DALHOUSIE UNIVERSITY

Special Graduate Topics in Applied Computer Science - CSCI 5901

SUMMER SEMESTER 2023

INSTRUCTOR: Darshana Upadhyay & Sagarika Gosh

Research Project

Group: G5

Authors

Name	Banner ID
Dixit Ghodadara	B00913652
Jainil Sevalia	B00925445
Nishith Gadhiya	B00917966
Shvet Anaghan	B00917946

Faculty Of Computer Science
Dalhousie University

1.Introduction:

Background and Overview:
Secure communication is essential in today's digital age to protect sensitive data while it is being transmitted over networks. Fundamental cryptographic protocols, such as Transport Layer Security (TLS) and Secure Sockets Layer (SSL), guarantee the confidentiality and integrity of data exchanged between clients and servers [1]. These protocols rely on cipher suites and collections of cryptographic algorithms and techniques to establish secure connections. The encryption algorithm, key exchange mechanism, and authentication procedure used during the SSL/TLS handshake are determined by cipher suites.

Advantages of WolfSSL:
WolfSSL, an open-source SSL/TLS library, is a lightweight and reliable solution for securing communications in various applications. WolfSSL Inc. created the library, and is now widely used because of its outstanding performance, small size, and versatility. WolfSSL supports multiple devices, from resource-constrained embedded systems to high-performance servers, focusing on resource efficiency and portability. It is the preferred option for embedded IoT devices, industrial control systems, and other edge devices because of its lightweight design, which makes it excellent for deployment in areas with limited processing power and memory [2].

Use case of Multiple Cipher Suites:
Because WolfSSL supports a variety of cipher suites, developers have the freedom to alter their SSL/TLS settings to meet particular security and performance needs. Because of this customization, applications can establish the ideal balance between security and the computational expense imposed by encryption operations. Developers can select from various cipher suites, each offering a different encryption technique, a different way to exchange keys, and a separate authentication system. Whether prioritizing performance optimization for latency-sensitive applications or enhancing communication security with cryptographic solid algorithms, WolfSSL's diverse cipher options cater to a broad spectrum of use cases. Due to its adaptability is now the chosen SSL/TLS solution in various sectors, from aerospace to automotive.

Purpose of the Report:
This report evaluates and compares the performance and security characteristics of different cipher suites available within the WolfSSL library. Based on their unique use cases and security requirements, the analysis aims to assist developers, system administrators, and security professionals choose the best cipher suites. By thoroughly examining the strengths and weaknesses of each cipher suite, this report provides an informed perspective on choosing cipher suites that strike a balance between security and efficiency.

This also provides a comprehensive understanding of the trade-offs between security and performance for each cipher suite. By evaluating encryption, decryption speeds, and memory usage, stakeholders can make informed decisions aligning cipher suite choices with their specific security requirements and operational constraints. Analysis of security weaknesses and strengths improves overall cybersecurity by assuring the best protection of private information and communications while preserving smooth system performance.

2. Methodology:

Installation of WolfSSL [3]:

To install WolfSSL, we followed the steps below:

1. Launching AWS EC2 Instance:
We started by creating and launching a Linux EC2 instance on AWS, selecting an instance type with sufficient resources, including CPU, memory, and storage, to accommodate both the WolfSSL library and our testing requirements.

2. Obtaining Connection Details:
Once the EC2 instance ran, we obtained its IP address and key pair from the AWS EC2 console. These details were crucial for connecting to the EC2 instance via SSH.

3. Connecting to EC2 Instance:
Using the SSH protocol, we established a connection to the EC2 instance from our local machine, providing the necessary connection details we obtained earlier.

4. Uploading WolfSSL Source Code:
With the SSH connection established, we uploaded the WolfSSL source code zip file from our local machine to the EC2 instance using the SCP (Secure Copy) command.

5. Unzipping WolfSSL Source Code:
On the EC2 instance, we then unzipped the uploaded WolfSSL source code using the following command:

- unzip wolfssl.zip

6. Installing Build Dependencies:
Before proceeding with the build and installation, we ensured that all the essential build dependencies and libraries were present on the EC2 instance. We installed the C/C++ compiler, make utility, and any additional libraries required by WolfSSL. To install GCC, we used the following command:

- sudo yum groupinstall "Development Tools"

7. Configuring WolfSSL Build:
After navigating to the WolfSSL source code directory, we configured the build based on our specific requirements. To include the Camellia and RC4 ciphers, we used the following configuration command with WolfSSL version 3.13.0:

- cd wolfssl-3.13.0
- ./configure --enable-camellia --enable-rc4

Additionally, for our AES and ChaCha cipher requirements, we used WolfSSL version 5.6.3. The configuration command for this version was as follows:

- cd wolfssl-5.6.3
- ./configure

8. Building WolfSSL Library:
We ran the make command with the build configuration set to compile the WolfSSL source code and generate the necessary library files and binaries.

- Make

9. Installing WolfSSL Library:
Once the compilation process was completed, we installed the WolfSSL library on the EC2 instance. This ensured that the library files, headers, and binaries were placed in appropriate locations for easy accessibility.

- sudo make install.

10. Testing WolfSSL Functionality:
Having installed the WolfSSL library, we conducted a series of tests to verify its correct functioning within the Linux environment. We tested its compatibility with our test suite and its ability to establish secure connections using different cipher suites.

11. Compiling and Running Server Test Program:
To verify the secure connection between a client and server, we compiled the server test program with the WolfSSL library using the following commands:

- gcc -o ser ser.c -L/usr/local/lib -lwolfssl
- ./ser

12. Compiling and Running Client Test Program:
Additionally, we compiled the client test program with the WolfSSL library using the following commands:

- gcc -o cli cli.c -L/usr/local/lib -lwolfssl
- ./cli

3. Architecture Diagram of WolfSSL:

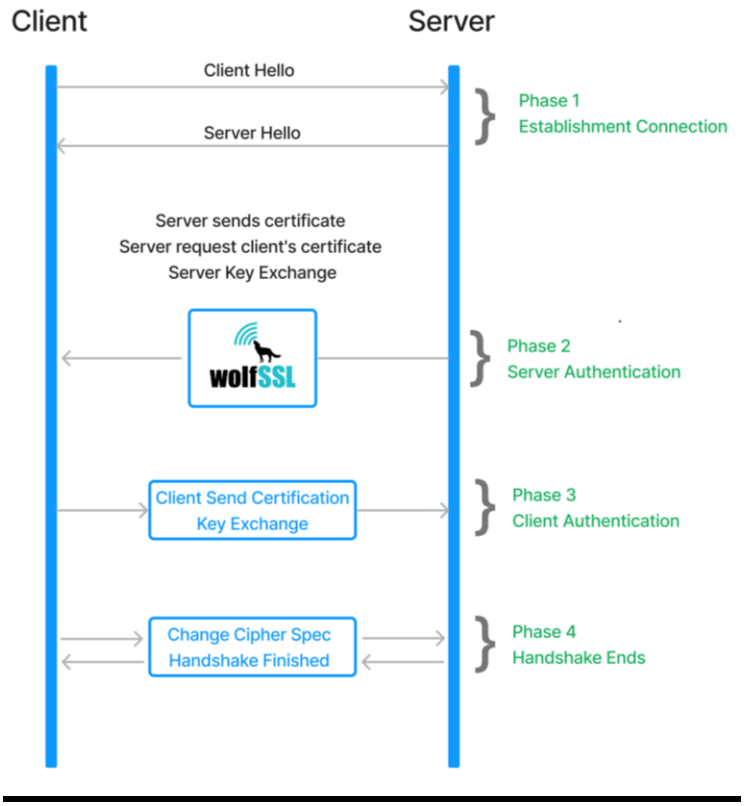

Figure 1: Architecture Diagram of WolfSSL

The above architecture shows the whole process of communication between client and server. First, the client sends a message to the server "Hello". And then in phase 2, the server sends the certificate to the client and also exchanges the key. After validating the server certificate, the client sends the certificate bake to the server and also exchanges the key. Then in the next phase server validate the client certificate and if all thing verified successfully, the client and server can communicate with each other and they use different cipher suites for encryption that provide privacy [4].

4. Cipher Suites:

Cypher suites are collections of cryptographic protocols and algorithms that create secure SSL/TLS connections between a client and a server. They consist of authentication techniques, key exchange protocols, and encryption algorithms. WolfSSL supports several

encryption techniques. Authentication choices include RSA, DSA, ECDSA, PSK, and key exchange techniques, including DH, ECDH, and RSA.

During our evaluation, we tested various cipher suites within the WolfSSL library, including ECDHE-RSA-AES128-SHA256, ECDHE-RSA-AES256-SHA384, CAMELLIA256-SHA256, RC4-SHA, and ChaCha20-Poly1305 [5].

Figure 2: Detais of chiper suite name

List of Cipher Suites Tested:
Below is a short description of every cipher that we tested for analysis.

1. **ECDHE-RSA-AES128-SHA256**: It uses the AES (Advanced Encryption Standard) symmetric encryption algorithm with a 128-bit key and SHA-256 (Secure Hash Algorithm 256-bit) for message authentication.

2. **ECDHE-RSA-AES256-SHA384**: It utilizes AES with a 256-bit key in CBC mode, ECDHE for key exchange, RSA for certificate and key generation, and SHA-384 for message authentication.

3. **DHE-RSA-CAMELLIA256-SHA256:** It is a cipher suite combining Diffie-Hellman Ephemeral for key exchange, RSA for certificate and key generation , SHA256 for authentication, and Camellia-256 for encryption.

4. **ECDHE-RSA-RC4-SHA**: This cypher suite combines Elliptic Curve Diffie-Hellman Ephemeral for key exchange, SHA for authentication, RSA for certificate and key generation and RC4 with a 128-bit key size for encryption.

5. **ECDHE-RSA-CHACHA20-POLY1305**: It uses the ChaCha20 stream cipher for encryption and the Poly1305 message authentication code for message integrity and authenticity. ChaCha20 is designed to be efficient in software implementations and provides excellent security. Poly1305 provides integrity checks for the encrypted data.

5. Description of Each Cipher Suite:

---

1. ECDHE-RSA-AES128-SHA256:

Encryption Algorithm: AES (Advanced Encryption Standard) with a 128-bit key in Cipher Block Chaining (CBC) mode. AES is a widely used symmetric encryption algorithm known

for its strong security and efficiency. Data secrecy is achieved in CBC mode by chaining ciphertext blocks, where each block depends on the one before [6].

Certificate and Key Method: RSA (Rivest-Shamir-Adleman) is used for key exchange, where the client and server generate a shared secret key that is used for secure communication. RSA is a popular asymmetric technique for safe key exchange because of its robust mathematical characteristics [7].

Key Exchange Method: ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) for secure and efficient key exchange.

Authentication Mechanism: SHA-256 (Secure Hash Algorithm 256-bit) is used for message authentication. SHA-256 generates a 256-bit hash value from the data, providing integrity and authentication to the transmitted messages.

2. ECDHE-RSA-AES256-SHA384:

Encryption Algorithm: AES with a 256-bit key in CBC mode. It uses a longer key length for more robust encryption, increasing the channel's security.

Key Exchange Method: ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) is used for key exchange.

Authentication Mechanism: SHA-384 provides enhanced security for message authentication.

3. DHE-RSA-CAMELLIA256-SHA256:

Encryption Algorithm: Camellia-256 is the symmetric encryption algorithm with a 256-bit key size. Camellia is a block cipher known for its strong security and performance, making it suitable for protecting data confidentiality [8].

Key Exchange Method: Diffie-Hellman Ephemeral (DHE) is employed for key exchange, providing forward secrecy. And in this, each DHE session generates a new secret key. Even if a private key is compromised, previous communications are still safe [9].

Certificate and Key Method: RSA is used for server authentication, ensuring the authenticity of the server's identity. During the SSL/TLS handshake, the server's digital signature is verified using its RSA public key.

Authentication Mechanism: SHA-256 is used for message authentication, providing integrity and authentication to the transmitted data.

4. ECDHE-RSA-RC4-SHA:

Encryption Algorithm: RC4 with a 128-bit key is used for symmetric encryption. The use of RC4 in modern cryptographic protocols is not advised due to its current reputation of being weak and attackable.

Key Exchange Method: Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) is employed for key exchange, providing forward secrecy. ECDHE increases the security of the key exchange process by enabling the client and server to create a shared secret key without sharing their private keys.

Certificate and Key Method: RSA is used for server authentication, ensuring the authenticity of the server's identity during SSL handshake.

Authentication Mechanism: SHA is used for message authentication, providing integrity and authentication to the transmitted data.

5. ECDHE-RSA-CHACHA20-POLY1305:

Encryption Algorithm: ChaCha20 stream cipher, is a modern and secure symmetric encryption algorithm known for its high performance and attack resistance.

Key Exchange Method: ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) for secure and efficient key exchange.

Authentication Mechanism: Poly1305 for message authentication, which provides strong integrity protection.

6. Execution Screenshots and Performance Analysis:

For the analysis of the ciphers, we are considering below mentioned 6 factors, and for each cipher suit, we are calculating every factor.

1. Connection Establishment Time:
2. Data Transmission Time:
3. Encryption/Decryption Speed:
4. Memory Usage:
5. Key Size Considerations:
6. Impact on CPU Utilization:

The screenshots of the results of executed program mention below.

1. ECDHE-RSA-AES128-SHA256:

Analysis factors:

1. Connection Establishment Time: 0.000099 Seconds
2. Data Transmission Time: 0.000041 Seconds
3. Encryption/Decryption Speed: 0.000041 Seconds
4. Memory Usage: 307 kb
5. Key Size Considerations: 128 bits
6. Impact on CPU Utilization: 0.003328 Seconds

```
[ec2-user@ip-172-31-87-59 test]$ ./cli
Time taken to establish connection: 0.000099 seconds
Message transmitted for ECDHE-RSA-AES128-SHA256
Time taken for encryption: 0.000041 seconds
Time taken to transmit the message: 0.000041 seconds
Memory Usage: 307 KB
CPU Utilization: 0.003328 seconds
```

Figure 3: Output data of the ECDHE-RSA-AES128-SHA256 Cipher Suite

2. ECDHE-RSA-AES256-SHA384:

Analysis factors:

1. Connection Establishment Time: 0.000285 Seconds
2. Data Transmission Time: 0.000045 Seconds
3. Encryption/Decryption Speed: 0.000045 Seconds
4. Memory Usage: 313 kb
5. Key Size Considerations: 256 bits
6. Impact on CPU Utilization: 0.003884 Seconds

```
[ec2-user@ip-172-31-87-59 test]$ ./cli
Time taken to establish connection: 0.000285 seconds
Message transmitted for ECDHE-RSA-AES256-SHA384
Time taken for encryption: 0.000045 seconds
Time taken to transmit the message: 0.000045 seconds
Memory Usage: 313 KB
CPU Utilization: 0.003884 seconds
```

Figure 4: Output data of the ECDHE-RSA-AES256-SHA384 Cipher Suite

3. DHE-RSA-CAMELLIA256-SHA256:

Analysis factors:

1. Connection Establishment Time: 0.000341 Seconds
2. Data Transmission Time: 0.000046 Seconds
3. Encryption/Decryption Speed: 0.000046 Seconds
4. Memory Usage: 312 kb
5. Key Size Considerations: 256 bits
6. Impact on CPU Utilization: 0.004333 Seconds

```
[ec2-user@ip-172-31-87-59 test]$ ./cli
Time taken to establish connection: 0.000341 seconds
Message transmitted for DHE-RSA-CAMELLIA128-SHA256
Time taken for encryption: 0.000046 seconds
Time taken to transmit the message: 0.000046 seconds
Memory Usage: 312 KB
CPU Utilization: 0.004333 seconds
```

Figure 5: Output data of the DHE-RSA-CAMELLIA256-SHA256 Cipher Suite

4. ECDHE-RSA-RC4-SHA:

Analysis factors:

1. Connection Establishment Time: 0.000372 Seconds
2. Data Transmission Time: 0.000028 Seconds
3. Encryption/Decryption Speed: 0.000028 Seconds
4. Memory Usage: 313 kb
5. Key Size Considerations: 128 bits
6. Impact on CPU Utilization: 0.003956 Seconds

```
[ec2-user@ip-172-31-87-59 test]$ ./cli
Time taken to establish connection: 0.000372 seconds
Message transmitted for ECDHE-RSA-RC4-SHA
Time taken for encryption: 0.000028 seconds
Time taken to transmit the message: 0.000028 seconds
Memory Usage: 313 KB
CPU Utilization: 0.003956 seconds
```

Figure 6: Output data of the ECDHE-RSA-RC4-SHA Cipher Suite

5. ECDHE-RSA-CHACHA20-POLY1305:

Analysis factors:

1. Connection Establishment Time: 0.000314
2. Data Transmission Time: 0.000026
3. Encryption/Decryption Speed: 0.000026
4. Memory Usage: 307 kb
5. Key Size Considerations: 256 bits
6. Impact on CPU Utilization: 0.003306

```
[ec2-user@ip-172-31-87-59 test]$ ./cli
Time taken to establish connection: 0.000314 seconds
Message transmitted for ECDHE-RSA-CHACHA20-POLY1305
Time taken for encryption: 0.000026 seconds
Time taken to transmit the message: 0.000026 seconds
Memory Usage: 307 KB
CPU Utilization: 0.003306 seconds
```

Figure 7: Output data of the ECDHE-RSA-CHACHA20-POLY1305 Cipher Suite

And for the all cipher suites the server communication remains same and below we provided the screenshot for that.

7. Performance Analysis:

After the execution of every cipher suite, we analyzed the performance. The performance analysis of cipher suites in the WolfSSL library aims to assess their efficiency in establishing secure connections and transmitting data over the network. The evaluation involves measuring various performance metrics to understand the impact of different cipher suites on system resources and communication speed. As we have tested the cipher suites, the following are graphs of performance we analyzed during the testing.

1. **Connection Establishment Time:**

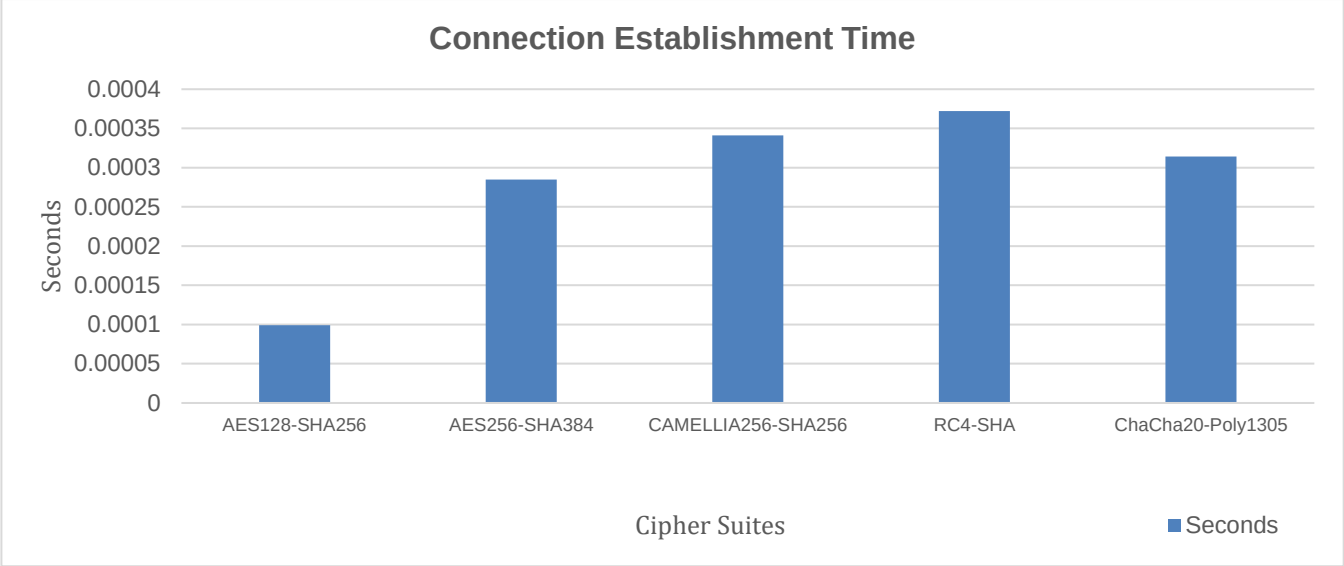

Figure 8: Conection establishment time

2. Data Transmission Time:

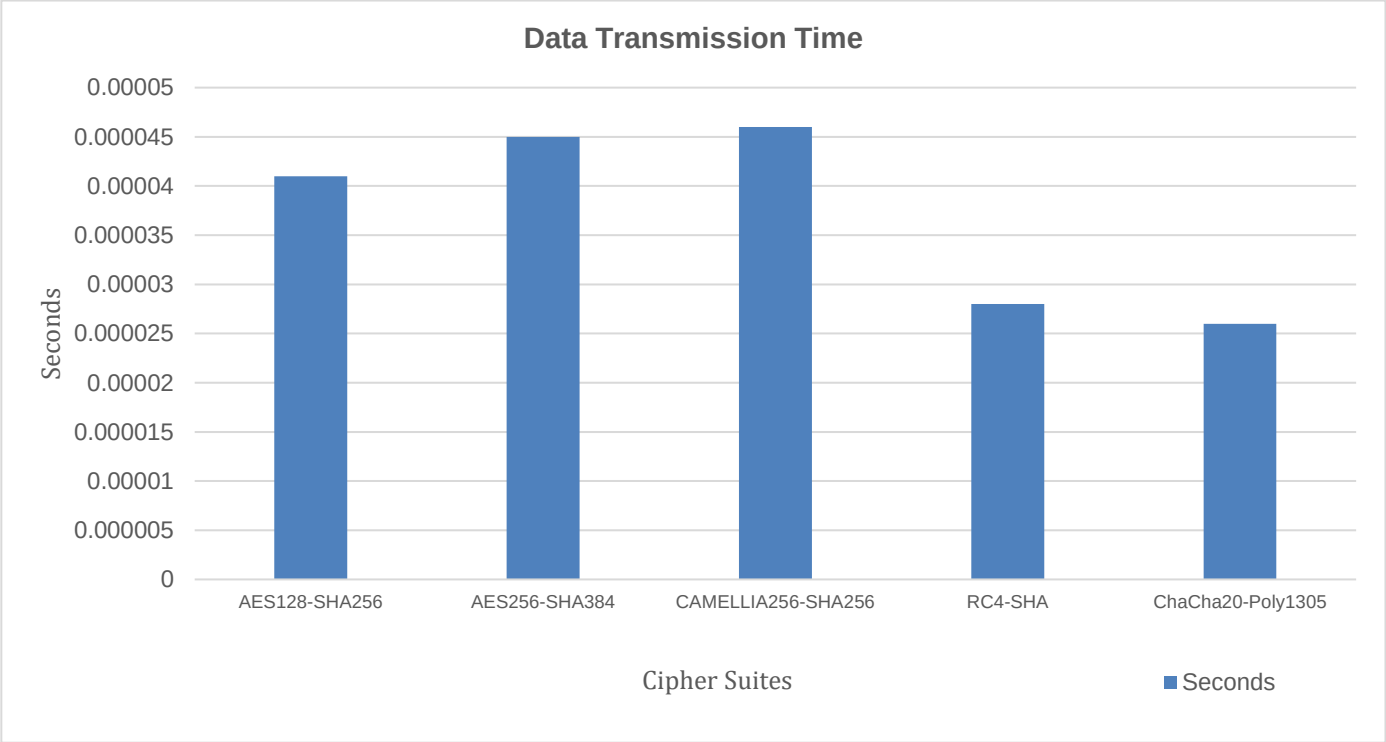

Figure 9: Data transmission time

3. Encryption/Decryption Speed:

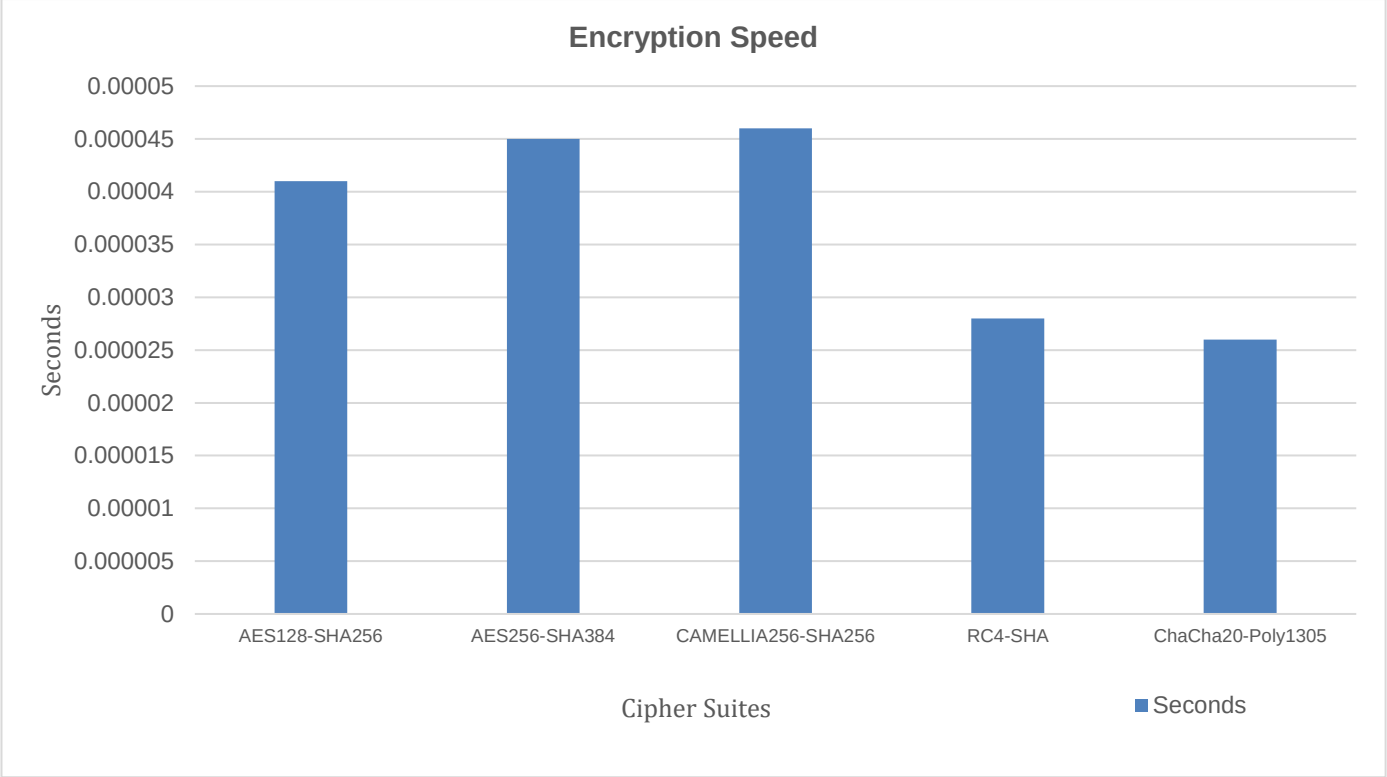

Figure 10: Encryption Speed

4. Memory Usage:

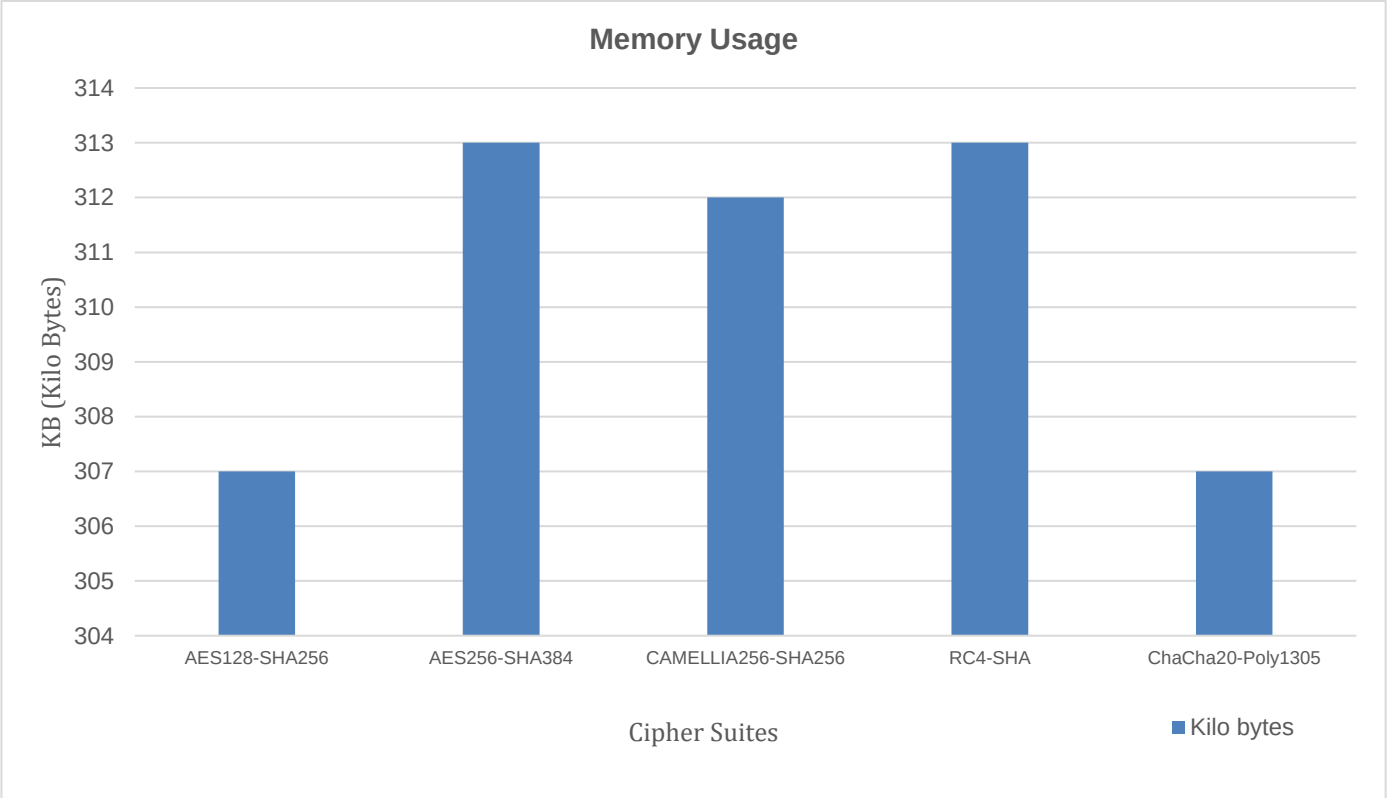

Figure 11: Memory Usage

5. Key Size Considerations:

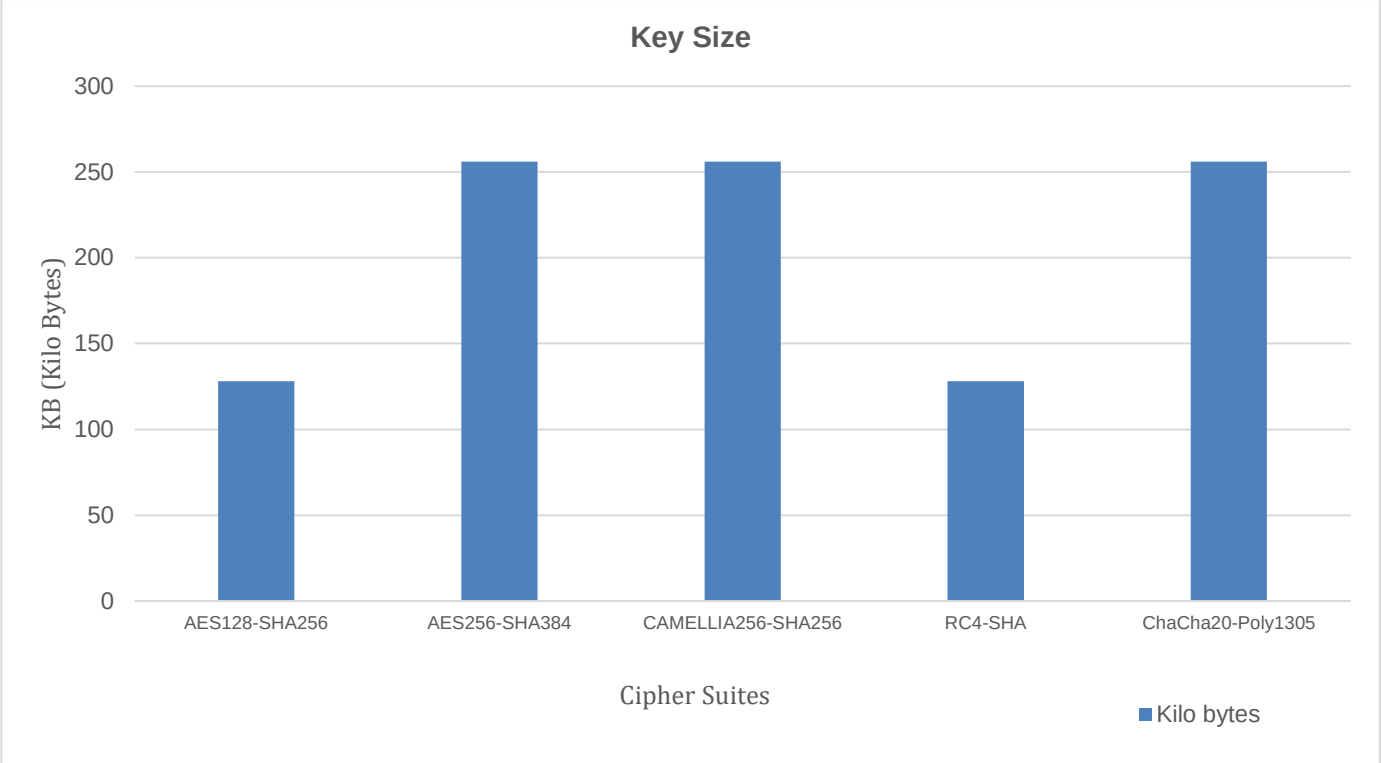

Figure 12: Key size considerations

6. Impact on CPU Utilization:

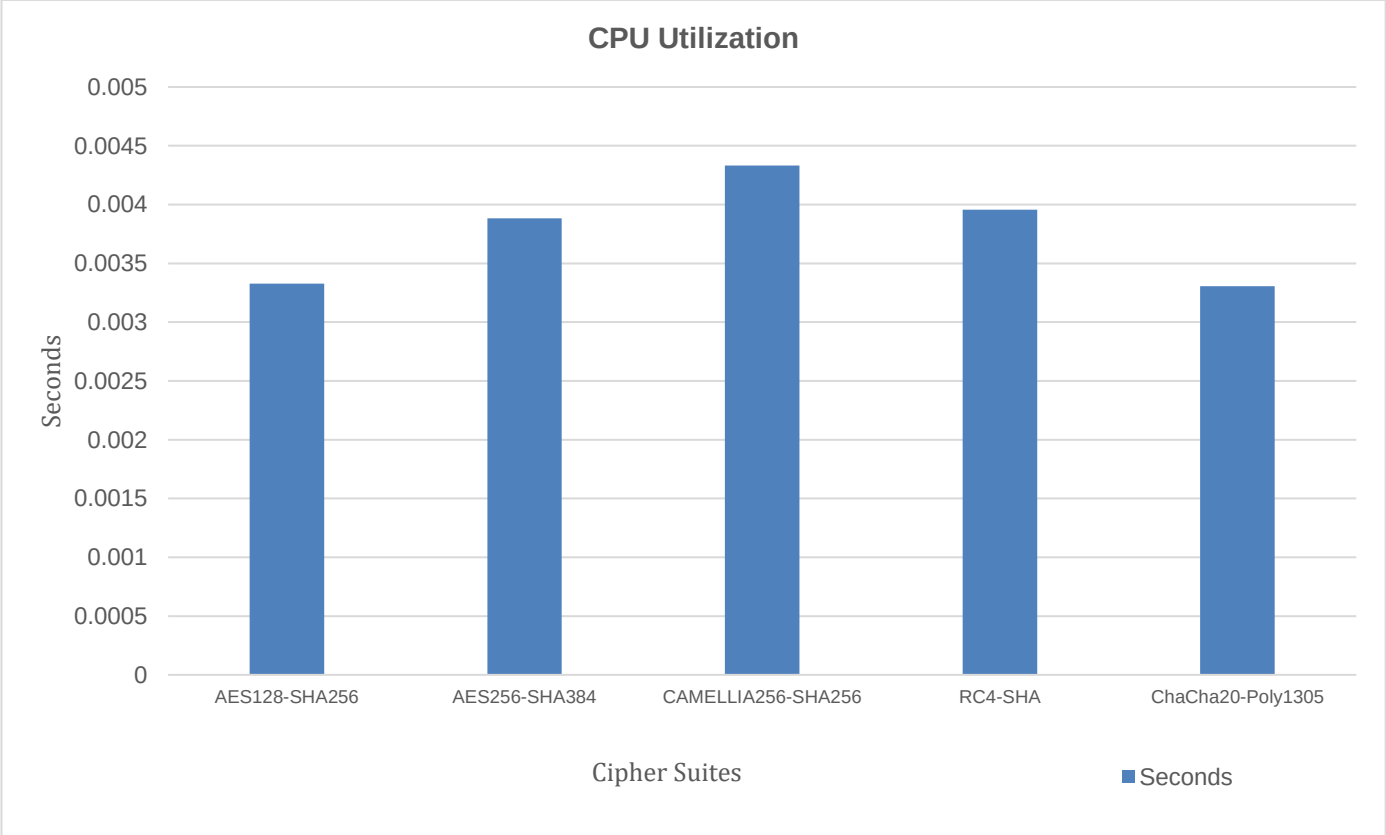

Figure 13: CPU Utilzation

8. Security Evaluation:

1. Strength of Encryption Algorithms [10]:

According to our analysis and research, the strength of the cipher suite is very important for security evaluation. We find the below strengths for every cipher.

- **ECDHE-RSA-AES128-SHA256:**
- ECDHE-RSA-AES128-SHA256 combines AES-128 for high-level security with SHA-256, a robust hashing algorithm known for collision resistance. This cryptographic suite ensures strong encryption, thwarting brute-force attacks, and guarantees data integrity and authenticity in secure communication protocols like TLS/SSL for website encryption.

- **ECDHE-RSA-AES256-SHA384:**
- ECDHE-RSA-AES256-SHA384 combines AES-256 for strong encryption with SHA-384, a collision-resistant hashing algorithm, providing a highly secure cryptographic suite. This combination offers exceptional protection against brute-force attacks and ensures data integrity in secure communication protocols.

- **DHE-RSA-CAMELLIA256-SHA256:**
- The main strength of DHE-RSA-CAMELLIA256-SHA256 lies in its use of strong Camellia-256 encryption, providing robust data confidentiality.

- **ECDHE-RSA-RC4-SHA:**
- The main strength of ECDHE-RSA-RC4-SHA lies in its adoption of forward secrecy through Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) key exchange, enhancing communication security.

- **ChaCha20-Poly1305:**
- ECDHE-RSA-CHACHA20-POLY1305 is a powerful cryptographic suite combining ChaCha20's strong encryption and speed with Poly1305's robust message authentication. It ensures high-level security and data integrity for various applications, with efficient performance.

2. Vulnerabilities and Known Attacks for Each Cipher Suite [10][11]:

- **ECDHE-RSA-AES128-SHA256:** AES with SHA-256 is widely used and has no known vulnerabilities in its standard configuration.

- **ECDHE-RSA-AES256-SHA384:** Similar to ECDHE-RSA-AES128-SHA256, AES with SHA-384 offers strong security without known vulnerabilities.

- **DHE-RSA-CAMELLIA256-SHA256:** While Camellia has no known vulnerabilities, DHE key exchange introduces the possibility of man-in-the-middle

attacks, necessitating proper key management and authentication measures.

- **:** The RC4 algorithm is now considered weak and vulnerable to cryptographic attacks like RC4 biases, rendering ECDHE-RSA-RC4-SHA less secure than other modern cipher suites.

- **ECDHE-RSA-CHACHA20-POLY1305:** ChaCha20 combined with Poly1305 for authentication offers robust security without known vulnerabilities.

3. Resistance to Attacks [11]:

- AES cipher suites (ECDHE-RSA-AES128-SHA256 and ECDHE-RSA-AES256-SHA384) are highly resistant to known cryptographic attacks and provide strong security for data transmission.
- **DHE-RSA-CAMELLIA256-SHA256:** DHE-RSA-CAMELLIA256-SHA256 offers robust resistance to potential attacks, thanks to the secure properties of Camellia-256 encryption and the forward secrecy provided by Diffie-Hellman Ephemeral (DHE) key exchange.

- **ECDHE-RSA-RC4-SHA:** It provides benefits from the forward secrecy provided by Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) key exchange and the strong cryptographic properties of RC4 encryption, contributing to its resilience against potential attacks.

- **ECDHE-RSA-CHACHA20-POLY1305** provides high attack resistance and is a secure choice for modern applications due to its strong security properties.

9. Findings and Results:

Connection Establishment Time:

Among the analyzed encryption algorithms, the connection establishment time plays a crucial role in determining the efficiency and security of the communication. ECDHE-RSA-AES128-SHA256 stands out with the fastest connection establishment time of 0.000099 seconds, making it an excellent choice for applications that require a quick and secure communication setup. This is particularly beneficial for time-sensitive use cases like real-time communication, online gaming, and financial transactions, where reduced latency and prompt responses are critical.

ECDHE-RSA-AES256-SHA384 follows with a still respectable connection establishment time of 0.000285 seconds, providing a higher level of security with its 256-bit key size. ECDHE-RSA-CHACHA20-POLY1305 offers a competitive connection time of 0.000314 seconds, making it suitable for resource-constrained environments without compromising security. However, CAMELLIA256-SHA256 and RC4-SHA exhibit relatively slower connection establishment times at 0.000341 and 0.000372 seconds, respectively, which may

introduce minor delays and potentially render the systems using them more vulnerable to certain types of attacks.

Data Transmission Time:

ECDHE-RSA-CHACHA20-POLY1305 emerged as the fastest in data transmission with only 0.000026 seconds, making it ideal for real-time communication and resource-constrained environments like IoT devices. RC4-SHA follows closely at 0.000028 seconds, also suitable for real-time applications. ECDHE-RSA-AES128-SHA256, ECDHE-RSA-AES256-SHA384, and CAMELLIA256-SHA256, though marginally slower at 0.000041, 0.000045, and 0.000046 seconds respectively, still balance speed and security.

The advantage of shorter data transmission times is reduced latency, better real-time communication, and lower resource utilization. However, longer data transmission times may introduce higher latency and data overhead, impacting the responsiveness of applications. While data transmission time doesn't directly affect algorithm vulnerability, longer times could expose applications to timing attacks.

Encryption/Decryption Speed:

The performance analysis of various encryption algorithms reveals that ECDHE-RSA-CHACHA20-POLY1305 boasts the fastest encryption/decryption speed at 0.000026 seconds, closely followed by RC4-SHA with 0.000028 seconds. ECDHE-RSA-AES128-SHA256, ECDHE-RSA-AES256-SHA384, and CAMELLIA256-SHA256 exhibit slightly slower encryption/decryption speeds of 0.000041, 0.000045, and 0.000046 seconds, respectively. A lower encryption/decryption speed provides advantages in terms of faster data processing, reduced latency, and improved system performance, making it suitable for time-sensitive applications, real-time communication, and high-volume data transmission. ECDHE-RSA-CHACHA20-POLY1305 and RC4-SHA are particularly well-suited for use cases where speed and efficiency are paramount, such as secure messaging, video conferencing, online gaming, and streaming services.

However, faster encryption/decryption may compromise security, as algorithms like RC4-SHA are susceptible to cryptographic vulnerabilities and biases that could lead to potential attacks on encrypted data. The impact on vulnerability depends on the specific use case and the cryptographic algorithm used. For applications prioritizing security, slower but more robust encryption algorithms like ECDHE-RSA-AES128-SHA256, ECDHE-RSA-AES256-SHA384, and CAMELLIA256-SHA256 should be preferred, considering the trade-offs between speed and security to ensure a well-balanced approach to data protection.

Memory Usage:

Comparing the memory usage of each cipher, ECDHE-RSA-AES128-SHA256 and ECDHE-RSA-CHACHA20-POLY1305 is the most efficient, consuming a mere 307 kb each. These algorithms become highly appealing for resource-constrained environments, such as IoT devices, mobile platforms, and embedded systems, where minimizing memory overhead is crucial. Their advantages lie in being resource-friendly, scalable, and potentially faster due to reduced memory demands. On the other hand, ECDHE-RSA-AES256-SHA384,

CAMELLIA256-SHA256, and RC4-SHA exhibit higher memory requirements, reaching 313 kb.

While still practical in many scenarios, their memory usage may pose challenges in memory-limited environments, hindering scalability and increasing infrastructure costs in cloud deployments. Moreover, high memory usage can indirectly impact vulnerability, making systems susceptible to memory-related attacks and potentially hampering timely security updates.

Impact on CPU Utilization:

The comparison of CPU utilization for five ciphers - ECDHE-RSA-AES128-SHA256, ECDHE-RSA-AES256-SHA384, DHE-RSA-CAMELLIA256-SHA256, ECDHE-RSA-RC4-SHA, and ECDHE-RSA-CHACHA20-POLY1305 - reveals that ECDHE-RSA-CHACHA20-POLY1305 has the lowest impact on CPU, taking just 0.003306 seconds. This highly efficient cipher is an excellent choice for applications prioritizing CPU optimization. Conversely, DHE-RSA-CAMELLIA256-SHA256 exhibits slightly higher CPU impact at 0.004333 seconds, which may be a consideration for resource-constrained scenarios. The findings provide valuable guidance in selecting the most suitable cipher based on security requirements and resource optimization goals.

Key Size Considerations:

The comparison of key sizes among the five ciphers shows that ECDHE-RSA-AES128-SHA256 and ECDHE-RSA-RC4-SHA use 128-bit keys, which are reasonably secure but could become vulnerable to attacks as computers get more powerful. On the other hand, ECDHE-RSA-AES256-SHA384, DHE-RSA-CAMELLIA256-SHA256, and ChaCha20-Poly1305 use 256-bit keys, offering much stronger security. Using larger keys makes these ciphers more resistant to potential attacks and ensures better protection for sensitive data. As cyber threats evolve, it's crucial to prioritize ciphers with bigger key sizes to safeguard digital communications and information effectively.

10. Discussion:

The selected cipher suites offer various advantages to cater to diverse security needs. ECDHE-RSA-AES128-SHA256 provides strong security with AES encryption and a 128-bit key, ensuring data confidentiality and message integrity. ECDHE-RSA-AES256-SHA384 enhances security with a longer 256-bit key, suitable for critical information protection. DHE-RSA-CAMELLIA256-SHA256 combines Camellia-256 encryption with DHE key exchange, offering forward secrecy and adaptability. ECDHE-RSA-RC4-SHA utilizes ECDHE key exchange for forward secrecy, making it suitable for legacy systems. ChaCha20-Poly1305, a modern and lightweight suite, ensures high-performance encryption and authentication, ideal for resource-constrained devices and high-performance applications.

Despite their strengths, the selected cipher suites also come with some limitations. ECDHE-RSA-AES128-SHA256, while robust, may not provide the same level of security as ECDHE-RSA-AES256-SHA384 due to its smaller key size. ECDHE-RSA-AES256-SHA384,

although highly secure, may incur higher computational overhead and slower performance compared to ECDHE-RSA-AES128-SHA256. DHE-RSA-CAMELLIA256-SHA256's use of Camellia encryption might lead to interoperability issues with certain systems not supporting this cipher. ECDHE-RSA-RC4-SHA, though enabling forward secrecy, uses the RC4 algorithm, which is now considered weak and susceptible to attacks. ChaCha20-Poly1305, while efficient, may lack compatibility with some legacy systems and software that only support traditional ciphers. Careful consideration of these disadvantages is essential when selecting cipher suites for specific use cases.

Considerations for Selecting a Cipher Suite:

ECDHE-RSA-AES128-SHA256: ECDHE-RSA-AES128-SHA256 offers a balanced approach between security and performance, making it a suitable choice for general applications where strong encryption is required without compromising efficiency.

ECDHE-RSA-AES256-SHA384: When dealing with highly sensitive data requiring maximum security, ECDHE-RSA-AES256-SHA384 becomes the preferred option due to its larger key size, providing stronger encryption.

DHE-RSA-CAMELLIA256-SHA256: DHE-RSA-CAMELLIA256-SHA256 may be chosen for scenarios where Camellia encryption is desired, but careful consideration should be given to compatibility issues, as not all systems may support this cipher.

ECDHE-RSA-RC4-SHA: Avoid using ECDHE-RSA-RC4-SHA due to its reliance on the RC4 algorithm, which is vulnerable to attacks. It is recommended to prioritize forward secrecy and choose more secure ciphers.

ECDHE-RSA-CHACHA20-POLY1305: For applications requiring high-speed encryption and low memory usage, ChaCha20-Poly1305 is an excellent choice, balancing security and performance, especially for modern systems and devices.

11. Conclusion:

In conclusion, analyzing different cipher suites has provided valuable insights into their strengths and weaknesses. The evaluation focused on crucial aspects such as security, performance, and compatibility. Based on the findings, ECDHE-RSA-AES128-SHA256, ECDHE-RSA-AES256-SHA384, and ChaCha20-Poly1305 emerged as robust choices, striking a balance between security and efficiency. However, caution should be exercised with ciphers like DHE-RSA-CAMELLIA256-SHA256 and ECDHE-RSA-RC4-SHA, considering their specific use cases and compatibility requirements. By selecting cipher suites judiciously, organizations can enhance their data security while optimizing communication performance in real-world applications.

12. Recommendation:

The comparison of different ciphers highlights the importance of selecting the appropriate encryption method based on specific application needs. For scenarios prioritizing performance and resource efficiency, ECDHE-RSA-AES128-SHA256 and ECDHE-RSA-RC4-SHA prove to be viable options. ECDHE-RSA-AES128-SHA256 strikes a balance between security and speed with its 128-bit key size, providing a reasonable level of protection while enabling fast connection establishment, data transmission, and encryption/decryption processes. Similarly, ECDHE-RSA-RC4-SHA showcases efficient data handling capabilities. However, it's essential to exercise caution when using this cipher due to its known vulnerabilities in modern cryptography, making it less suitable for applications with stringent security requirements.

Conversely, for applications demanding higher levels of security and protection against potential future threats, ECDHE-RSA-AES256-SHA384, DHE-RSA-CAMELLIA256-SHA256, and ChaCha20-Poly1305 emerge as more robust contenders. These ciphers utilize 256-bit keys, significantly enhancing cryptographic strength and resilience against brute-force attacks. ECDHE-RSA-AES256-SHA384 excels in efficient data handling while maintaining secure communication, and both DHE-RSA-CAMELLIA256-SHA256 and ChaCha20-Poly1305 exhibit respectable performance metrics. Among them, ChaCha20-Poly1305 stands out with the lowest impact on CPU utilization, making it an appealing choice for scenarios where reducing CPU overhead is a critical consideration.

13. References:

[1] “What is SSL, TLS and HTTPS? | DigiCert,” *Digicert.com*, Jun. 07, 2023. Available: https://www.digicert.com/what-is-ssl-tls-and-https . [Accessed: Jul. 31, 2023]

[2] wolfSSL Embedded SSL/TLS Library, “wolfSSL Embedded SSL/TLS Library – wolfSSL,” *Wolfssl.com*, 2023. Available: https://www.wolfssl.com/avionics-mag/#:~:text=wolfSSL%20supports%20industry%20standards%20up,cryptography%20library%2C%20and%20much%20more [Accessed: Jul. 31, 2023]

[3] Q. Guide, “wolfSSL Quickstart Guide | Documentation – wolfSSL,” Wolfssl.com, 2023. Available: https://www.wolfssl.com/docs/quickstart/ [Accessed: Jul. 31, 2023]

[4] “D. SSL/TLS Overview - wolfSSL Manual,” Wolfssl.com, 2018. Available: https://www.wolfssl.com/documentation/manuals/wolfssl/appendix04.html [Accessed: Jul. 31, 2023]

[5] “4. Features - wolfSSL Manual,” Wolfssl.com, 2023. Available: https://www.wolfssl.com/documentation/manuals/wolfssl/chapter04.html#:~:text=a%20supported%20suite.-,Block%20and%20Stream%20Ciphers,RC4%20are%20enabled%20by%20default [Accessed: Jul. 31, 2023]

[6] “Advanced Encryption Standard AES,” GeeksforGeeks, Oct. 15, 2021. Available: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ [Accessed: Jul. 31, 2023]

[7] A. Bhattacharya, "What is RSA? How does an RSA work?," Encryption Consulting, Sep. 23, 2020. Available: https://www.encryptionconsulting.com/education-center/what-is-rsa [Accessed: Jul. 31, 2023]

[8] Hans Christian Rudolph, "Ciphersuite Info," Ciphersuite.info, 2013. Available: https://ciphersuite.info/cs/TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA/ [Accessed: Jul. 31, 2023]

[9] "Why use Ephemeral Diffie-Hellman — Mbed TLS documentation," Readthedocs.io, 2023. Available: https://mbed-tls.readthedocs.io/en/latest/kb/cryptography/ephemeral-diffie-hellman/#:~:text=When%20a%20key%20exchange%20uses,past%20communication%20is%20still%20secure [Accessed: Jul. 31, 2023]

[10] Hans Christian Rudolph, "Ciphersuite Info," Ciphersuite.info, 2017. Available: https://ciphersuite.info/cs [Accessed: Jul. 31, 2023]

[11] "CVE security vulnerability database. Security vulnerabilities, exploits, references and more," Cvedetails.com, 2013. Available: https://www.cvedetails.com/ [Accessed: Jul. 31, 2023]