Volume 66, Issue 2, 2022

Journal of Scientific Research

of

The Banaras Hindu University

Performance Analysis of SSL/TLS Crypto Libraries: Based on Operating Platform

Suresh Prasad Kannojia[1[0000-0001-5439-4269]] and Jitendra Kurmi[2[0000-0002-0804-4556]]

[1] Lucknow University, Lucknow, Uttar Pradesh, India - 226007
[2] Lucknow University, Lucknow, Uttar Pradesh, India - 226007
spkannojia@gmail.com
jitendrakurmi458@gmail.com

Abstract. Security in Computer Network Communication is of great importance because unauthorized users attempt to steal, modify, misuse, interrupt, and try to un-stabilize, smartly our network systems. Therefore up to some extent, the secure communication provided by Transport Layer Protocol, implementation of the TLS function, and distinct libraries were designed by researchers, of which each library has the broad support of the encryption algorithms. But security can be compromised and seen in an offensive maneuver of the digital world as the main challenge in communication. In this paper, performance analysis of the most authentic six libraries: OpenSSL, AWS s2n, GnuTLS, NSS, BoringSSL, and Cryptlib performed to find appropriate TLS libraries for uncompromised communication based on throughput, CPU usage in the different virtual operating environments.

Keywords: TLS, OpenSSL, GnuTLS, Performance.

1 Introduction

Security in communication is essential to facilitate reliability, data integrity, and confidentiality. Transport Layer Security can provide these aspects for secure connection over computer networks. To prevent the eavesdropping and tampering of data, in various services such as Email, Voice over Internet Protocol, Web browsing, and bank transaction, where uses a set of security algorithms like the key exchange algorithm, hash function, and public-key cryptography, to meets the increasing demand for enhanced security over the years. Different versions of TLS have been designed and developed by researchers as per the requirement of secure communication.

According to the Internet Engineering Task Force, using cipher suites older than TLS 1.0 for secure communication is ineffective though some browsers warn you if a site uses an older version. The widely used Google Chrome and Mozilla Firefox browsers support the implementation of TLS1.3, widely used by users. In the case of connectionless applications, they use the datagram transport layer security. Although DTLS is similar to TLS, with the exception that DTLS must deal with packet loss and reordering issues. Three characteristics of DTLS implementation are as follows:

1) Packet Retransmission – The damaged and lost data packets are retransmitted.
2) Sequencing of Packets – To reordering damage and lost data packets, a sequence number is assigned.
3) Replay Detection – Reply detection is used to avoid duplicate packets and discard-ing old received packets.

In this paper, we have took six well known TLS libraries for performance analysis based on the throughput and CPU use up over five different operating platforms.

2 Review of Literature

TLS supports connectionless services, to deal with issues of packet loss and reordering for data packets [1]. Computer aided cryptography plays a crucial role in the standardization processes where proses, formulas, and pseudo code are wants to write cryptographic standards with clarity, simple implementation, and ability [2]. The most effective standard way to analyze the massive cryptographic systems completed is by composing less complicated building blocks. Numerous cryptographic researchers found that preserving the safety underneath composition is tough Universal composability is employed for analyzing large cryptographic systems by most game based security definitions [3]. The function correctness proofs prefer by automating equivalence to solve the sequence of simple transformations. However, most function correctness proofs are not automatic in proving the functional correctness and simple transformation [4]. Furthermore, computer-aided cryptography using the machine checkable approach to design, analyze, and implement has developed and to evaluate the accuracy, scope, trustworthiness, usability of state-of-the-art tools, and their research difficulties the taxonomy was created [5].

In literature, an open source TLS library Network Security Service supports cross platform server side and hardware, in 1997, the smart cards on the client side advance by Netscape [6]. Further, in 1998 Eric Andrew Young and Tim Hudson established an Open-Source framework OpenSSL for secure communication over a computer network [7]. In 2003 Nikos Mavrogiannopoulos created the GnuTLS as a library that allows client applications to provide a secure connection over a computer communication network using suitable protocols [8]. In 2003 Cryptlib was created by Gutmann as an open source security toolkit that supports multiple crypto graphical libraries for implementing secure sessions in SSL/TLS [9]. In 2014 Google built and developed Boring SSL to meet their demands,

DOI: 10.37398/JSR.2022.660212

and it supports a variety of cipher suite algorithms for secure communication [10]. In 2015, Amazon Web Services Developed AWS s2n services as an open source library that supports different cryptographic algorithms to implement SSL/TLS [11]. Whereas EverCypt has developed by the Everest project, a C/x86 cryptography library [12-15]. The performance and recent findings are outstanding, well optimized as the OpenSSL program. It follows distinct concepts where the library and proof are co-designed abnormal situations code synthesized. Some handwritten libraries such as AWS-LC, BoringSSL, and OpenSSL could be replaced by the above as per approach.

In literature, the CASM toolchain uses SMT solvers and symbolic execution. It only looks at functions above message blocks even it does not check the most optimized variants of this algorithm. The SHA-256 is analyzed and verified in the CASM project [16].
However, for algorithms proven Fiat Crypto was not applied [17]. The high level specifications were uses to build portable C field arithmetic implementations, the Fiat Crypto's code has already been incorporated in OpenSSL [18]. After this, an integration method vectorized in x86 implementations with acceptable performance originated by Jasmin [19]. Jasmin's implementation of ChaCha20-Poly1305 performs much better than other hand optimized implementations, whereas SHA-256 and AES-GCM are not to discharge in Jasmin [20].
A bug found related to Nettle signature verification functions such as GOST DSA, EDDSA, and ECDSA have been found in the GnuTLS library to call the Elliptic curve cryptography, point multiply function with out-of-range scalars, potentially resulting in incorrect results in GnuTLS versions before 3.7.2. An attacker can use this flaw to force an invalid signature, resulting in an assertion failure or possible validation failure. Confidentiality, integrity, and system availability are the most serious threats posed by this vulnerability [21]. GnuTLS will fix bugs in the versions.
From the literature, it's clear that none of the TLS libraries listed above can guarantee secure communication in all circumstances. It encourages me to analyze the performance of these TLS libraries and find a better one that meets our needs..

3 Methodology

Here, six well known libraries like OpenSSL, BoringSSL, GnuTLS, NSS, AWS s2n, and Cryptlib have been taken for performance analysis, based on throughput, CPU usage for secure communication on the different operating systems in the virtual environ-ment, to find the most appropriate TLS libraries based on performance with mini-mum system requirements.

4 Experimental Setup

The performance analysis of different SSL/TLS open-source libraries and the evaluation based on publicly available documentation. The RFCs for TLS have considered the authoritative source for evaluation, if a particular library confirms the TLS standard or not, with minimum system requirements. Set of performance tests performed against a set of test data on a reference system with Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz, RAM 8GB, and 25GB Hard Disk on the virtual machine with the different operating system.

5 Experimental Results

The experiment has been performed over various operating systems such as Ubuntu, Fedora, Debian-etch, Windows, Mac for six open-source libraries such as OpenSSL, BoringSSL, Cryptlib, AWS s2n TLS, GnuTLS, NSS, and obtained results were tabulated in Table 1. The performance analysis based on the throughput of each cipher suite described in sub Section 5.1, Performance analysis based on the CPU usage for TLS Library described in sub Section 5.2

5.1 Performance analysis based on the throughput of each cipher suite

Speed test with Key Exchange Mechanism (Asymmetric Ciphers). We have experimented on five different Operating Systems out of which three operating systems are from Linux (Ubuntu, fedora & Debian-etch) and the remaining two are windows, mac to reveal details about the throughput of each library. The throughput is calculated in terms of sign and verified per unit time that is bytes/second. Each speed test consists of one sign pass directly followed by a verify pass. The key exchange cipher suites of each library are as follows:

- OpenSSL - RSA, DHE-RSA, DHE-DSS, ECDH-ECDSA, ECDHE-ECDSA, ECDH-RSA, ECDHE-RSA, GOST 28147-89
- GnuTLS - RSA, DHE-RSA, DHE-DSS, ECDHE-ECDSA, ECDHE-RSA
- BoringSSL- RSA, DHE-RSA, DHE-DSS, ECDHE-ECDSA, ECDHE-RSA
- AWS s2n - RSA, DHE-RSA, ECDHE-RSA, ECDHE-ECDSA
- NSS - RSA, DHE-RSA, DHE-DSS, ECDH-ECDSA, ECDHE-ECDSA,ECDH-RSA,ECDHE-RSA, GOST 28147-89
- Cryptlib - RSA, DHE-RSA, DHE-DSS, ECDHE-ECDSA

Here the libraries GnuTLS and OpenSSL Key exchange mechanism ciphers Sign/s and verify/s on Ubuntu operating system has been implemented obtained experimental results tabulated in Table 1, and Table 2.

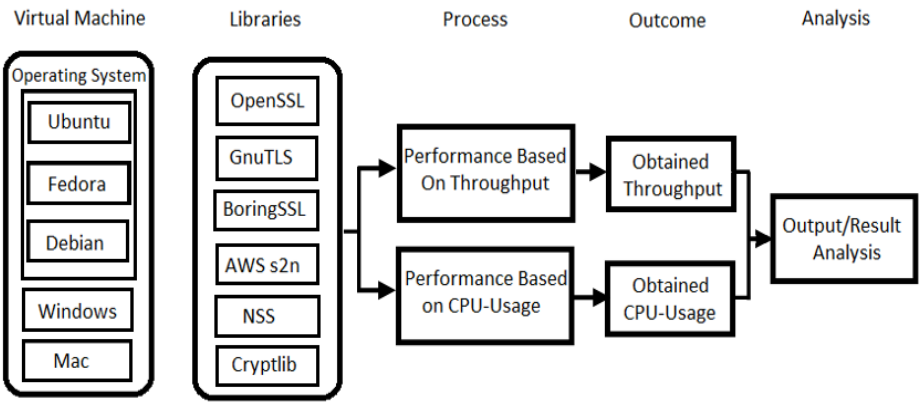

Fig. 1. Proposed Methodology for performance analysis of libraries with distinct Operating Systems

Table 1. OpenSSL Library, Key Exchange Mechanism Ciphers Sign/s and Verify/s with Ubuntu

Operating System	Ciphers	Total Number of Input Buffer Size										Average		Total Average	
		1024 bits / 160 bits		2048 bits / 224 bits		3072 bits / 256 bits		7680 bits / 384 bits		15360 bits / 521 bits					
Ubuntu		S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s
OpenSSL	RSA	1233	25899	1380	26470	1474	26912	1487	27104	1492	27835	**1413.2**	**26844**	**1347**	**26754**
	DHE-RSA	1209	25961	1296	26645	1351	26697	1389	27284	1497	27805	**1348.4**	**26878.4**		
	DHE-DSS	1201	25623	1317	26470	1334	26759	1383	27340	1485	27785	**1344**	**26795.4**		
	ECDH-ECDSA	1190	25329	1232	26165	1294	26310	1327	27467	1342	27531	**1277**	**26560.4**		
	ECDHE-ECDSA	1220	25589	1262	26554	1314	26402	1371	26567	1387	27536	**1310.8**	**26529.6**		
	ECDH-RSA	1304	26098	1279	26459	1324	26422	1367	26670	1454	27643	**1345.6**	**26658.4**		
	ECDHE-RSA	1334	26214	1306	26770	1384	26712	1379	26964	1492	27735	**1379**	**26879**		
	GOST-28147-	1239	25599	1316	26890	1344	26923	1389	27064	1502	27958	**1358**	**26886.8**		

Table 2. GnuTLS Library, Key Exchange Mechanism Ciphers Sign/s and Verify/s with Ubuntu

Operating System	Ciphers	Total Number of Input Buffer Size										Average		Total Average	
		1024 bits / 160 bits		2048 bits / 224 bits		3072 bits / 256 bits		7680 bits / 384 bits		15360 bits / 521 bits					
Ubuntu		S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s
GnuTLS	RSA	1303	24989	1480	25470	1574	25912	1647	26804	1792	27435	**1559.2**	**26122**	**1432**	**26233**
	DHE-RSA	1332	25361	1406	25645	1441	26097	1604	26484	1667	26805	**1490**	**26078.4**		
	DHE-DSS	1301	25223	1377	25870	1404	26159	1488	26340	1535	26785	**1421**	**26075.4**		
	ECDHE-ECDSA	1220	25589	1262	26554	1314	26402	1371	26567	1387	26836	**1310.8**	**26389.6**		
	ECDHE-RSA	1334	26214	1306	26770	1384	26215	1379	26564	1492	26735	**1379**	**26499.6**		

Table 3. Sign/s and Verify/s Comparison for Key Exchange Mechanism Ciphersuites of TLS Libraries

Operating System	OpenSSL		GunTLS		BoringSSL		AWS s2n		NSS		Cryptlib	
	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s	S/s	V/s
Ubuntu	1347	26754	1432	27233	1367	27790	1501	27802	1490	27870	1495	27895
Fedora	1356	26756	1440	27237	1373	27787	1507	27810	1493	27877	1491	27899
Debian-etch	1354	26759	1439	27233	1371	27789	1503	27809	1497	27874	1489	27904
Avg(linux)	1352	26756	1437	27234	1370	27788	1503	27809	1493	27873	1491	27899
Windows	1348	26763	1434	27232	1368	27786	1510	27808	1498	27878	1497	27906
Mac	1347	26769	1433	27233	1368	27788	1504	27807	1495	27879	1494	27901
Average	**1350**	**26760**	**1435**	**27233**	**1369**	**27788**	**1505**	**27807**	**1494**	**27815**	**1493**	**27901**

The above Table 3 has been prepared by adding the throughput in terms of S/s (sign/s) and V/s (verify/s) of cipher suites of each library per distribution after that average per library is computed and a bar chart has been prepared and presented as fig.2.

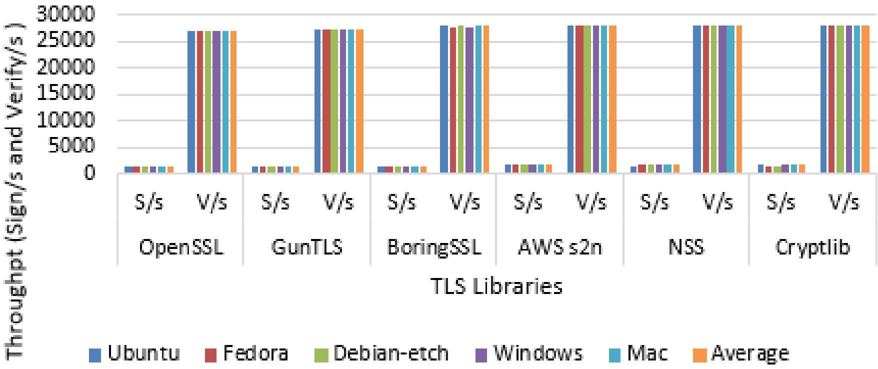

Fig. 2. Comparison of Sign/s and verify/s of Key Exchange Mechanism cipher suites of TLS libraries on different Operating System

Table 4. Boring SSL Library, Hashing Algorithm Ciphers, Throughput (KB/s) with Windows

Operating System	**Ciphers**	**Total Number of Input Buffer Size**					**Average**	**Total Average**
Windows		**16 bytes/s**	**64 bytes/s**	**256 bytes/s**	**1024 bytes/s**	**8192 Bytes/s**		
Boring SSL	HMAC(MD5)	6826.56	11719.	14428.16	17211.69	21161.41	**14269.4**	**14567.01**
	HMAC-SHA1	4781.35	12920.15	17515.97	19469.15	21148.46	**15167.02**	
	HMAC-SHA256	4420.11	10641.73	16433.64	19783.17	20786.27	**14412.98**	
	HMAC-SHA384	2746.43	10954.58	17531.5	19271.62	21589.13	**14418.65**	

Table 5. AWS s2n Library, Hashing Algorithm Ciphers, Throughput (KB/s) with Windows

Operating System	**Total Number of Input Buffer Size**						**Average**	**Total Average**
Windows	**Ciphers**	**16 bytes**	**64 bytes**	**256 bytes**	**1024 bytes**	**8192 bytes**		
AWS s2n	HMAC(MD5)	6826.56	11719.17	14428.16	18211.69	24663.41	**15169.8**	**15084.06**
	HMAC-SHA1	4781.35	11920.15	16515.97	19369.15	23860.46	**15289.42**	
	HMAC-SHA256	4420.11	10641.73	16433.64	19183.17	23851.27	**14905.98**	
	HMAC-SHA384	2746.43	10954.58	17531.5	18731.62	24891.13	**14971.05**	

Similarly, we can obtain the Total Number of Input Buffer Size of Sign/s and Verify/s of the key exchange mechanisms for libraries such as Boring SSL, AWS s2n, NSS, and Cryptlib with remaining four operating systems such as Fedora, Debian-etch, Windows, and Mac.

First, we have to calculate the average of throughput Sign/s and Verify/s of each cipher suite using equation 1, then total average throughput Sign/s and Verify/s of each library were calculated using equation 2 and tabulated in Table 1 and Table 2.

$$X=\sum_{l=1}^{n}\frac{ThKEMCiphers(m)}{(Total\ Numberof\ input\ Buffer\ size)} \quad (1)$$

$$T_{AVG}Th_{OS(i)\_TLSLibraries(j)} = \sum_{k=1}^{n}\frac{X}{(Total\ number\ of\ Operating\ System)} \quad (2)$$

WhereX= Average Throughput of TLS Libraries (for Sign/s, verify/s)

Tavg Thos_TLSLibraries = Total Average Throughput of TLS Libraries (for Sign/s, verify/s) with Operating system

ThKEM Ciphers = Throughput of Key Exchange Mechanism Ciphers
i = Ubuntu, Fedora, Debian-etch, Windows, and Mac operating systems
j = OpenSSL, GnuTLS, BoringSSL, AWS s2n, NSS and Cryptlib
k=1 & l=1
m= KEM Ciphers such as RSA, DHE-RSA ……in each library
n= Total number of Key Exchange Mechanism in each library

From Table 3 and fig. 2, it is clear that the sign/s of NSS is less than AWS s2n but higher than Cryptlib, whereas NSS verify/s is higher throughput than AWS s2n, BoringSSL, GnuTLS, and OpenSSL on Linux machine and Mac machines. The sign/s throughput of GnuTLS has higher than OpenSSL, BoringSSL but less than AWS s2n, NSS, and Cryptlib, whereas verify/s throughput of OpenSSL has higher on Mac than Windows and Linux in GnuTLS. As implementation results will scale up the throughput due to optimized implementation on new Operating Systems and better processors, OpenSSL will still provide high throughput.

Observation: There are various issues regarding the methods used for the test conducted in the research.

- We can observe that the cipher suites for the key exchange mechanism tested for each library are not the same. It could lead to result in shifting the throughput of the cipher in each library.
- Now results are obtained with varying the buffer size, containing data for each cipher suits only once, then the average is computed with different buffer sizes. Multiple results with the same buffer would have produced the exact measurement then the average is taken.

Speed test with Comparison of Hash Algorithms (Message Authentication Code).

We have experimented on five different operating systems, of which three operating systems are from Linux (Ubuntu, fedora & Debian-etch) remaining two are windows, mac to reveal details about the throughput of each library. The throughput computed, data processed per unit time that is bytes/second. The data collected is from five different operating systems to reveal details about the throughput of each library. Each speed test consists of one encryption pass directly followed by a decryption pass. The Ciphers tested in each library are as follows:

- OpenSSL - HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384, AEAD, GOST 28147-89, GOST R 34.11-94
- GnuTLS - HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384
- BoringSSL - HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384
- s2n - HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384
- NSS - HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384
- Cryptlib - HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384

In this section, the libraries BoringSSL and AWS s2n, hashing algorithms ciphers throughput with windows operating system have been implemented and experimental results tabulated in Table 4 and Table 5.

Similarly, as per the result of Table 4 and Table 5, we can obtain the throughput (KB/s) of Hashing Algorithms for libraries such as OpenSSL, GnuTLS, NSS, and Cryptlib on remaining four operating systems such as Ubuntu, Fedora, Debian-etch, and Mac. The average throughput (KB/s) of every library is tabulated in Table 4 and Table 5. Then total average throughput (KB/s) of each library is tabulated in Table 6, derived from Table 4 and Table 5 using the following formulae represented in equations (3) and (4):

$$X = \sum_{l=1}^{n} \frac{ThHashingAlgorithmCiphers(m)}{(Total\ Numberof\ input\ Buffer\ size)} \quad (3)$$

$$T_{AVG}Th_{OS(i)\_TLSLibraries(j)} = \sum_{k=1}^{n} \frac{X}{(Total\ number\ of\ Operating\ System)} \quad (4)$$

Where X= Average Throughput of TLS Libraries

And $T_{AVG}TH_{OS\_TLSLibraries}$ = Total Average Throughput of TLS Libraries
ThHashingAlgorithm Ciphers = Throughput of Hashing Algorithm Ciphers
i = Ubuntu, Fedora, Debian-etch, Windows, and Mac operating systems
j = OpenSSL, GnuTLS, BoringSSL, AWS s2n, NSS and Cryptlib
k=1 & l=1
m= Hash Algorithms Ciphers such as HMAC (MD5), HMAC-SHA1…….in each library
n= Total number of Hash Algorithms in each library

Table 6. Throughput (KB/s) Comparison of Hashing Algorithms for TLS Libraries different with operating system

Operating System	OpenSSL	GunTLS	BoringSSL	AWS s2n	NSS	Cryptlib
Ubuntu	15,756	15,152	14,529	15,192	14,892	14,374
Fedora	15,640	15,410	14,557	15,140	14,877	14,365
Debian-etch	15,776	15,797	14,565	14,899	14,890	14,388
Windows	15,684	15,765	14,567	15,084	14,886	14,378
Mac	15,699	15,269	14,566	15,061	14,879	14,377
Average	**15,711**	**15,478.6**	**14,556.8**	**15,075.2**	**14,884.8**	**14,376.4**

The above Table 6 has been prepared by adding the throughput of ciphers of each library per distribution. After that average per library is calculated and a bar chart has been presented as Fig. 3.

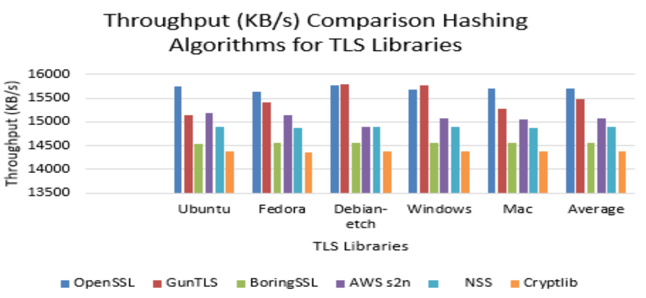

Fig. 3. Throughput (KB/s) comparison of Hashing Algorithms of TLS libraries with different operating system

Fig. 3 and Table 6, clearly show that the OpenSSL has higher throughput for the operating system as Ubuntu, Fedora, and Mac as compare to TLS libraries: BoringSSL, Gnu TLS, AWS s2n, NSS, and Cryptlib, whereas OpenSSL throughput is low on Debian and windows as compare to Gnu TLS. The boringSSL and Cryptlib having low throughput among all six TLS libraries in each operating system. The Library AWS s2n has better throughput as compared to Boring SSL, NSS, and Cryptlib libraries. The NSS has higher throughput as compared to Boring SSL and cryptlib libraries. As the implementation results will scale up the throughput due to optimized implementation on new Operating Systems (OS) and better processors, so OpenSSL will still provide high throughput.

***Observation*:** There are some issues regarding the methods used for the test conducted in the re-search.

- We can observe that the hash algorithms ciphers tested for each of the libraries are not the same. It could result in shifting the throughput of ciphers in each library.
- When buffer size containing data varies for each cipher only once. Multiple results with the same buffer would have produced the exact measurement then the average is taken.

Speed test and Comparisons of Symmetric Ciphers.

Experiment using symmetric ciphers found in well-known open source cryptography libraries, out of this OpenSSL, GnuTLS, BoringSSL, s2n, NSS, and Cryptlib chosen. The throughput is calculated in terms of data processed per unit time that is bytes/second, to reveal details about the throughput of each library. Each speed test consists of one encryption pass directly followed by a decryption pass. The Ciphers tested in each library as:

- OpenSSL - AES GCM, AES CCM, AES CBC, Camellia CBC, ARIA GCM, SEED CBC, 3DES, GOST 28147-89, ChaCha20-Poly1305
- GnuTLS - AES GCM, AES CCM, AES CBC, Camellia GCM, Camellia CBC, , 3DES, ChaCha20-Poly1305
- BoringSSL - AES GCM, AES CBC, 3DES, Chacha20-Poly1305
- s2n - AES GCM, AES CBC, 3DES, ChaCha20-Poly1305
- NSS - AES GCM, AES CBC, Camellia CBC, SEED CBC, 3DES, Chacha20-Poly1305
- Cryptlib - AES GCM, AES CBC, 3DES

In this section, the NSS and Cryptlib, symmetric ciphers, throughput with Mac operating system have been performed, and experimental results tabulated in Table 7 and Table 8.

Similarly, as per the result of Table 7 and Table 8, we can obtain the throughput (KB/s) of Symmetric ciphers for libraries such as OpenSSL, GnuTLS, BoringSSL, and AWS s2n on remaining four operating systems such as Ubuntu, Fedora, Debian-etch, and Windows. The average throughput (KB/s) of each library are computed, and the total average throughput (KB/s) of each library is tabulated in Table 9, derived from Table 7 and Table 8 using the following formulae represented in equations (5) and (6):

$$X = \sum_{l=1}^{n} \frac{ThSymmetricCiphers(m)}{(Total\ Numberof\ input\ Buffer\ size)} \quad (5)$$

$$T_{AVG}Th_{OS(i)\_TLSLibraries(j)} = \sum_{k=1}^{n} \frac{X}{(Total\ number\ of\ Operating\ System)} \quad (6)$$

Where X= Average Throughput of TLS Libraries

And $T_{AVG}TH_{OS\_TLSLibraries}$ = Total Average Throughput of TLS Libraries

ThSymmetric Ciphers = Throughput of Symmetric Ciphers

i = Ubuntu, Fedora, Debian-etch, Windows and Mac operating systems

j = OpenSSL, GnuTLS, BoringSSL, AWS s2n, NSS and Cryptlib

k=1 & l=1

m= Symmetric Ciphers such as AES GCM, AES CBC…….in each libraries

n= Total number of Symmetric ciphers in each libraries

Table 7. NSS library, Symmetric Ciphers, Throughput (KB/s) with Mac

Operating System	Ciphers	Total Number of Input Buffer Size					Average	Total Average
Mac		16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes		
NSS	AES GCM	5826.56	10719.17	18428.16	21211.69	29123.41	**17061.8**	**16192.41**
	AES CBC	4671.35	10920.15	18515.97	22269.15	29210.46	**17117.42**	
	Camellia CBC	4981.35	10657.15	18515.97	21269.15	29210.46	**16926.82**	
	SEED CBC	5781.35	10920.15	19515.97	23269.15	29010.4	**17699.4**	

	3DES	4420.11	10641.73	19433.64	23183.17	20191.27	**15573.98**	
	Chacha20-Poly1305	2746.43	6554.58	15531.5	20031.62	19011.13	**12775.05**	

Table 8. Cryptlib Library, Symmetric Ciphers, Throughput (KB/s) with Mac

Operating System	**Ciphers**	**Total Number of Input Buffer Size**					**Average**	**Total Average**
Mac		**16 bytes**	**64 bytes**	**256 bytes**	**1024 bytes**	**8192 bytes**		
Cryptlib	AES GCM	4826.56	7719.17	9028.16	12211.69	14123.41	**9581.798**	**9756.133**
	AES CBC	4781.35	7820.15	9011.97	11263.15	14305.46	**9436.416**	
	3DES	4420.11	8241.73	11227.64	14183.17	13178.27	**10250.18**	

Table 9. Throughput (KB/s) Comparison of Ciphers for TLS Libraries with different operating system

Operating System	**OpenSSL**	**GunTLS**	**BoringSSL**	**AWS s2n**	**NSS**	**Cryptlib**
Ubuntu	33,743	27,192	19,792	12,192	16,192	9,756
Fedora	32,225	27,140	18,940	11,140	16,140	9,140
Debian-etch	33,814	27,899	19,099	10,899	16,899	9,876
Avg (Linux)	**33260**	**27410**	**19277**	**11410**	**16410**	**9590**
Windows	33,179	27,384	19,984	12,384	16,384	9,684
Mac	33,844	27,261	19,261	11,261	16,261	9,899
Average	**33,359.2**	**27,375.2**	**19,415.2**	**11,575.2**	**16,375.2**	**9,671**

The above Table 9 has been prepared by adding the throughput of ciphers of each library per distribution. Then at the end, the average per library is calculated and a bar chart has been presented as Fig. 4.

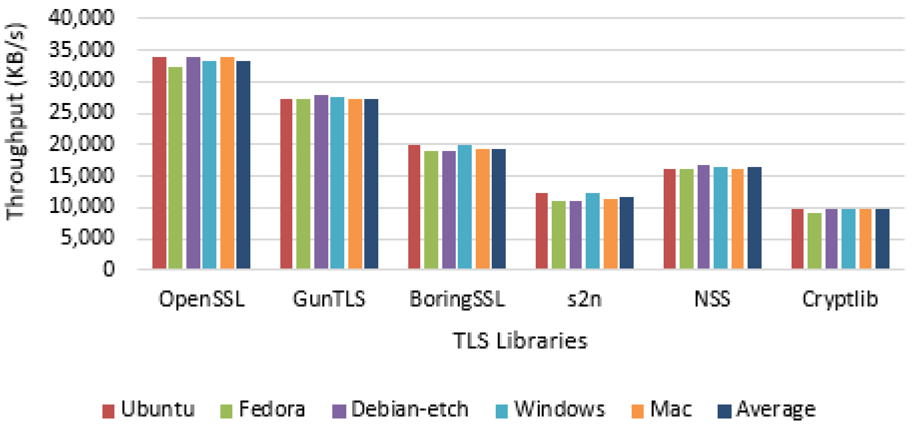

Fig. 4. Comparison of Throughput (KB/s) of Ciphers of TLS libraries on different Operating System

We conclude from above Fig. 4 and Table 9, which clearly show that the OpenSSL has higher throughput regardless of the Linux distribution it is running on. GnuTLS has higher throughput as compared to Boring SSL, AWS s2n, NSS, and Cryptlib libraries. The NSS having better output as compared to AWS s2n and Cryptlib. As the implementation results will scale up the throughput due to optimized implementation on new Operating Systems and better processors, so OpenSSL will still provide high throughput.

Observation: There are some issues regarding the methods used for the test conducted in the research.

- We can observe that the ciphers tested for each of the libraries are not the same leads to shifting in the true throughput of each cipher in the library.
- By varying the buffer size, results are obtained containing data for each cipher only once. Then the average is calculated with different buffer sizes. Multiple results with the same buffer would have produced the exact measurement then the average is taken.

5.2 Comparison of TLS Libraries with CPU-Usage

Here CPU usage is calculated for different TLS libraries using vmstat from the procps package with command vmstat –m based on Linux distribution. The CPU Usage (%) can be obtained using the formulae given in equations (7) and (8) and values tabulated in Table 10.

$$U_M\ (Mb) = T_M - (F_M + B_M + C_M) \quad (7)$$

Where U_M represents Utilized Memory, T_M represents Total Memory, F_M represents free Memory, B_M represents Buffered Memory and C_M represents Cached Memory.

$$\text{CPU-Usage}\ (\%) = (U_M / T_M) * 100 \quad (8)$$

Table 10. Comparison of TLS libraries with CPU-Usage on Linux

TLS Libraries	Utilized Memory (U_M)(Mb)	Total Memory (T_M) (Mb)	CPU-Usage (%)
OpenSSL	8063	8192	**98.42**
GnuTLS	7923	8192	**96.71**
BoringSSL	7834	8192	**95.62**
S2n	7412	8192	**90.47**
NSS	7568	8192	**92.38**
Cryptlib	7497	8192	**91.51**

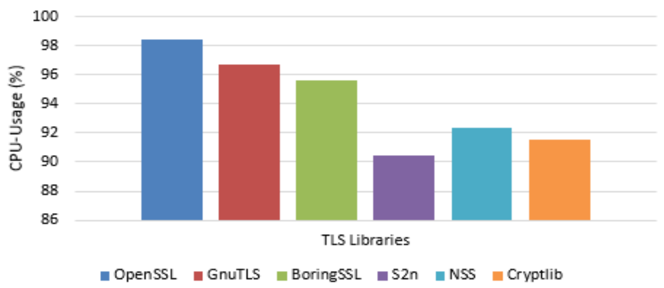

Fig. 5. Comparison of TLS Libraries with CPU –Usage

In, Fig. 5 and Table 10, we can observe the comparison between run type of memory and CPU utilization. The CPU usage by OpenSSL is higher as compared to libraries such as Gnu TLS, BoringSSL, AWS s2n, NSS, and Cryplib. It means higher the performance of CPU usage faster the libraries will run. In this case, the OpenSSL has the faster execution of libraries cipher suite.

Similarly, GnuTLS has high CPU usage compared to BoringSSL, NSS, AWS s2n, and Cryptlib. The CPU usage by AWS s2n is also low among all libraries.

5.3 Result Analysis

As we have seen various analysis criteria for the TLS libraries, it should be easier to get a clear view of the best library among all compared. But the choice of the best library can be categorized based on the following categories.

Higher Throughput: The results observed consistently showed that OpenSSL has an overall high throughput for cryptographic algorithms. The optimization present in the library, support for the AES-NI set can be beneficial to achieve desired throughput and which is easy to implement.

Portable and lightweight: When questions arise about the implementation of TLS on a mobile platform or memory constraints of the systems, the developer will need to use the library that has less size and is portable. GnuTLS can provide a lightweight C language API for various cryptographic operations. There have been some security cocerns regarding the bug discovered in GnuTLS for certificate verification and fixed in the latest versions.

Cross - Platform Support: If the application needs to support cross-platform functionality, then NSS can be an excellent choice. The same library has components and modules which are compatible with both UNIX-based and Windows systems.

License compatibility: The OpenSSL is under Apache License and is open to use, but some constraints make this license incompatible with General Public License. In development, there is a possibility of interoperability between applications with these licenses, which can cause some license issues. GPL licenses are widely used GnuTLS with GPL and NSS under Mozilla license have compatibility with GPL license. So if there are some components in the infrastructure using GPL license, then selecting GnuTLS or NSS would be a better choice.

Novice TLS developer: If the developer implementing a TLS solution for the application or even if wanted to learn the TLS semantics, then OpenSSL will be the better choice. It has support avail-able from the community. It has industry-standard implementation and easy configuration.

6 Conclusion

In this paper, a comparison of six different TLS libraries having unique features has been done, to find an appropriate TLS library for secure communication. Performance tests observed and conducted justified the expected higher throughput for the OpenSSL library. The throughput calculation for Asymmetric algorithms, hashing algorithms, and cipher is analyzed, which will provide insights on resource intensive operations and tasks and how each library scales to that load. Consistent results observed in each performed test, the comparisons among OpenSSL, GnuTLS, BoringSSL, s2n, NSS, and Cryptlib in virtual environments proved the occurrences of overhead in virtual machine causing the throughput to lower. If the overhead in-creases with increasing the buffer size, then there is the possibility of drastic change in the throughput. The tests also justified the high throughput for OpenSSL than other TLS libraries in a virtual operating system environment. The experimental results obtained from performance based on CPU usage by OpenSSL is high compared to other libraries such as Gnu TLS, BoringSSL, AWS s2n, NSS, and Cryplib. It means the higher the CPU usage of TLS libraries, the faster libraries will run. The methods used in this work can be improved and fine-tuned in the future.

References

[1] E. Rescorla, RTFM Inc., N. Modadugu, Google Inc., "Datagram Transport Layer Security Version 1.2", (January 2012), RFC 6347, Internet Engineering Task Force (IETF), Available: https://tools.ietf.org/html/rfc6347#section-3.3

[2] K. Bhargavan, F. Kiefer, and P. Strub, "hacspec: Towards verifiable crypto standards," in International Conference on Security Standardisation Research (SSR), ser. LNCS, vol. 11322. Springer, 2018, pp. 1–20.

[3] R. Canetti, A. Stoughton, and M. Varia, "EasyUC: Using EasyCrypt to mechanize proofs of universally composable security," in IEEE Computer Security Foundations Symposium (CSF). IEEE, 2019, pp.167–183.

[4] J. P. Lim and S. Nagarakatte, "Automatic equivalence checking for assembly implementations of cryptography libraries," in Proc. of the IEEE/ACM International Symposium on Code Generation and Optimization, (CGO). IEEE, 2019, pp. 37–49.

[5] Barbosa, Manuel, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. "SoK: Computer-aided cryptography." In SP 2020-42nd IEEE Symposium on Security and Privacy. 2021.

[6] Mozilla Developer Network. Network Security Services. Available: https://developer.mozilla.org/enUS/docs/Mozilla/Projects/NSS#Documentation. Accessed: Aug 10, 2021.

[7] OpenSSL, Cryptography and SSL/TLS Toolkit - Threads, 1.0.2 manpages , (n.d), Available : https://www.openssl.org/docs/man1.0.2/crypto/threads.html

[8] GnuTLS, Transport Layer Security Library for the GNU system, for version 3.7.1, 3 March 2021. Available: https://www.gnutls.org/manual/gnutls.html

[9] Gutmann, Peter (2019). "Downloading". cryptlib. University of Auckland School of Computer Science. Retrieved 2021-07-07.

[10] Google. "BoringSSL." Google. https://boringssl.googlesource.com/boringssl/ (accessed 13 Jun, 2020).

[11] Schmidt, Stephen. "Introducing s2n, a new open-source TLS implementation, June 2015." (2015).

[12] Ye, K.Q., Green, M., Sanguansin, N., Beringer, L., Petcher, A., Appel, and A.W.: Verified correctness and security of mbedtls HMAC-DRBG. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–03 November 2017, pp. 2007–2020. ACM (2017).

[13] Bond, B., et al.: Vale: verifying high-performance cryptographic assembly code. In: 26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, pp. 917–934. USENIX Association (August 2017)

[14] Fromherz, A., Giannarakis, N., Hawblitzel, C., Parno, B., Rastogi, A., Swamy, N.: A verified, efficient embedding of a verifiable assembly language. Proc. ACM Program. Lang. 3(POPL), 63:1-63:30 (2019).

[15] Timo Bingmann, Speedtest, and Comparison of Open-Source Cryptography Libraries and Compiler Flags, 14 July 2008 [online]. Available: https://panthema.net/2008/0714-cryptography-speedtestcomparison/

[16] Zinzindohoué, J.K., Bhargavan, K., Protzenko, J., Beurdouche, B.: HACL*: a verified mod-ern cryptographic library. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–03 November 2017, pp. 1789–1806. ACM (2017).

[17] Lim, J.P., Nagarakatte, S.: Automatic equivalence checking for assembly implementations of cryptography libraries. In: Kandemir, M.T., Jimborean, A., Moseley, T. (eds.) IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2019, Washington, DC, USA, 16–20 February 2019, pp. 37–49. IEEE (2019).

[18] Erbsen, A., Philipoom, J., Gross, J., Sloan, R., Chlipala, A.: Simple high-level code for cryptographic arithmetic - with proofs, without compromises. In: Proceedings of the 40th IEEE Symposium on Security and Privacy, S&P 2019 (May 2019)

[19] Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Sean Gulley, Wajdi Feghali "Improving OpenSSL* Performance" in IA Architects Intel Corporation, October 2011. Available: https://software.intel.com/sites/default/files/open-sslperformance-paper.pdf

[20] Boston B. et al. (2021) Verified Cryptographic Code for Everybody. In: Silva A., Leino K.R.M. (eds) Computer Aided Verification. CAV 2021. Lecture Notes in Computer Science, vol 12759. Springer, Cham.

[21] Nettle signature verification CVE-2021-20305, 2021, [online] Available at: <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2021-20305> [Accessed 13 October 2021].

***