Comparison of SSL/TLS libraries based on Algorithms/languages supported, Platform, Protocols and Performance

Akshay Thorat, George Mason University Fairfax, Virginia 22030

***Abstract*— Security in Computer Network Communication is important which can be achieved by Transport Layer Security (TLS) protocols. Various libraries have been created for implementation of TLS functions, of which each has wide support of the encryption algorithms, key exchange mechanism from which one can implement TLS solution. Depending on the need of TLS application, developer or organization has to choose effective and efficient library which will provide the appropriate results and fulfill the expectation of the project or application.**

In this work, the focus is on comparisons of the widely used libraries like OpenSSL, GnuTLS, NSS, Cryptlib. The primary focus of comparison will be on the Cipher suites present, Platform, Protocol Support and Performance based on processor and physical memory usage. For OpenSSL and GnuTLS libraries, the implementations are tested on virtual machine for comparisons of throughput. This work will provide unified view of the libraries which are scattered as those are managed by independent and separate communities. The selection of correct library can be tricky as each library has different cipher support, portability, platform and flexibility. Depending on the type of users like student, researchers, developer, organizations, the selection can change according to the ease of implementation, debugging, portability and licenses. The comparisons based on these criteria are made in this work.

***Index Terms*—TLS, OpenSSL, GnuTLS, Performance**

I. Introduction

Security in communication is essential to facilitate reliability, data integrity and confidentiality, Transport Layer Security (TLS) can provide these aspects for secure connection over computer networks. It can be used in Email, VOIP (Voice Over Internet Protocol), Web browsing, Bank transactions for prevention of eavesdropping and tampering of data. As TLS has been evolve from SSL 1.0 over the years, some of the features like hash functions, key exchange algorithms have been changed to comply with the need to have better security.

TLS versions evolved over the years are as follows,
SSL1.0->SSL2.0->SSL3.0->TLS1.0->TLS1.1->TLS1.2->TLS1.3 (draft). Each suite contains authentication, message authentication code (MAC), key exchange and encryption algorithms.

Using version below TLS1.0 is not advised, as those are less secure and most of the browsers provide warning if some site is having old version. The support for implementation TLS1.3 working draft, which is still in development but can be used, is provided by the widely used Google Chrome and Mozilla Firefox browsers (by default TLS 1.3 is not enabled). Websites like https://www.cloudflare.com/ have started using TLS 1.3 implementation.

For applications using datagrams, which is connection less, Datagram Transport Layer Security (DTLS) is used. DTLS is made similar to TLS [1] except that for DTLS, it has to solve problems of packet lost and reordering. DTLS implements

1) Packet Retransmission – Lost packets retransmission mechanism.
2) Sequencing for Packets – Assign sequence number to datagram for reordering and packet loss.
3) Replay detection – Used to avoid duplicate packets and discarding old received packets.

II. Libraries Under Consideration

A. Primary

1) OpenSSL – It is robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) [2]. It is licensed under an Apache license from the organization for free use.
2) GnuTLS - Portable ANSI C based library with Lesser General Public license (GPL)

B. Other

1) Network Security Services (NSS) - Library designed to support cross-platform development of security-enabled client and server applications
2) Cryptlib - Open source cross-platform software security toolkit library for SSL/TLS and SSH secure sessions, Certificate Authority (CA) services.

For the mentioned libraries, comparisons will be provided based on the criteria such as implementation, features, Algorithms supported and Performance.

III. Library to Implementation comparison

- From Table 1, it is clear that all the libraries support TLS versions till 1.2

TABLE I
LIBRARY TO IMPLEMENTATION/VERSION COMPARISONS

Library	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	DTLS 1.0	DTLS 1.2
OpenSSL	Yes	Yes	Yes	Yes	Yes	Yes	Yes
GnuTLS	Yes	Yes	Yes	Yes	No	Yes	Yes
NSS	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Cryptlib	Yes	Yes	Yes	Yes	No	No	No

- For TLS 1.3(draft), implementation is supported only by OpenSSL and NSS
- Except Cryptlib, all three supports DTLS versions. So if the datagram application is needing TLS implementation then the developer has to use libraries other than Cryptlib for implementation.

IV. ALGORITHMS SUPPORTED

Each library includes various types of algorithms which facilitates Authentication, Message Authentication Code (MAC), key exchange and encryption services for the communication.

TABLE II
LIBRARY TO ALGORITHMS COMPARISONS

	OpenSSL	GnuTLS	NSS
Public key Cryptography	RSA, DSA, Diffie–Hellman key exchange, Elliptic curve	RSA, DSA, Diffie–Hellman key exchange, Elliptic curve	RSA, DSA, Elliptic curve
Hash Functions	MD4, MD5, SHA-1, SHA-2, GOST, BLAKE2, Whirlpool	MD5, SHA-1, SHA-2, RIPEMD-160	MD5, SHA-1, SHA-2
Ciphers	AES, Blowfish, Camellia, RC5, 3DES	AES, Camelia, 3DES, RC2, RC4, SALSA20	AES, 3DES, Camellia, RC2, RC4

All the libraries support the important algorithms, the only variation can be seen in the supported key, MAC (hash functions) and historical ciphers. For successful implementation of TLS, the correct combinations of ciphers should be used. Each library has default algorithm selected which can be changed according to the need of security.

V. GENERAL COMPARISON

The general comparisons criteria include, Languages, Cryptographic Token, Thread safety and CPU assisted Cryptography. Each of the type provides unique specification for the libraries and selection criteria can be vastly varied.

A. *Languages supported*

All of the libraries support for C language with NSS having extra feature for C++. The one of the unique part of NSS is having cross-platform support. The same library has modules and functions which can be used directly with UNIX based or Windows machine. As for others some other wrapper package or patches needs to be added.

TABLE III
GENERAL COMPARISONS

	OpenSSL	GnuTLS	NSS
Languages	C	C	C or C++
Cryptographic Token Interface-PKCS #11	Not Present natively *	Present	Present
Thread safety	Two callback function with POSIX or Win Threads	Use POSIX or Win Thread	Yes
CPU Assisted Crypt. With AES-NI	Yes	Yes (+VIA Padlocks)	Yes

B. *Cryptographic Token Interface*

It provides Programming interface to create and manipulate cryptographic tokens. It has Platform Independent Application Programming Interface (API) for Hardware Security Modules (HSM) and Smart Cards. This API has been included in Public Key Cryptographic Standard #11, commonly known as PKCS #11, "Cryptoki".

We can notice that GnuTLS and NSS has support for PKCS #11 natively, but it’s not the case with OpenSSL. To use the API in OpenSSL, API engine needs to be added externally through a patch. Now selecting a correct engine as per the requirement will be essential for successful token implementations. So in this criteria, selecting native implementation of PKCS #11, which is present in NSS or GnuTLS, will be uncomplicated and effortless.

C. *Thread Safety*

It insures safe handling of shared data structures by which all threads behave correctly without violating their specification. By safely locking the shared data, race conditions can be avoided, else will result in faulty behavior.

Now, even if OpenSSL supports thread safety with POSIX or Windows Threads, earlier versions than 1.1.0, it can safely be used in multi-threaded applications provided that at least two callback functions are set, locking_function and threadid_func and for later versions, with some limitations; For example, an SSL connection cannot be used concurrently by multiple threads [2]. GnuTLS and NSS also provide thread safety with use of POSIX or windows thread. GnuTLS utilizes mutual exclusion locks for the data structure protection, for example Random number generator locks are setup by GnuTLS on library initialization [3].

D. CPU Assisted Cryptography

This mechanism uses hardware acceleration for cryptographic functions with supported instruction set. By using accelerators, intensive cryptographic operations can be run more efficiently than running on general purpose CPU.
Advanced Encryption Standard New Instructions(AES-NI) is an extension for x86 instruction set architecture, used for Intel or AMD processor. It can be used to improve the speed of applications performing encryption and decryption using the Advanced Encryption Standard (AES). All three libraries can take advantage of this feature. GnuTLS has additional support for VIA PadLock Security Engine in VIA x86 processors. It uses different instruction set than AES-NI for AES acceleration. So GnuTLS can provide support for systems with VIA processor.

VI. Performance Comparison

A. Speedtest and Comparison of Open-Source Cryptography

This comparison results have been obtained from internet article named "*Speedtest and Comparison of Open-Source Cryptography Libraries and Compiler Flags*" by Timo Bingmann[4]. The speed comparison test was performed using ciphers found in well-known open source cryptography libraries, out of which OpenSSL and GnuTLS considered in this work. The throughput calculated is, data processed per unit time that is bytes/second. The data collected is from five different CPUs and five different Linux distributions to reveal details about 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 - AES, Blowfish, CAST5, 3DES
- GnuTLS – AES, Blowfish, CAST5,3DES, Serpent, Twofish

TABLE IV
Throughput(KB/s) Comparison for Libraries

Linux Version	GnuTLS	OpenSSL
Ubuntu-hardy	12,192	33743
Debian-lenny	12,384	33179
Ubuntu-Gutsy	11,261	33844
Fedora8	11,140	32225
Debian-etch	10,899	33814
Average	11,575	33361

The above Table IV has been calculated with adding the throughput of ciphers of each library per distribution. Then at the end average per library is calculated.

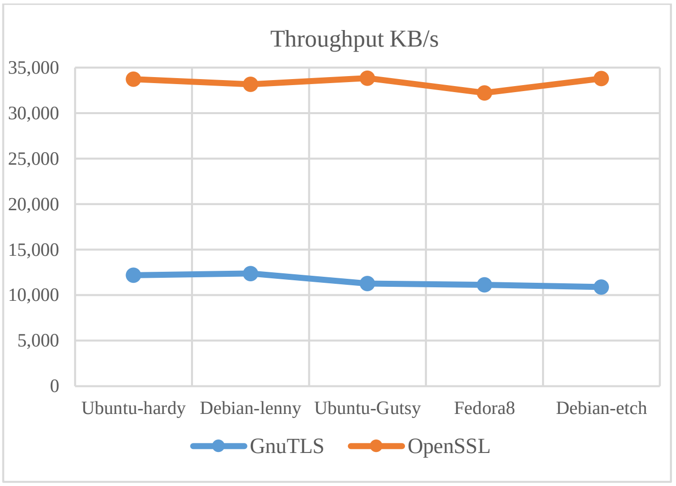

Fig. 1. Comparison of Throughput to OS for OpenSSL and GnuTLS

The graph constructed from above Table IV clearly shows that the OpenSSL has higher throughput regardless of the Linux distribution it is running on. The same trend will be observed for latest Linux versions, 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.

Problems
There are some issues observed, regarding the methods used for test conducted in the research.
 - We can observe that the ciphers tested for each of the library are not same. This could result in shifting the true throughput each ciphers in each library.
- The results were obtained with varying the buffer size, containing data, for each cipher only once, after the average was calculated with different buffer size. Multiple results with the same buffer would have produced the exact measurement and then the average should have been taken.

B. Comparison for NSS Library

This comparison is based on the archived NSS Performance Results [5] from the developer of the library Mozilla, present at Mozilla Developer Network website.

TABLE V
Library to Implementation/Version Comparisons

Type	Operations/sec	CPU-Usage (%)
Full	156.23	95
Full-zones	216.55	100
Restart	220.76	90
Restart-zones	569.82	86

- For above tests the ciphers used were as follows SSL_RSA_WITH_RC4_128_MD5 (SSL3)
- When Restart runs are used for testing, it utilizes cache which results in higher throughput which saves memory lookup time.

- Full runs use handshake every connection which introduce overhead so throughput decreases.

In, Fig. 2, we can observe the comparison between, for run type of Throughput and CPU utilization.

- For each run, CPU utilization is near about same as significant change is not observed.
- But the throughput is increasing for same CPU utilization, which could imply that varying throughput could be a function of Memory access speed.
- Also the use of cache can be helpful in increasing the throughput as it saves memory access time for data fetching.

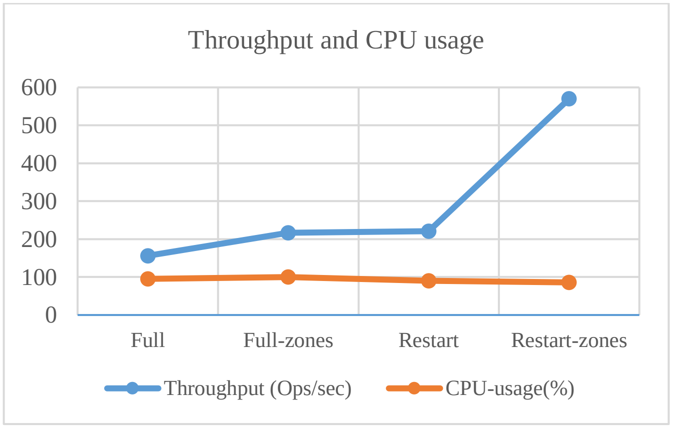

Fig. 2. Comparison of Throughput to CPU Usage in each run type

C. *Performance Tests with Virtual Machine*

For the third part of comparisons, OpenSSL and GnuTLS is installed on my test machine. Comparisons are done based on throughput of hashing algorithms, where libraries are running on native Operating System (OS) and OS running inside Virtual Machine. Here Type 2 – Hypervisor is used, where Virtual Machine Monitor (VMM) runs on the OS, just as another application or program.
The environment for running the libraries are same for both native OS and virtual OS. The configuration for the system are as below,

- OS - Ubuntu 16.04
- Processor – Intel core i5 2.20GHz
- Physical Memory – 4GB
- Physical Disk – 25GB

1) OpenSSL

The measurement for OpenSSL are taken by running the command *"openssl speed [cipher_name]"*. Here the tests are conducted for hashing algorithms. SHA256 is ran with varying buffer size of 16,64,256,1024,8192 Bytes. The results obtained are as follows for each OS,

Native OS

```
Doing sha256 for 3s on 16 size blocks: 11070869
(iterations) sha256's in 3.00s
Doing sha256 for 3s on 64 size blocks: 5862352 sha256's in
3.00s
Doing sha256 for 3s on 256 size blocks: 2852521 sha256's
in 3.00s
Doing sha256 for 3s on 1024 size blocks: 928502 sha256's
in 3.00s
Doing sha256 for 3s on 8192 size blocks: 119962 sha256's
in 3.00s

The 'numbers' are in 1000s of bytes per second processed.
type    16 bytes    64 bytes    256 bytes    1024 bytes    8192 bytes
sha256  59044.63k  125063.51k  243415.13k  316928.68k    327576.23k
```

Virtual OS

```
Doing sha256 for 3s on 16 size blocks: 9530478 sha256's in
2.92s
Doing sha256 for 3s on 64 size blocks: 5334553 sha256's in
2.92s
Doing sha256 for 3s on 256 size blocks: 2443784 sha256's
in 2.81s
Doing sha256 for 3s on 1024 size blocks: 778802 sha256's
in 2.95s
Doing sha256 for 3s on 8192 size blocks: 90726 sha256's in
2.97s


The 'numbers' are in 1000s of bytes per second processed.
type   16 bytes   64 bytes   256 bytes   1024 bytes   8192 bytes
sha256 52221.80k  116921.71k  222636.55k  270336.69k  250244.91k
```

Above results are combined into table for better understand how the throughput is getting affected. The throughput values are in Kbytes/seconds and buffer sizes are in bytes.

TABLE VI

THROUGHPUT TABLE FOR OPENSSL RUNNING IN NATIVE OS AND VIRTUAL OS

Size(Bytes)	**Native**(KB/s)	**Virtual**(KB/s)
16	59044.63	52221.8
64	125063.51	116921.7
256	243415.13	222636.6
1024	316928.68	270336.7
8192	327576.23	250244.9

Fig. 3, shows throughput comparison between running the OpenSSL library in native OS and Virtual Machine (VM) OS. It is clear that the throughput obtained with library running natively is higher. This is due to the overhead of running library in VM, where the operations are translated into base machine instructions which is handled by VMM, instead of directly handled by OS, so it is less efficient.

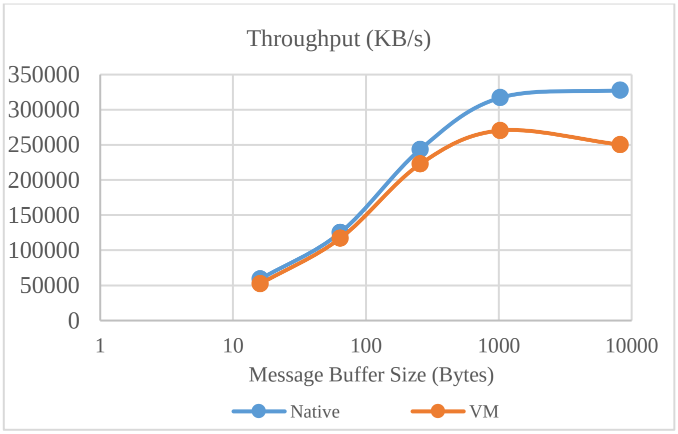

Fig. 3. Throughput Comparison for OpenSSL for Native and Virtual Machine OS

Observed behavior

- After increasing the buffer size above 1000 bytes, the throughput starts to saturate, so this can specify the optimal buffer size for maximum throughput. Here the observed throughput is around 330MB/s
- The interesting observation can be done for VM that as the throughput starts to decrease at buffer size 10000. After increasing the buffer size above this value, bottleneck can occur. This is due to the resources (here physical disk) are getting exhausted for handling the data. This bottleneck can cause the throughput to lower, below the acceptable level.
- So it is recommended, either to run library in native OS or keeping the buffer values in well-defined limit.

2) *GnuTLS*

The command used for GnuTLS performance measurement is "gnutls-cli" with benchmark ciphers options. Test is performed with the fixed payload buffer size of 16384 bytes with three separate hashing functions SHA1, SHA256 and SHA512. The results obtained from the run are as follows,

```
Native OS
Checking MAC algorithms, payload size: 16384
        SHA1 0.61 GB/sec
        SHA256 0.29 GB/sec
        SHA512 0.32 GB/sec

Virtual OS
Checking MAC algorithms, payload size: 16384
        SHA1 0.55 GB/sec
        SHA256 0.26 GB/sec
        SHA512 0.29 GB/sec
```

Table VII is constructed from above data. The measurements are converted into kilo bytes of data processed per unit time.

TABLE VII

THROUGHPUT COMPARISON FOR GNUTLS RUNNING IN NATIVE OS AND VIRTUAL OS

Type	**Native**(KB/s)	**Virtual**(KB/s)
SHA1	639631.4	576716.8
SHA256	304087	272629.8
SHA512	335544.3	304087

In Fig. 4, the same observation, as for previous OpenSSL library, of lower throughput value can be seen in the VM. As discussed before, this is due to introduction of the overhead of running library in VM.

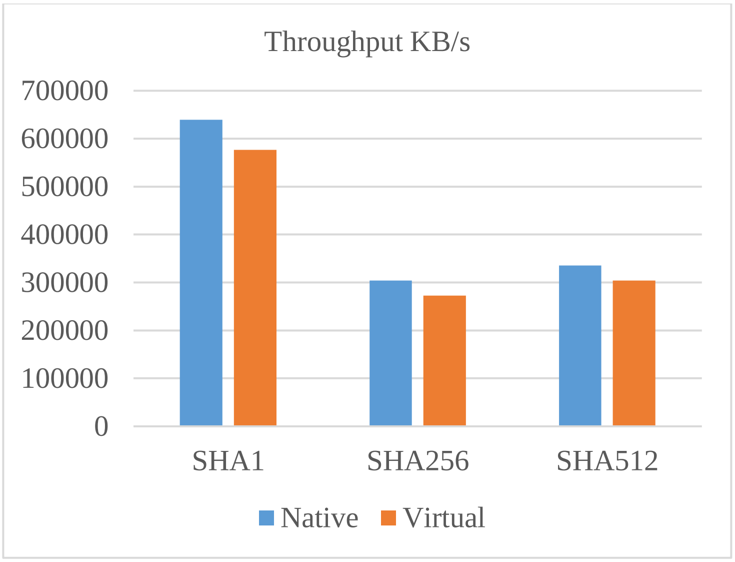

Fig. 4. Throughput comparison for different Hashing Algorithm in Native and Virtual OS for GnuTLS.

We can see that the throughput for SHA256 is around 300MB/s which is less than OpenSSL. So it can be confidently said that OpenSSL has better throughput for the implementation of hashing which is tested for SHA256.

VII. BEST LIBRARY

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

A. *Higher Throughput - OpenSSL*

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

B. *Portable and lightweight - GnuTLS*

When the question is about the implementation of TLS on mobile platform or memory constraints in the system, then the developer will need to use library which is of less size and portable. GnuTLS can provide lightweight C language API for various cryptographic operations. There has been some security concern regarding the bug discovered in GnuTLS for certificate verification but it has been fixed in latest versions.

C. *Cross-Platform Support – NSS*

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

D. *License compatibility*

It is important to notice that OpenSSL is under Apache License and is open to use, but there are constraints that make this license incompatible with General Public License (GPL). If in development, there is interoperability between applications with this licenses, it can cause some license issues. GPL license are widely used, GnuTLS with GPL and NSS under Mozilla license has compatibility with GPL license. So if there are some components in the infrastructure using GPL license then selecting GnuTLS or NSS would be better choice.

E. *Novice TLS developer – OpenSSL*

If the developer implementing TLS solution for the application or even if wanted to learn of the TLS semantics, then OpenSSL will be better choice. It has wide support available from the community. It has industry standard implementation and easy configuration.

VIII. Conclusion

The work done in this project concisely provides various differentiating factors for comparing the TLS libraries. The libraries considered, each has some unique features which can be used to implement cryptographic functions. At first we saw, how various versions of the TLS implementation has been included in each of the library. Only TLS1.3 and DTLS support was varied. Algorithm supported provides unified view of implemented cipher and algorithms. General comparisons provided API support, HSM model support which is provided by GnuTLS and NSS but not by OpenSSL natively (external engine patch needed). Multithreaded support insight along with AES-NI operations compatibility was discussed.

Performance tests observed and conducted justified the expected higher throughput for OpenSSL library. The consistent results were observed in each of the tests performed. For NSS, Fig. 2 provides information on the varying throughput with respect the memory access speed. The comparisons made for OpenSSL and GnuTLS in native OS and virtual OS proved, the occurrences of overhead in VM causing the throughput to lower. If the overhead increases with increasing the buffer size, then there is possibility of drastic change in the throughput. The tests also justified the high throughput for OpenSSL than GnuTLS in both native and virtual OS.

For future scope the methods used in this work can be improved and fine-tuned. The throughput calculation for Asymmetric algorithms (Public Key) can be calculated, which will provide insights on resource intensive operations and tasks and how each library scales to that load.

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] 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

[3] GnuTLS, Transport Layer Security Library for the GNU system, for version 3.6.1, 21 October 2017. Available: https://www.gnutls.org/manual/gnutls.html

[4] Timo Bingmann,(14th July 2008), Speedtest and Comparison of Open-Source Cryptography Libraries and Compiler Flags, [online]. Available: https://panthema.net/2008/0714-cryptography-speedtest-comparison/

[5] Mozilla Developer Network. Network Security Services. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS#Documentation . Accessed: Dec 12, 2017.

[6] Transport Layer Security. (n.d). Wikipedia. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security

[7] Comparison of TLS implementations. (n.d). Wikipedia. Available: https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations

[8] Digital Data Security Limited. (n.d). cryptlib. Available: http://www.cryptlib.com/

[9] Daniel Stenberg (n.d). Compare SSL Libraries. [online]. Available: https://curl.haxx.se/docs/ssl-compared.html

[10] OpenWrt Oraganisation (n.d). OpenSSL Benchmarks tool. [online]. Availble: https://wiki.openwrt.org/doc/howto/benchmark.openssl

[11] 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-ssl-performance-paper.pdf