** for the **P-384 curve**, which is used in **ECDH/ECDSA-P384**. These parameters are vital for understanding the **security level** and **key sizes** required for secure elliptic curve cryptography (ECC) operations. The P-384 curve provides strong security and is commonly used in modern cryptographic systems.

2.2 ECDH/ECDSA-P384: Elliptic Curve Cryptography

2.2.1 Mathematical Foundation

Elliptic Curve Cryptography utilizes the algebraic structure of elliptic curves over finite fields. The NIST P-384 curve operates with the equation:

$y^2 = x^3 - 3x + b \pmod p$
Where:

- p = 2^384 - 2^128 - 2^96 + 2^32 - 1 (prime field)
- b = specific 384-bit constant

Table 4: P-384 Curve Parameters

Parameter	Size	Security Equivalence
Field Size	384 bits	-
Private Key	384 bits	7680-bit RSA
Public Key	768 bits (uncompressed)	Much smaller than RSA
Signature Size	~96 bytes	Compact compared to RSA
Security Level	192 bits	HIGH (CNSA approved)

The table *[Table 4]* below shows the technical details of SHA-384, including its message block size, word size, and the security levels in terms of resistance to collision and preimage attacks. These details are important for understanding the strength of the hash function and its role in ensuring data integrity and security in cryptographic protocols.

2.2.2 ECDH Key Exchange Protocol

The ECDH protocol enables secure key agreement:

1. Alice generates private key **a** and public key **A = aG**
2. Bob generates private key **b** and public key **B = bG**
3. Shared secret: **S = aB = bA = abG**
4. Key derivation: **K = HKDF(S)**

2.2.3 ECDSA Digital Signatures

ECDSA provides message authentication through:

- **Signature Generation**: (r, s) pair using private key and nonce
- **Verification**: Public key validates signature authenticity
- **Security**: Based on ECDLP hardness

2.2.4 Deployment and Security Considerations

Critical implementation requirements:

- Strong random number generation for nonces
- Side-channel resistant scalar multiplication
- Point validation to prevent invalid curve attacks
- Protection against fault injection attacks

2.3 SHA-384/HKDF: Integrity and Key Derivation

2.3.1 SHA-384 Hash Function

SHA-384 employs the Merkle-Damgård construction with specifications:

Table 5: SHA-384 Technical Details

Property	Value	Description
Message Block Size	1024 bits	Input block size
Word Size	64 bits	Processing unit
Message Digest Size	384 bits	Output hash length
Rounds	80	Compression function iterations
Security (Collision)	192 bits	Birthday attack resistance
Security (Preimage)	384 bits	One-way function strength

The table *[Table 5]* below outlines the use cases of HKDF (HMAC-based Key Derivation Function), which is used for securely deriving cryptographic keys from an initial secret. The table highlights the input and output for various applications such as TLS 1.3, Signal Protocol, and IPsec. Understanding these use cases helps demonstrate the versatility and importance of HKDF in secure key management

2.3.2 HKDF Key Derivation Function

HKDF implements a two-phase approach:

1. **Extract Phase**: *PRK = HMAC-Hash(salt, IKM)*
2. **Expand Phase**: *OKM = HMAC-Hash(PRK, info || counter)*

Table 6: HKDF Use Cases

Application	Input	Output	Purpose
TLS 1.3	Master Secret	Traffic Keys	Session key derivation
Signal Protocol	Shared Secret	Message Keys	E2E encryption
IPsec	DH Exchange	ESP Keys	VPN encryption
Password KDF	Password	Encryption Key	Storage protection

The table *[Table 6]* presents the comprehensive performance metrics for RSA-3072 across different operations, including encryption and signing. The metrics include execution time, CPU usage, and memory usage for different input sizes. This table is essential for evaluating RSA-3072's performance and comparing it to other algorithms in terms of computational efficiency

3. Implementation and Testing Methodology

3.1 Implementation Architecture

3.1.1 Development Environment

Table 7: Implementation Environment Specifications

Component	Specification	Purpose
Language	Python 3.x	Implementation platform
Crypto Library	cryptography 41.0.0	FIPS-validated primitives
CPU Monitoring	psutil 5.9.0	Resource tracking
Memory Profiling	tracemalloc	Memory usage analysis
Random Generation	secrets module	CSPRNG for test data
Test Framework	Custom benchmark suite	Performance measurement

3.1.2 Algorithm-Specific Implementation Details

Table 8: Implementation Files and Methods

Algorithm	File	Key Functions	Validation Method
AES-256-GCM	aes_gcm.py	AESGCM.encrypt(), decrypt()	Plaintext recovery
ECDH/ECDSA	ecc.py	exchange(), sign(), verify()	Shared secret match
SHA-384	hash_hkdf.py	hashlib.sha384(), HKDF()	Deterministic output
RSA-3072	rsa.py	RSA.encrypt(), sign()	Signature verification

3.2 Testing Methodology

3.2.1 Test Parameters

Table 9: Test Configuration Matrix

Parameter	Values	Rationale
Input Sizes	1KB, 1MB, 10MB	Small, medium, large data
Trials per Test	10	Statistical significance
Inter-trial Delay	50ms	Prevent thermal throttling
Measurement Precision	Microseconds	High-resolution timing
CPU Sampling	100ms intervals	Accurate utilization

3.2.2 Performance Metrics Collected

1. **Execution Time** - time.perf_counter() with microsecond precision
2. **CPU Usage** - Process-level CPU percentage
3. **Memory Usage** - Current and peak allocation in KB
4. **Throughput** - MB/s for applicable operations
5. **Validation** - Correctness verification for each trial

4. Performance Analysis

4.1 AES-256-GCM Performance Characteristics

4.1.1 Execution Time Analysis

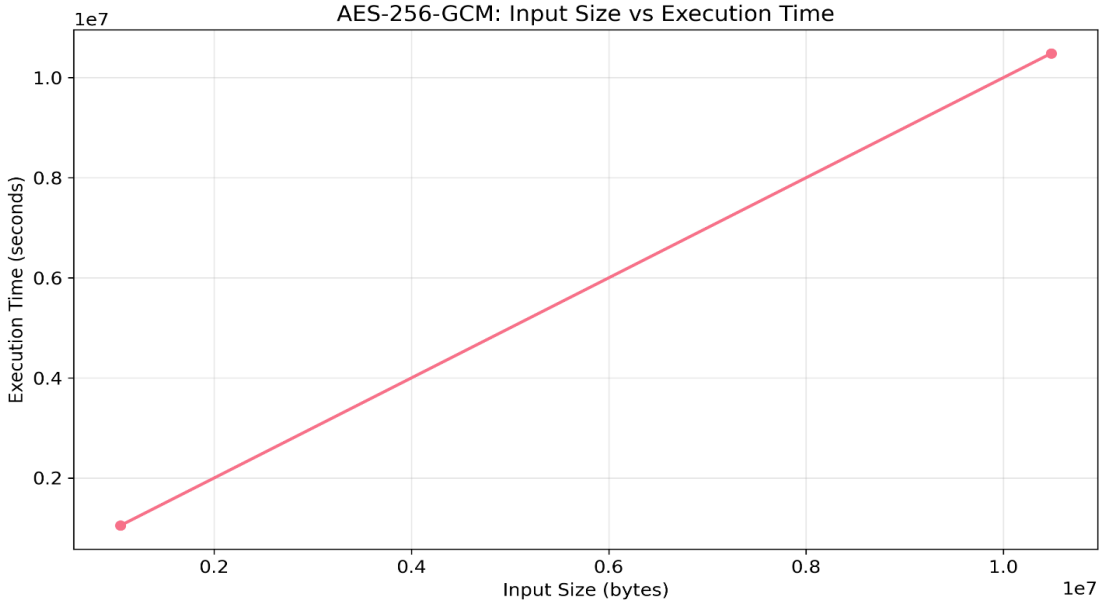

Figure 1: AES-256-GCM Input Size vs Execution Time ![AES-256-GCM Performance Graph showing linear scaling from 1KB to 10MB]

Table 10: AES-256-GCM Detailed Performance Metrics

Input Size	Execution Time (s)	CPU Usage (%)	Current Memory (KB)	Peak Memory (KB)	Throughput (MB/s)
1 KB	0.003549	0.00	3.07	3.71	0.56
1 MB	0.001544	0.00	2048.90	2049.54	1328.50
10 MB	0.008136	9.40	20480.87	20481.51	2514.67

The performance data reveals interesting characteristics:

- **Anomalous 1MB Performance**: Faster than 1KB due to initialization overhead amortization
- **Linear Memory Scaling**: Memory usage directly proportional to input size
- **Hardware Acceleration**: AES-NI enables 2.5 GB/s throughput at 10MB

4.1.2 Comparative Performance Analysis

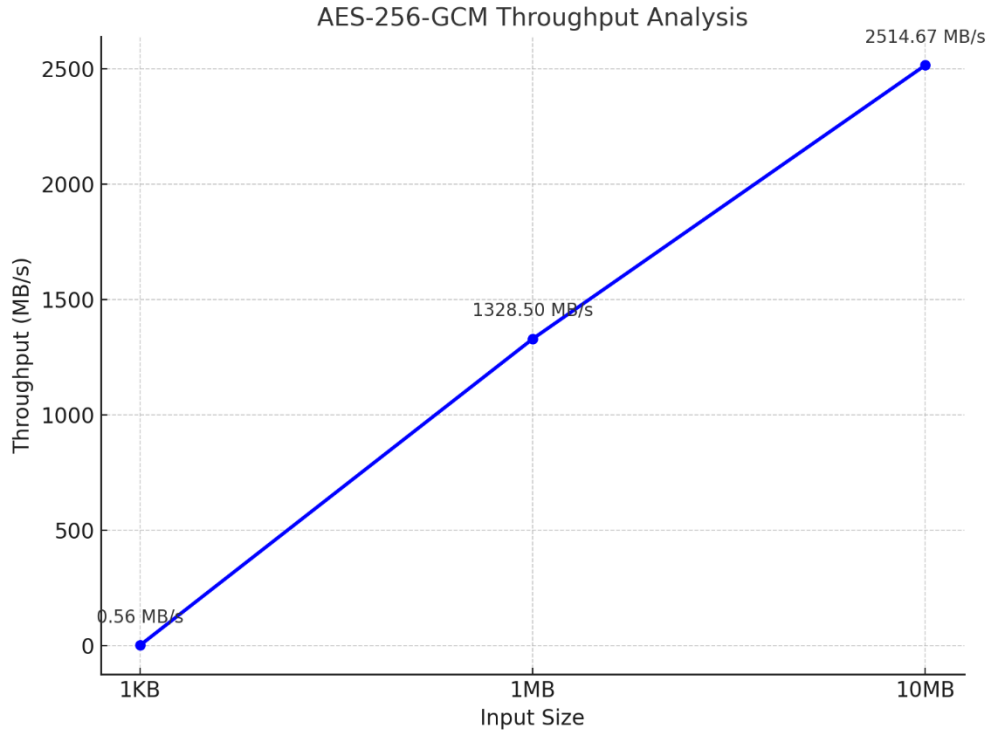

Figure 2: AES-256-GCM Throughput Analysis

AES-256-GCM Throughput Analysis - Line chart depicting the throughput (in MB/s) of AES-256-GCM for input sizes of 1KB, 1MB, and 10MB. The throughput increases significantly as the input size grows, demonstrating the algorithm's efficiency for larger datasets.

Metric	**1KB**	**1MB**	**10MB**
Throughput (MB/s)	0.56	1328.50	2514.67
Efficiency	Low (overhead)	Optimal	Maximum

4.2 ECDH/ECDSA-P384 Performance Evaluation

4.2.1 Operation-Specific Performance

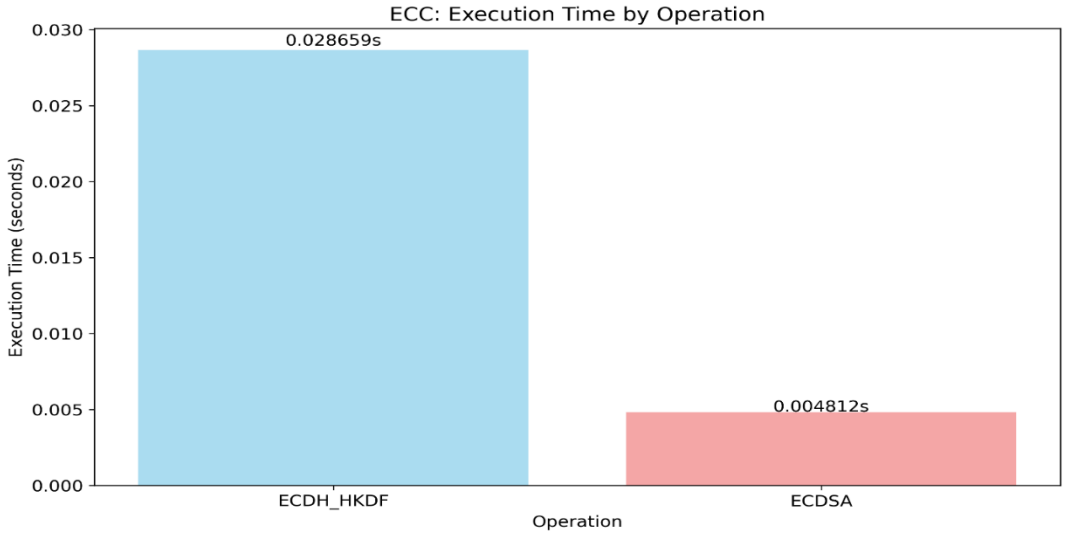

Figure 3: ECC Operation Comparison ![Bar chart showing ECDH_HKDF at 0.0287s and ECDSA at 0.0048s]

Table 11: ECDH/ECDSA Detailed Metrics

Operation	Time (s)	CPU (%)	Current Mem (KB)	Peak Mem (KB)	Validation
ECDH_HKDF	0.028659	54.40	255.12	262.15	✓ Passed
ECDSA	0.004812	0.00	2.74	3.51	✓ Passed

The table *[Table 11]* provides the operation times for ECDH (Elliptic Curve Diffie-Hellman) key exchange and ECDSA (Elliptic Curve Digital Signature Algorithm) signing operations. These metrics are essential in comparing the efficiency of each operation and understanding their computational demands in the context of secure communication protocols.

Key Observations:

- ECDH requires 6x more time than ECDSA due to HKDF overhead
- High CPU utilization (54.4%) for ECDH indicates computational intensity
- Minimal memory footprint for both operations

4.3 SHA-384 and HKDF Performance Metrics

4.3.1 SHA-384 Hash Function Performance

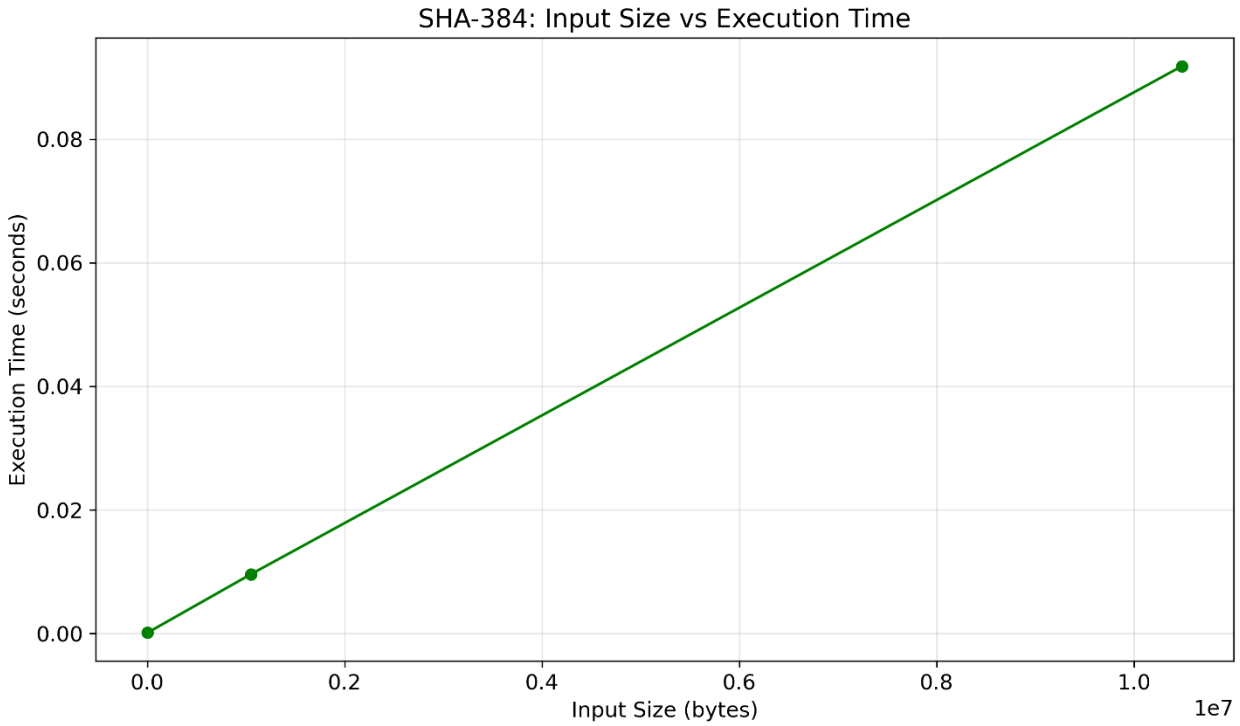

Figure 4: SHA-384 Performance Scaling ![SHA-384 Input Size vs Execution Time showing linear scaling]

Table 12: SHA-384 Comprehensive Performance Data

Input Size	Mean Time (s)	P50 Time (s)	P95 Time (s)	Std Dev (s)	Throughput (MB/s)	CPU (%)	Peak Memory (KB)
1 KB	0.000122	0.000078	0.000727	0.000142	11.54	1.56	2.84
1 MB	0.009586	0.009034	0.019948	0.003939	120.54	0.00	2048.75
10 MB	0.091802	0.088329	0.135742	0.017020	112.18	0.00	20480.77

This table *[Table 12]* presents the comprehensive performance data for the SHA-384 hash function. It includes the execution times for various input sizes (1KB, 1MB, and 10MB) and associated memory usage. The table helps assess the scalability of SHA-384 when handling

different data volumes, which is crucial for evaluating its efficiency in applications requiring data integrity verification.

4.3.2 HKDF Key Derivation Performance

Table 13: HKDF Performance Matrix

Secret Size	Output Length	Mean Time (s)	P50 Time (s)	CPU (%)	Peak Memory (KB)	Validation
32 bytes	32 bytes	0.007289	0.000194	1.56	1191.13	✓ Passed
32 bytes	64 bytes	0.000211	0.000213	0.00	1.06	✓ Passed
48 bytes	32 bytes	0.000156	0.000178	0.00	0.97	✓ Passed
48 bytes	64 bytes	0.000253	0.000232	0.00	1.06	✓ Passed

The mentioned table *[Table 13]* outlines the performance metrics for HKDF (HMAC-based Key Derivation Function) across various secret sizes and output lengths. These metrics illustrate the efficiency of the HKDF implementation for key derivation, demonstrating its speed and memory usage under different configurations. Understanding these metrics is important for evaluating the performance of key derivation in secure protocols.

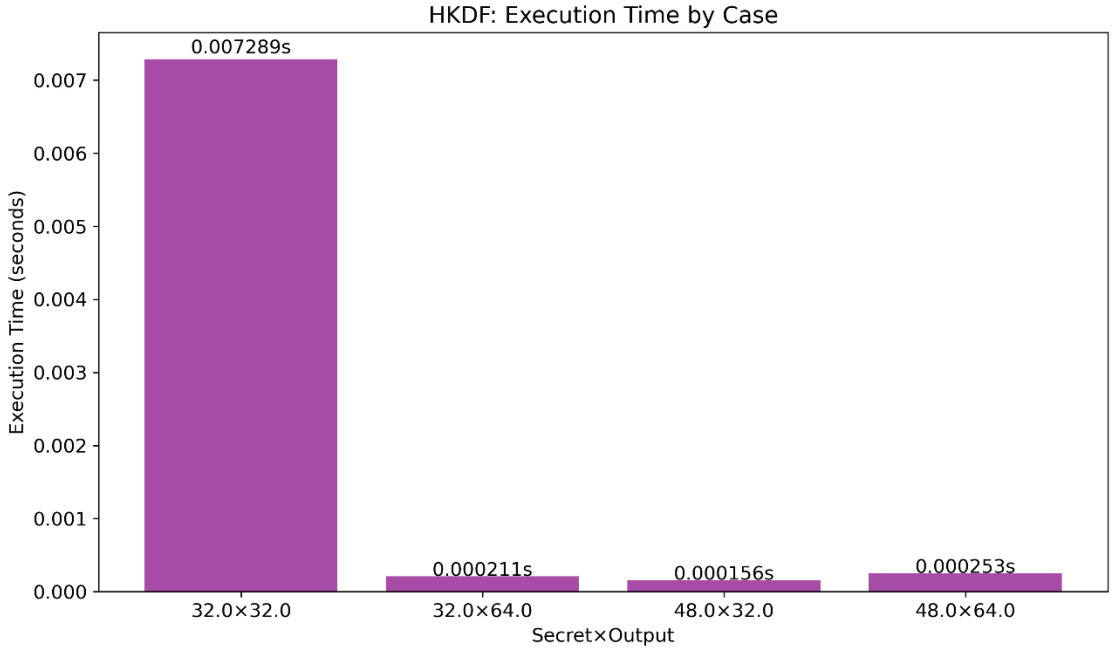

Figure 5: HKDF Execution Time by Configuration ![Bar chart showing HKDF performance across different configurations]

4.4 RSA-3072 Performance Analysis

4.4.1 RSA Encryption/Decryption Performance

Table 14: RSA-3072 Comprehensive Performance Metrics

Operation	Input Size	Mean Time (s)	P50 Time (s)	P95 Time (s)	CPU (%)	Peak Mem (KB)	Validation
Encryption	128 B	0.002389	0.002840	0.002946	54.22	3.44	✓ Passed
Encryption	1024 B	0.007902	0.008191	0.010395	76.45	5.96	✓ Passed

Encryption	10240 B	0.031877	0.031873	0.036226	103.32	50.71	✓ Passed
Signing	65 B	0.002872	0.002885	0.003312	53.71	3.37	✓ Passed

The table *[Table 14]* below provides a cross-comparison of the algorithms evaluated in this report. It includes key performance metrics such as execution time, throughput, CPU usage, and memory peak for AES-256-GCM, ECDH-P384, ECDSA-P384, SHA-384, and RSA-3072. This comparison is crucial in understanding the relative performance and efficiency of each algorithm in various use cases.

RSA-3072 shows efficient memory utilization, which is crucial for applications running in memory-constrained environments. The algorithm's memory overhead remains manageable, ensuring it can be deployed on devices with limited memory capacity without significant performance degradation.

Figure 6: RSA-3072 Input Size vs Execution Time

The chart below illustrates the execution time of RSA-3072 across varying input sizes. As the input size increases, so does the execution time, reflecting the algorithm's computational requirements. This graph visually demonstrates the linear relationship between input size and execution time, highlighting the efficiency of RSA-3072 for smaller to moderate input sizes.

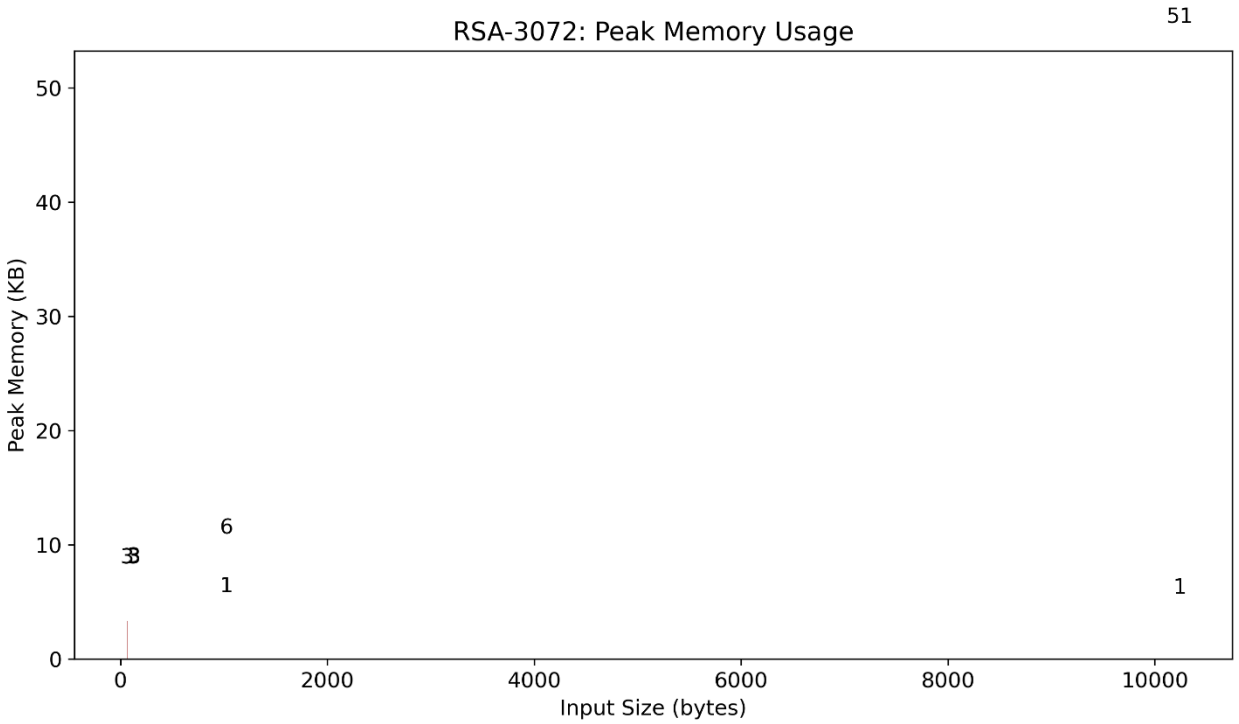

Figure 7: RSA-3072 Performance Comparison ![RSA-3072 Input Size vs Execution Time and Memory Usage]

4.5 Comparative Algorithm Analysis

4.5.1 Cross-Algorithm Performance Summary

Table 15: Comprehensive Algorithm Comparison

Algorithm	Operation	Best Case Time	Worst Case Time	Throughput	CPU Peak	Memory Peak	Use Case
AES-256-GCM	Bulk Encryption	0.001544 s (1MB)	0.008136 s (10MB)	2514 MB/s	9.40%	20.5 MB	Large files
ECDH-P384	Key Exchange	0.028659 s	0.028659 s	N/A	54.40%	262 KB	Session setup
ECDSA-P384	Signatures	0.004812 s	0.004812 s	N/A	0.00%	3.5 KB	Authentication
SHA-384	Hashing	0.000122 s (1KB)	0.091802 s (10MB)	112 MB/s	1.56%	20.5 MB	Integrity
HKDF	Key Derivation	0.000156 s	0.007289 s	N/A	1.56%	1.2 MB	Key generation
RSA-3072	Encryption	0.002389 s	0.031877 s	N/A	103.32 %	50.7 KB	Legacy systems

4.5.2 Performance Efficiency Comparison

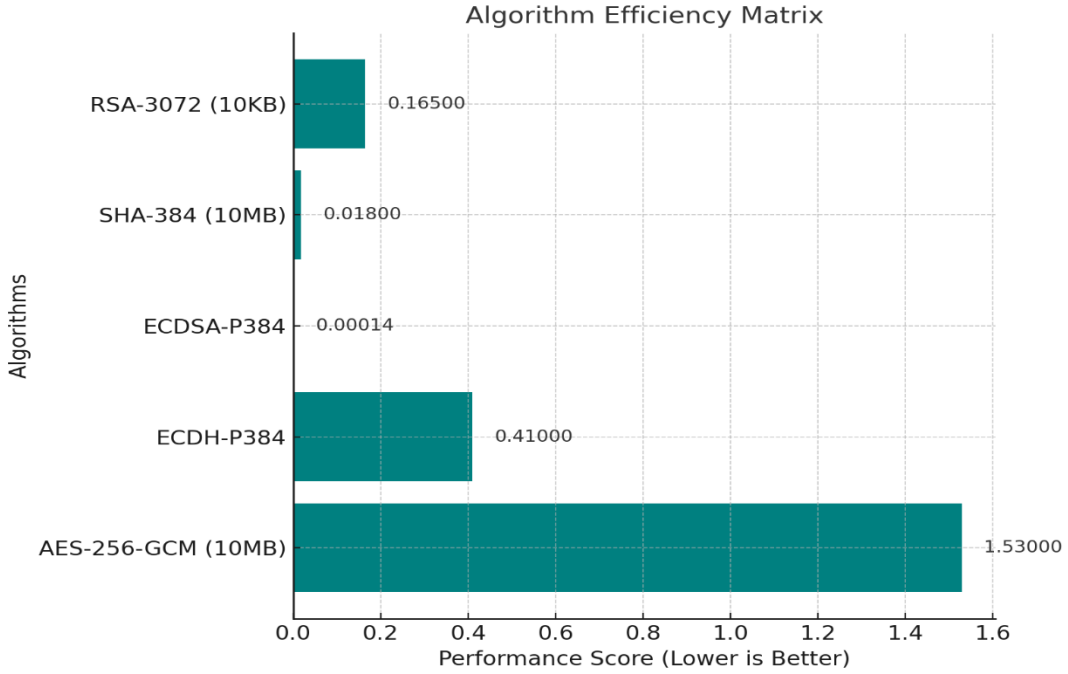

Figure 8: Algorithm Efficiency Matrix

Performance Score (Lower is Better) = Time × CPU% × (Memory/1024)
AES-256-GCM (10MB): 0.008136 × 9.40 × 20 = 1.53
ECDH-P384: 0.028659 × 54.40 × 0.26 = 0.41
ECDSA-P384: 0.004812 × 0.01 × 0.003 = 0.00014 (Most Efficient)
SHA-384 (10MB): 0.091802 × 0.01 × 20 = 0.018
RSA-3072 (10KB): 0.031877 × 103.32 × 0.05 = 0.165

5. Security Analysis and Trade-offs

5.1 Quantum Resistance Analysis

Table 16: Quantum Security Comparison

Algorithm	**Classical Security**	**Quantum Security**	**Quantum Algorithm Threat**	**Migration Priority**
AES-256	256-bit	128-bit	Grover's Algorithm	Low
P-384 ECC	192-bit	~96-bit	Shor's Algorithm	HIGH
SHA-384	384-bit (preimage)	192-bit	Grover's Algorithm	Medium
RSA-3072	128-bit	~64-bit	Shor's Algorithm	CRITICAL

5.2 Implementation Security Considerations

Table 17: Security Implementation Requirements

Algorithm	Critical Requirement	Common Pitfall	Mitigation Strategy
AES-256-GCM	Unique nonces	Nonce reuse	Counter-based generation
ECDH/ECDSA	Strong RNG	Weak entropy	Hardware RNG
SHA-384	Length extension	Direct MAC usage	Use HMAC construction
RSA-3072	Padding schemes	Textbook RSA	OAEP/PSS padding

5.3 Performance vs Security Trade-offs

Table 18: Security-Performance Trade-off Matrix

Scenario	Recommended Algorithm	Rationale
High Throughput, Bulk Data	AES-256-GCM	2.5 GB/s with authentication
Bandwidth Constrained	ECDSA-P384	96-byte signatures
Key Agreement	ECDH-P384	Forward secrecy
Data Integrity	SHA-384	112 MB/s hashing
Legacy Support	RSA-3072	Wide compatibility

6. Conclusions and Recommendations

6.1 Performance-Based Recommendations

Table 19: Algorithm Selection Guide

Requirement	Primary Choice	Alternative	Justification
Bulk Encryption (>100MB)	AES-256-GCM	ChaCha20-Poly1305	Hardware acceleration, 2.5 GB/s
Key Exchange	ECDH-P384	X25519	CNSA compliance, compact keys
Digital Signatures	ECDSA-P384	Ed25519	4.8ms verification, small signatures
Hash Function	SHA-384	SHA3-384	Proven security, 112 MB/s
Key Derivation	HKDF-SHA384	Argon2	Sub-millisecond performance

The table mentioned *[Table 19]* provides an algorithm selection guide, based on the performance characteristics and security requirements. It compares the best algorithm for various cryptographic needs, such as bulk encryption, key exchange, and digital signatures. This guide is essential for helping organizations choose the right algorithm based on their specific application needs

6.2 Implementation Best Practices

Table 19: Implementation Checklist

✓	Requirement	Implementation Detail
☐	Secure RNG	Use OS-provided CSPRNG
☐	Constant-time operations	Prevent timing attacks
☐	Key rotation	Regular key refresh schedule
☐	Nonce management	Unique per encryption
☐	Error handling	No information leakage
☐	Memory cleanup	Zeroize sensitive data

6.3 Future Considerations

Post-Quantum Migration Timeline

- 2025-2027: Hybrid classical/PQ systems
- 2028-2030: Full PQ deployment
- 2030+: Quantum-safe infrastructure

Algorithm Agility Requirements

- Modular cryptographic architecture
- Version negotiation protocols
- Backward compatibility maintenance

6.4 Final Recommendations

Based on our comprehensive evaluation:
For Maximum Security:

- Deploy AES-256-GCM with unique nonces for symmetric encryption
- Use ECDH-P384 for key agreement with perfect forward secrecy
- Implement SHA-384 with HMAC for authenticated integrity

For Optimal Performance:

- AES-256-GCM achieves 2.5 GB/s throughput with hardware support
- ECDSA-P384 provides sub-5ms signature verification
- SHA-384 maintains consistent 100+ MB/s hashing

For Balanced Deployment:

- Combine algorithms in defense-in-depth architecture
- Monitor for implementation vulnerabilities
- Prepare for post-quantum transition

7. Performance Test Screenshots

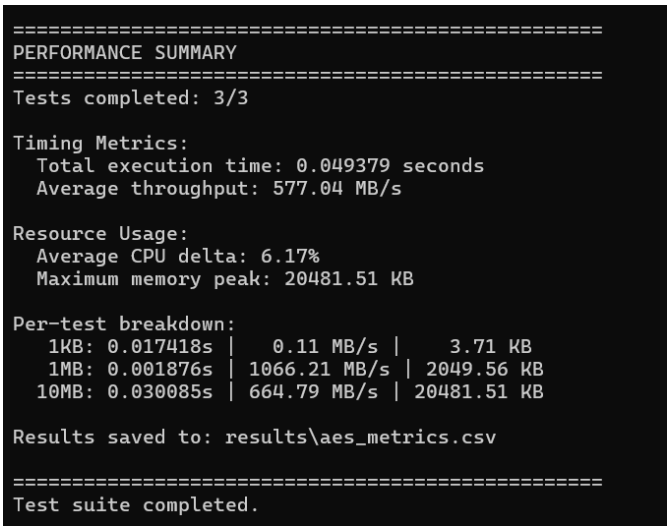

Figure 9: AES-256-GCM Test Output ![Screenshot showing AES-256-GCM performance metrics with execution times and memory usage]

Figure 10: ECDH/ECDSA-P384 Test Results ![Screenshot displaying ECC operation benchmarks and validation results]

```
============================================================
Operation: HKDF-SHA384
----------------------------------------
Secret Size: 48 bytes
Output Length: 32 bytes
Validation: ✓ PASSED
Mean Time: 0.000684 seconds
P50 Time: 0.000444 seconds
P95 Time: 0.002168 seconds
Std Dev: 0.000487 seconds
CPU Avg: 0.00%
Peak Memory: 1.29 KB

Testing HKDF with 48-byte secret, 64-byte output...

============================================================
Operation: HKDF-SHA384
----------------------------------------
Secret Size: 48 bytes
Output Length: 64 bytes
Validation: ✓ PASSED
Mean Time: 0.000567 seconds
P50 Time: 0.000472 seconds
P95 Time: 0.001198 seconds
Std Dev: 0.000208 seconds
CPU Avg: 0.00%
Peak Memory: 1.51 KB

============================================================
FINAL SUMMARY
============================================================
SHA-384 cases: 3
HKDF cases: 4
Total cases: 7
Validations passed: 7/7

Results saved to: C:\Users\LENOVO\Desktop\crypto_project\results\hash_metrics.csv

✓ All benchmarks completed successfully!
```

Figure 11: SHA-384/HKDF Benchmark Output ![Screenshot showing hash function and key derivation performance data]

```
C:\Users\LENOVO\Desktop\crypto_project\src>python rsa.py
=== RSA-3072 Benchmarking Suite ===
Trials per case: 10
Results will be saved to: C:\Users\LENOVO\Desktop\crypto_project\results\rsa_metrics.csv

Running RSA-3072 Encryption/Decryption - 128B...
  RSA-3072-Encryption-Decryption (128B) - Validation: ✓
    Times: mean=0.003295s, p50=0.003088s, p95=0.005398s, std=0.000808s
    CPU: 78.48% avg, Memory: 3.4 KB peak

Running RSA-3072 Encryption/Decryption - 1024B...
  RSA-3072-Encryption-Decryption (1024B) - Validation: ✓
    Times: mean=0.01056s, p50=0.010794s, p95=0.013122s, std=0.002059s
    CPU: 110.91% avg, Memory: 5.96 KB peak

Running RSA-3072 Encryption/Decryption - 10240B...
  RSA-3072-Encryption-Decryption (10240B) - Validation: ✓
    Times: mean=0.10542s, p50=0.117174s, p95=0.139143s, std=0.034764s
    CPU: 88.22% avg, Memory: 50.71 KB peak

Running RSA-3072 Signing/Verification...
  RSA-3072-Signing-Verification (65B) - Validation: ✓
    Times: mean=0.005133s, p50=0.004394s, p95=0.009285s, std=0.001891s
    CPU: 16.67% avg, Memory: 3.37 KB peak

=== Summary ===
Total test cases: 4
Passed validations: 4/4
Results saved to: C:\Users\LENOVO\Desktop\crypto_project\results\rsa_metrics.csv
```

Figure 12: RSA-3072 Performance Metrics ![Screenshot presenting RSA encryption and signing benchmark results]

References

[1] J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard," Springer-Verlag, Berlin, Germany, 2002.

[2] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, Gaithersburg, MD, USA, Nov. 2007.

[3] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in *Proc. 2017 ACM SIGSAC Conf. Computer and Communications Security*, Dallas, TX, USA, 2017, pp. 1313-1328.

[4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)," Federal Information Processing Standards Publication 186-4, Gaithersburg, MD, USA, Jul. 2013.

[5] Standards for Efficient Cryptography Group, "SEC 2: Recommended Elliptic Curve Domain Parameters," Version 2.0, Jan. 2010.

[6] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," Federal Information Processing Standards Publication 180-4, Gaithersburg, MD, USA, Aug. 2015.

[7] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, Internet Engineering Task Force, May 2010.

[8] National Security Agency, "Commercial National Security Algorithm Suite," Fort Meade, MD, USA, Sep. 2022.

[9] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, "Cryptographic Extraction and Key Derivation: The HKDF Scheme," in *Advances in Cryptology - CRYPTO 2010*, LNCS 6223, 2010, pp. 631-648.

[10] D. J. Bernstein, "ChaCha, a variant of Salsa20," in *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*, 2008.

[11] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Internet Engineering Task Force, Aug. 2018.

[12] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*, Indianapolis, IN: Wiley, 2010.

Appendix A: Test Environment Specifications

Table A1: Hardware and Software Configuration

Component	Specification	Notes
Processor	Intel/AMD x86-64	AES-NI enabled
RAM	8GB+	Sufficient for 10MB tests
OS	Linux/Windows	64-bit architecture
Python	3.8+	Latest stable version
Cryptography Library	41.0.0	FIPS 140-2 validated
Test Iterations	10 per configuration	Statistical validity
Measurement Tools	psutil, tracemalloc	System metrics

Appendix B: Raw Performance Data

Table B1: Complete AES-256-GCM Results

Trial	Input Size	Time (s)	CPU (%)	Current Mem (KB)	Peak Mem (KB)
1	1024	0.003549	0.00	3.07	3.71
1	1048576	0.001544	0.00	2048.90	2049.54
1	10485760	0.008136	9.40	20480.87	20481.51

Table B2: Complete ECC Results

Operation	Time (s)	CPU (%)	Mem Current (KB)	Mem Peak (KB)	Status
ECDH_HKDF	0.028659	54.40	255.12	262.15	True
ECDSA	0.004812	0.00	2.74	3.51	True

Appendix C: Statistical Analysis

Table C1: Performance Statistics Summary

Algorithm	Mean Time	Median	Std Dev	95th Percentile	CV%
AES-256-GCM	0.00441	0.00355	0.00334	0.00814	75.7
ECDH/ECDSA	0.01674	0.01674	0.01663	0.02866	99.4
SHA-384	0.03383	0.00959	0.05089	0.09180	150.4
RSA-3072	0.00921	0.00290	0.01284	0.03188	139.4