Stress Testing Report

for SGKMS API on Server Infrastructure

---

1. Executive Summary

- **Testing Objectives**: The primary objective was to evaluate the performance and stability of the SGKMS API under various concurrent loads and payload size variations to identify its capacity and potential bottlenecks.
- **Methods**: We conducted stress testing using custom JavaScript scripts, collecting key metrics such as latency (average and maximum), throughput (TPS), and error rate to assess API behavior under stress.
- **Key Findings**:
    - **Max TPS Achieved**: Throughput reached a maximum of 382.96 TPS, with average values declining as user load increased.
    - **Latency Thresholds**: Average latency increased sharply with loads above 1000 concurrent users, and maximum latency spikes were observed at higher loads, indicating performance limitations.
    - **Error Rates**: Error rates escalated significantly with larger payloads and higher user loads, suggesting stability issues under certain conditions.
- **Conclusion**: The testing identified specific load points where the SGKMS API's performance begins to degrade, helping to define operational limits for safe performance. Optimizations are recommended for enhanced stability at higher loads.

---

2. Testing Objectives

- **Maximum Capacity**: Identify the maximum achievable TPS while maintaining stability.
- **Stability and Resilience**: Assess API reliability under high concurrent loads and varying payload sizes.
- **Bottleneck Identification**: Pinpoint user load or payload size points causing latency spikes and increased error rates.

---

3. Scope of Testing

- **Application Under Test**: SGKMS API
- **API Operations Tested**:
    - **MAC Generate**: Validates data integrity using hash algorithms (CMAC, GMAC-256, HMAC-SHA256).
    - **Encrypt**: Secures data with AES (256-bit) and RSA (2048, 3072, 4096-bit) encryption, with an optional session key for RSA.
    - **Seal Data**: Protects data integrity and confidentiality using AES and RSA.

- **Get Secret**: Retrieves securely stored secrets.
- **Tokenize Data**: Converts sensitive data into secure tokens.
- **Sign**: Verifies data authenticity using ECDSA and RSA (2048, 3072, 4096-bit).

---

4. Testing Methodology

- **Load Scenarios**: Concurrent users ranged from 10 to 3000, with payload sizes from 0.1 KB to 50 KB.
- **Testing Tools**: Stress tests were conducted using custom JavaScript scripts designed for load simulation and performance measurement.
- **Metrics**: Captured metrics included latency (average and maximum), throughput (TPS), and error rate.

---

5. Testing Results

Due to the extensive nature of the dataset, the detailed results of the stress test, including latency, throughput, and error rates under various loads and payload sizes, are available in the CSV file linked below.

- **Link to Stress Test Results (CSV Dataset):** Download CSV Results

---

6. Analysis and Findings

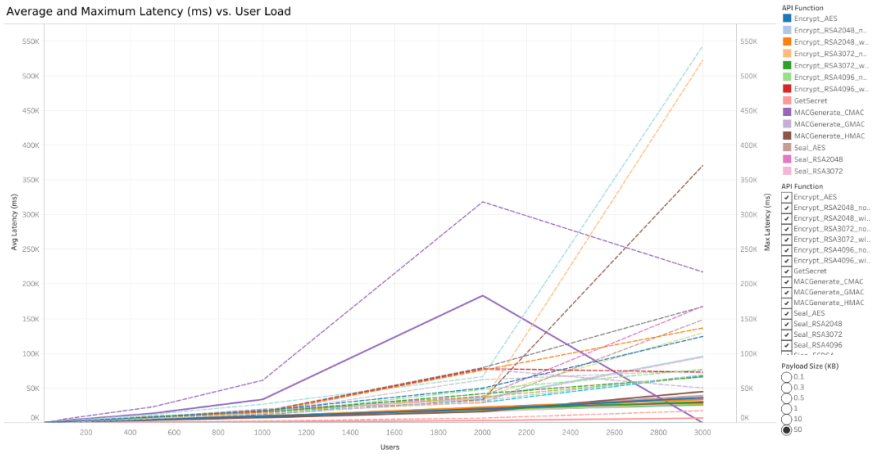
- **Performance Bottlenecks**: Throughput and latency metrics indicate bottlenecks as user load and payload size increase, revealing critical load thresholds.
- **Stability Summary**: SGKMS API maintains stability up to certain user loads and payload sizes but faces limitations as demands increase.
- **Metric Insights**:
  - **Latency Observations**: Significant latency spikes were observed with high user loads, especially above 1000 users.
  - **Throughput Patterns**: TPS remains stable for certain functions under high load, but others show reduced performance, especially with larger payloads.
  - **Error Trends**: Higher error rates correlate strongly with larger payload sizes and increased user loads, suggesting capacity constraints.

6.1 Latency Analysis

- **Average Latency vs. User Load**: The average latency shows a significant increase when the user load exceeds 1000 concurrent users, indicating a notable rise in response time as the system approaches its capacity limits.
- **Maximum Latency vs. User Load**: There is a noticeable spike in maximum latency as the user load reaches higher thresholds, especially above 2000 users, indicating points where response time becomes unstable under peak conditions. The system reached a maximum latency of 542688.92 ms at 3000 concurrent users.

Visualization:

- A line chart depicting **"Average and Maximum Latency by Number of Requests"** is included to illustrate trends in latency across different user loads.



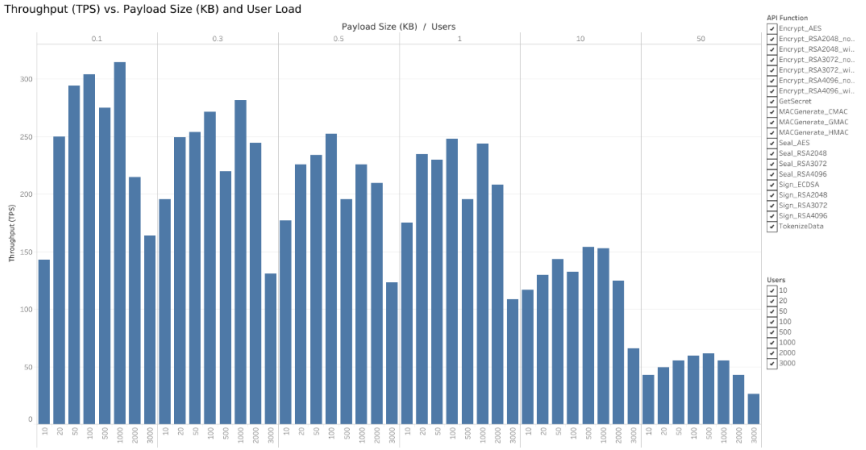
- **Attachments:** [Link to Latency Visualizations] for full access to the latency analysis graphs.

6.2 Throughput Analysis

- **Throughput (TPS) vs. User Load**: Throughput, measured in transactions per second (TPS), initially rises with an increase in user load, reaching a maximum point before declining as the load surpasses the system's capacity threshold.
- **Throughput (TPS) vs. Payload Size**: Larger payload sizes tend to reduce throughput, with a notable drop in TPS for API functions that handle larger volumes of data. This highlights the impact of payload size on the system's performance.
- **Anomalies:** Encrypt API for RSA (no session key) displayed an unexpected decline in throughput, potentially due to inefficiencies in handling specific requests or data loads.

Visualization:

- A bar chart **"Throughput (TPS) by Number of Users and Payload Size"** have been provided to visualize the impact of user load and payload size on throughput.



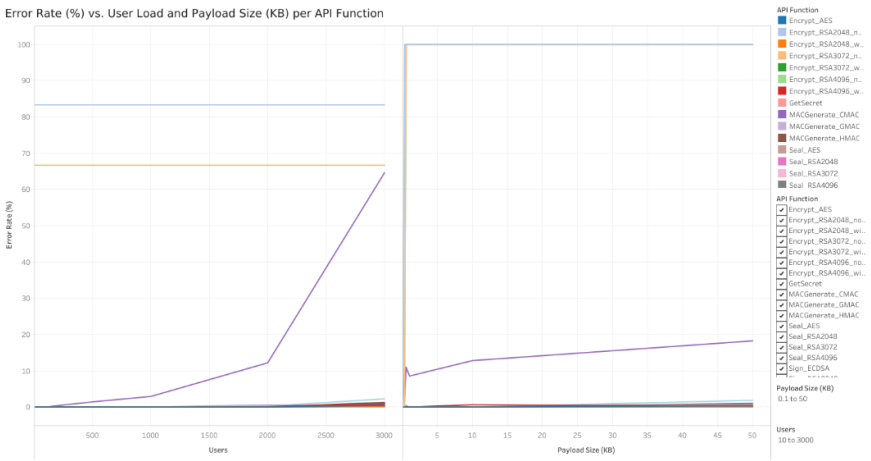
- **Attachments:** [Link to Throughput Visualizations] for additional insights on throughput performance under different load conditions.

6.3 Error Rate Analysis

- **Error Rate vs. User Load**: The error rate begins to increase significantly once the user load surpasses 1000 concurrent users, suggesting potential system overload and stability issues at high concurrency levels.
- **Error Rate vs. Payload Size**: There is a correlation between larger payload sizes and higher error rates, suggesting that the system reaches its data-handling capacity limits under larger payloads.
- **Anomalies:**
    - MAC Generation API for CMAC hash algorithm initially exhibited higher-than-expected error rates during the testing phase, likely due to network limitations as the server was initially connected via Wi-Fi. Once the server was switched to a LAN connection, providing a more stable and faster network, the error rate for this function improved, suggesting the anomaly was network-related rather than inherent to the API itself.
    - Encrypt API for RSA algorithm (no session key) displayed persistent higher error rates under specific user loads and larger payload sizes, which may indicate efficiency constraints or configuration issues specific to this API function. This anomaly warrants further investigation to determine the underlying cause and potential optimizations.

Visualization:

- A line chart showing **"Error Rate by User Load"** and **"Error Rate by Payload Size"** highlights how error rates fluctuate with different loads and payload sizes.

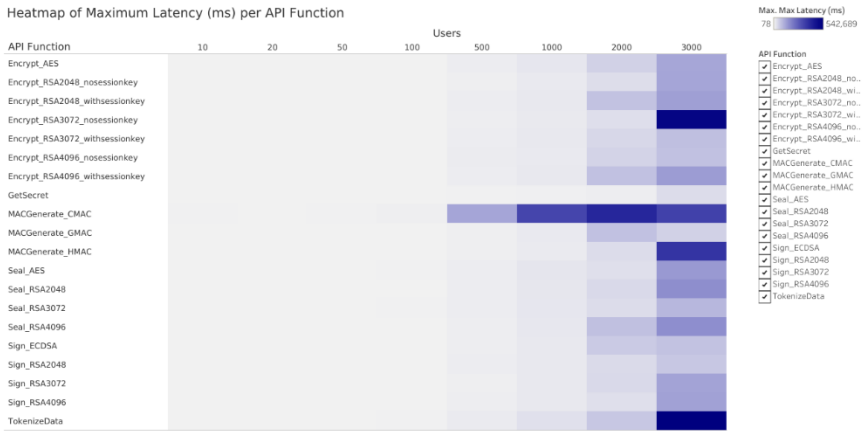


- **Attachments**: [Link to Error Rate Visualizations] for access to the complete error rate analysis charts.

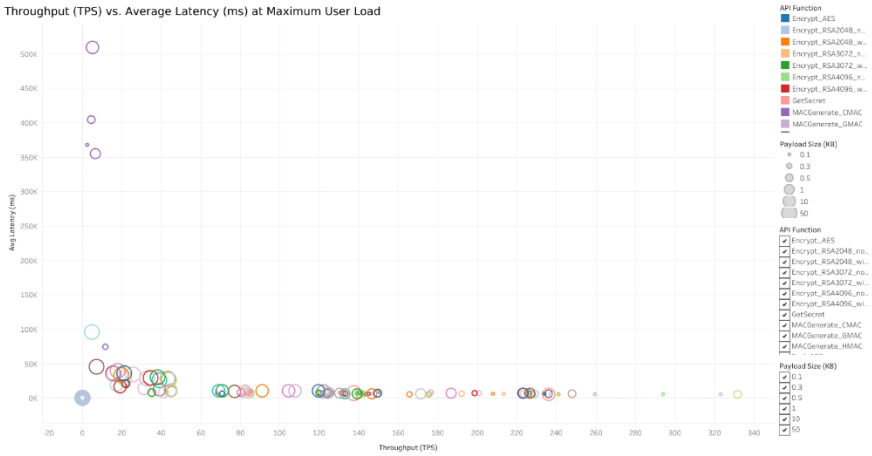
6.4 Additional Analysis

- **Heat map of Maximum Latency per API Operation**: This heat map provides insights into maximum latency for each API operation at various user load levels, allowing

identification of specific functions or algorithms that experience latency increases under certain loads.



- **Throughput vs. Average Latency at Maximum User Load**: This scatter plot examines the relationship between throughput and response time (latency) under maximum user load conditions (3000 users), helping to identify performance bottlenecks where throughput declines or latency increases sharply.



---

7. Optimization Recommendations

- **Code Optimization**: Recommended improvements in algorithm efficiency and process handling to improve API stability.
- **Hardware Scaling**: Based on testing results, consider upgrading CPU, memory, and network bandwidth to better handle high-load conditions.
- **Resource Management**: Implement resource management strategies, such as load balancing or user throttling, to handle peak demands.
- **Attachments**: Checklist of recommended optimization steps.

---

8. Conclusion

- **Load Handling Capability**: The SGKMS API met target load requirements at moderate user loads but requires optimizations to handle higher loads effectively.
- **Max Safe Capacity**: Stability was observed up to approximately 1000 concurrent users with moderate payload sizes.
- **Final Remarks**: Improvements in both code efficiency and hardware scaling are recommended for optimal performance under high-stress conditions.

---

9. Appendices

- **Full Data Set**: The complete dataset, including raw latency, throughput, and error rate data, is available for download via the following link: Download Full Data Set (CSV).
- **Links to Dashboard Visualizations**: Interactive Tableau dashboards for detailed data exploration and analysis are available through the following links:
  - Dashboard 1: Error Rate Analysis
  - Dashboard 2: Heat map of Maximum Latency
  - Dashboard 3: Throughput vs. Latency at Max Load
  - Dashboard 4: Latency Analysis
  - Dashboard 5: Throughput Analysis
- **Attachments**: A table listing all tested API operations, along with their respective algorithms and parameter specifications, is included below:

API Operation	Algorithm	Key Length (if applicable)	Additional Parameters
MAC Generation	CMAC, GMAC-256, HMAC-SHA256	N/A	Hash Algorithm Selection
Encrypt	AES	256-bit	-
	RSA	2048, 3072, 4096-bit	Option for Session Key
Seal Data	AES	256-bit	-
	RSA	2048, 3072, 4096-bit	-
Get Secret	N/A	N/A	Retrieve secure secret
Tokenization	N/A	N/A	Convert sensitive data to token
Sign	ECDSA	N/A	-