Journal of Engineering Science and Technology
Vol. 17, No. 2 (2022) 1508 - 1522

THE IMPACT OF BLOCK AND STREAM CIPHER ON SESSION INITIATION PROTOCOL PROXY PERFORMANCE

ALI ABDULRAZZAQ K.

Department of Computer Science, College of
Education for Pure Science, University of Mosul, Mosul. Iraq
E-mail: aliabd@uomosul.edu.iq

Abstract

Session Initiation Protocol (SIP) has a promise future in communication era, it is known to be common protocol found to provide desired services in multimedia communication (voice and video). SIP transmits its signalling massages and media through various mean of transport protocols UDP, TCP, TLS, and RTP. Generally, TLS has been considered as a promise approach to secure SIP signalling; authentication and registration. Till recent, using TLS for SIP is not yet widespread in research community and practice fields, potentially due to concerns about the performance overhead caused by cryptographic algorithms. However, few studies have considered performance overhead in SIP over TLS in general. Based on these considerations, this paper studies the performance impact of applying set of bulk cipher algorithms of TLS over SIP server's communication. Five Block/stream cipher algorithms (AES, 3DES, SEED, Camellia, and RC4) have been tested across SIP server to be evaluated individually. The aim of this research is to illustrate how each cipher algorithm affects SIP server performance. Experimental results depict that AES cipher algorithm cost OpenSIPS proxy 1.9% of CPU usage due to the fact that AES requires high process to encrypt SIP messages. Throughout the scenarios, RC4 runs with insufficient throughput due to stream cipher behaviour. Lastly, for devices with limited/low CPU resource, Camellia algorithm is a proper choice as it records ever lower CPU usage. This work aims to help SIP service provider to utilize cipher algorithms with better understanding the impact of each of which when real time communication.

Keywords: Block cipher, Cipher algorithms, SIP performance, SIP, Stream cipher, TLS.

1. Introduction

SIP (Session Initiation Protocol) [1] is a proper choice for handling voice over IP communication due to its flexibility, simplicity, and QoS. SIP is a signalling protocol used as a reliable technique for initiating, modifying and terminating multimedia sessions. Technically, SIP works jointly with SDP (Session Description Protocol) [2] as multimedia parameters' descriptor and transmitting data (signalling and/or audio) via various transport protocols such as UDP (User Datagram Protocol) or TCP (Transmission Control Protocol) [3]. However, none of them are secured enough, they are providing no signalling confidentiality, authenticity, or integrity. Given a trace of SIP connectivity, attackers can sniff sensitive information like who is communicating with whom, when, and what is being said. Thus, SIP uses TLS (Transport Layer Security) protocol as a preferred transport protocol to secure its connectivity [4].

TLS protocol is found to ensure encryption, authentication, and data integrity for transmitting information over networks. Major standards Internet applications including web, email, and real-time multimedia communication have all adopted TLS as the core security protocol. TLS protocol is processed in two phases [5]: first phase called handshake, in this phase several agreements are set between client and server, such as TLS version, compression method, and cipher suite. Cipher suite string includes the following algorithms; server authentication, key exchange, bulk/stream encryption, and measure digest. This agreement (cipher-suite string) is offered by the client and to be accepted by the server side based on a proper and preferable algorithm. In particular, the main actions in this phase are to establish a shared session key (private and/or public). Also, the client and server have to authenticate (optional) each other via certificate. This certificate is provided by a trusted third-party called Certification Authority (CA) [6]. The second phase in TLS protocol is called record layer, in which data within each record is encrypted/decrypted using the negotiated block cipher [7].

As the security concern widen with TLS protocol, variant of block cipher algorithms has been deployed to meet the desired security criteria. Block cipher algorithms includes RC4, 3DES, AES, SEED, Camellia, and Blowfish, each of which are complex in nature and have variant structure. That in turn cause SIP service providers pay extra concern in deploying TLS with low level of understanding the impact of heavy algorithms there [8]. Thus, further studies have to be conducted on how much each block cipher is affectively impact SIP server.

The main objective of this research is to provide an experimental performance study of the impact of using each block cipher algorithm on SIP server. One algorithm has different block size, key size, box structure which all yields in deferent performance while deploying SIP over TLS. Performance evaluation uses library and kernel profile to examine and explain SIP server performance. Three parameters are utilized to evaluate block ciphers performance such as CPU usage profile, call response time, and throughput under different scenarios: inbound, outbound, and multi-to-one proxy. In addition, this research offers a guideline to highlight a preferable block cipher for SIP service provider and in which circumstances.

In literature, several studies have been conducted on TLS performance in general or in other application such as web servers. Shen et al. [9] studied the impact of TLS on SIP server in general with different TLS actions; TLS session reuse, mutual authentication, and then compare it with UDP and TCP. Shen et al. [10] investigated the performance impacts of SIP server using TLS and SRTP as secured protocols and compare it with UDP and TCP as non-secured protocols. Müller et al. [11] proposed a

benchmarking approach to determine the performance impact of TLS in cloud database system (DynamoDB and Cassandra) in general with no concern about cipher suite algorithms. Adrianto and Lin [12] studied the resource consumption in IoT platforms when it uses TLS handshake and Record layer. Zhao et al. [13] profiled the TLS processing and show how the crypto algorithms affect SSL. Also, the overall effect of SSL performance in a web server environment has been analysed by showing the time spent into crypto and non-crypto portions. However, none of the literature have concerned about block cipher algorithms with SIP server.

The first three sections show the background on SIP, TLS, and Block cipher algorithms. Section 5 presents the Methods of this paper. Later, Section 6 discusses the experimental results and shows the final finding of this research. Last, conclusion is presented in Section 7.

2. Session Initiation Protocol

SIP is a signalling protocol defined and adopted by the SIP Working Group, under the umbrella of Internet Engineering Task Force (IETF). The protocol was published with the first issue of RFC 3261 [1] toward the status of a proposed standard. In general, SIP is commonly used for triggering and controlling multimedia sessions (voice, video) by establishing, modifying and terminating sessions [14].

SIP cooperates with SDP (Session Description Protocol) [2]. SDP is responsible of setting multimedia parameters such as IP version, port number, and voice codecs for RTP (Real-Time Transport Protocol) streams [15].

During call setup, SIP at session layer delivers data to next layer through TLS protocol, specifically in TLS record layer. Record layer is responsible of symmetric cryptography, fragmentation, encryption/decryption then sending/receiving TLS messages to/from the transport layer, preferably TCP. It is worth to mention earlier that TLS protocol applied to SIP signalling session only (call setup) over reliable protocol such as TCP. In other words, transferring media (voice, video) will be handled using unreliable protocol UDP which in practice, it is not recommended to use TLS over it [16].

SIP and TLS server are possible to be located physically in one machine for to avoid external network factors. Once the SIP server receives an INVITE message from the client, initially the SIP sever will check wither the client willing to communicate with TLS or not. If the client holding the TLS header in it SIP message the SIP server will invoke SSL server for encryption/decryption process, see Fig. 1.

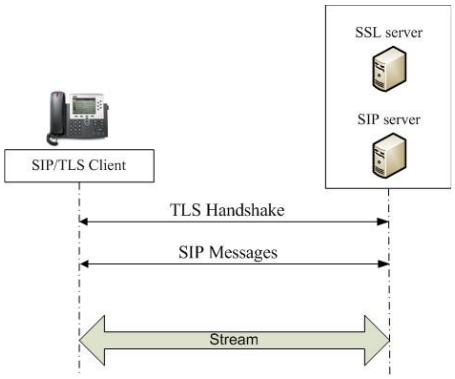


Fig. 1. Message exchange diagram.

3. Transport Layer Security (TLS)

TLS protocol is found to create a secure connection between server and client, to obtain high authenticity, integrity, and confidentiality. Technically, TLS protocol achieved through two processing layers which are handshake and record layers. Handshake layer adopts different Public Key Cryptography (PKC) algorithms for identity authentication and cipher suite negotiation. However, cipher suite string offers most preferable cryptographic algorithms between clients and server [17].

Every SSL/TLS connection in handshake triggers the negotiation between two parties that describes the details of TLS connection between UAC and SIP server [18]. In this stage, the handshake decides what cipher suite will be used to encrypt communications, verifies the server (and the client optionally), and establishes a secure connection prior data transmission [19].

After the handshake phase complete, record layer will be invoked to transmit data between server and client. Data will be secured and ready to be transmitted using symmetric cryptographic algorithms that already been accepted during handshake steps.

Record layer protocol submits messages for transmission, fragmenting the data into manageable blocks, data compression (optional), applying a MAC, encryption, and delivering to lower-level (TCP). At the receiving side the reversed active will take place.

A bulk cipher is a symmetric encryption algorithm where the actual amount of data will be encrypted/decrypted [20]. There are two types of bulk ciphers in TLSv1.2 which are block and stream ciphers. A block cipher encrypts data in blocks of bytes. During the TLS development, meanwhile several Bulk cipher algorithms have been proposed, each of which provides variant level of security and performance. Block ciphers include Triple-Data Encryption Standard (3DES) (168-bit), Advanced Encryption Standard (AES), SEED, and Camellia which are the most common block ciphers in research community.

Stream cipher encrypts 1 byte of data at a time, converts a key to a key stream to encrypt serious of Bytes and produce cipher text. The received part will convert the shared key to the exact key stream and decrypts the plaintext back to data. RC4 (128 bit) is the most common stream cipher [21].

It is worth to mention here, that all the operations mentioned earlier in TLS section such as obtaining certificate, key generation, signature, and certificate authenticity are all occurred once during connection setup. Whereas bulk cipher invoked with each data transmission. That, however, motivates the author of this paper research to concern about bulk cipher algorithms' performance as it considers a real factor that affects SIP server real time performance when TLS protocol is deployed.

4. Bulk Cipher Algorithms

A cipher suite is a set of cryptographic algorithms essential for TLS protocol setup in order to secure end-to-end connection. The cryptographic algorithms are used for handshake along with record layer operations including key (public and/or private) exchange, signature to handle authentication, bulk data encryption, and message integrity. A cipher suite string example shown in Fig. 2.

In this research only bulk cipher algorithms have been considered while keeping other algorithms unchanged. There are several types of bulk cipher algorithms

known in the research community and most used in practice. Bulk ciphers algorithms are briefly listed at the following and summarized in Table 1.

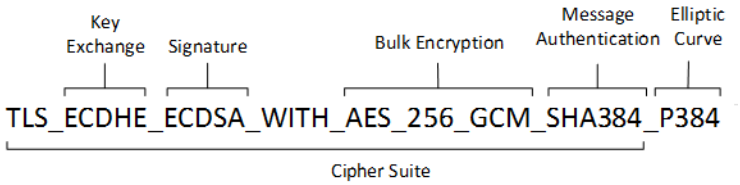


Fig. 2. Example of cipher suite string.

- Advanced Encryption Slandered (AES): AES is an iterative cipher, based on 'substitution-permutation network'. Thus, AES deals with 128 bits of a plaintext block as 16 bytes. AES uses 128, 192, and 256-bit keys with number of rounds 10, 12, and 14 respectively. Technically, each of these rounds are using different 128-bit round key, originally they are calculated from the initial AES key [22]. AES is computed using bytes rather than bits.
- Triple Data Encryption Standard (3DES): 3DES is a symmetric key block cipher [23]. 3DES in fact use DES cipher algorithm but with three times for each data block. By which, 3DES will provide more secure encryption of DES. 3DES operates with 168-, 112- or 56-bits key and it use a block size 64 bits using 48 DES-equivalent rounds.
- Camellia: Camellia is a symmetric key block cipher with 128 bits block size using key sizes of 128, 192 and 256 bits. Camellia algorithm found by Mitsubishi Electric and NTT in Japan. However, Camellia works either with 18 or 24 rounds using (128-bit), 192- or 256-bits key respectively. Camellia implemented with (four) 8×8-bit *S-boxes* that supported with both input/output transformations and logical operations. In addition, Camellia cipher is implemented with input and output key whitening [24].
- SEED: this algorithm is a 128-bit key symmetric block cipher that is designed to use the S-boxes and permutations. SEED has 128-bit key length with 128-bit block size. It uses Feistel structure with a 16-round [25]. The 128-bit block is divided into two sub-blocks which are 64-bit left and 64-bit right. Initially, the right block taken as an input for the next round function (F), with a 64-bit sub-key Ki that generated by key schedule. Left block considered as the most significant 64-bit input, while Right is the least significant 64-bit.
- Rivest Cipher 4 (RC4): RC4 is a stream cipher utilizes key size 40-2048 bits and a stream of 2064 bits for 1 round. RC4 is known wildly for its simplicity and high performance in software. Meanwhile, several vulnerabilities have been discovered in RC4 leading it insecure [26].

Table 1. Bulk ciphers algorithms.

Bulk Cipher	Key size	Block size	Round
AES	128, 192, and 256-bit	128 bits	10, 12, and 14
3DES	168, 112 or 56 bits	64 bits	48
Camellia	128, 192 and 256 bits	128 bits	18 or 24
SEED	128 bits	128 bits	16
RC4	40-2048 bits	2064 bits	1

5. Methods

The next subsection illustrates the testing environment by showing the test bed used server, the SIP traffic generator to SIP proxy, and the network elements specifications. By nature, each algorithm developed with different key size series for example AES uses 128, 192, and 256, whereas 3DES uses 168, 112, and 56 bits. Due to this various structure in each algorithm the test has considered only the largest Key size from each algorithm in order to test the highest effect from each one.

5.1. Test bed server

OpenSIPS [27, 28] server is used as testing bed platform for TLS cipher algorithms performance. OpenSIPS version 2.4.3 proxy is an open source freely-available server with high level of compatibility, reliability, connectivity, and fully TLS supported environment. TLS parameters are configured by OpenSIPS logic route and routing script. For better route tracing, OpenSIPS configured as a stateful proxy. To support *TLSv*1.2 protocol, OpenSIPS interoperates with OpenSSL server version 1.0.2*g* to support SIP with TLS libraries. Logically, all the servers involved in this test are running on the same machine (Ubuntu 16.4 server).

Several configurations have been applied to OpenSIPS server in order to meet all our testing scenarios. In particular, Diffie-Hellman and Elliptic Curve key exchange algorithm have been added into OpenSSL server individually by generating a *DH param* file using *openssl dhparam -out dhparam.pem* 4096. Later, OpenSIPS will refer to the generated file using parameter configuration. Also, we have made each call achieved individually through a new connection each time by opening a new port connection to avoid TLS resumption to same UAC even it has the same IP address. This is required for multiple connection mode between the UAC and UAS. We also made configuration regarding *NONCE* parameter that increasingly prevented SIPp client from generation high load test toward OpenSIPS proxy due to the limited initial value set in *MAX_NONCE_INDEX*. Manually we have increased the value from 100,000 to 1,000,000 to be sufficient for high load test.

5.2. SIP traffic generator

SIPp is a de facto standard tool utilized to generate SIP traffic toward SIP server [29]. SIPp clients are used as UAC and UAS individually in end-to-end devices. SIPp is used to generate SIP traffic from source UAC toward distention UAS. Besides, it has ability of message modification using high customized XML language. In this research SIPp is selected due to its ability to generate high load SIP traffic and deal with TLS parameters smoothly. SIPp in order to work with TLS, SIPp compiles by adding the *"--with-openssl*". Note that certificate and keys files (*user-cert.pem* and *user-privkey.pem* respectively) in the server side have to be added manually at the same directory of SIPp client [30].

5.3. Network elements

The testing environment has implemented in a local network, all machines are isolated on a 1000-Mbps to avoid exterior factors. SIP over TLS testing scenarios is implemented inside an OpenSIPS server, specifically SIP proxy. The proxy is connected with two legs connections which are TLS-over-TCP and TCP devices. The first leg connection from UAC to server is carried over TLS connection, whereas the

second leg, server to UAS is connected without TLS. SIP server is configured with TLS interface by listening to a separated port which is 5061. Table 2 provides a hardware specification for all devices involved in SIP testing over TLS cipher algorithms.

Table 2. Hardware specifications for the test bed.

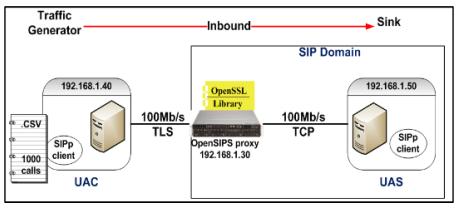
Entity	SIP proxy Server	UAC Machine	UAS Machine
Domain name/IP	IPv4 : 192.168.1.30	IPv4 : 192.168.1.40	IPv4 : 192.168.1.50
Machine Model	Dell-Vostro (PC)	Dell-Vostro Laptop	Lenovo B570e Laptop
Operating System	Ubuntu 16.04.5 server	Debian Stretch version 9	Debian Stretch version 9
Kernel	4.4.0 -131-generic	4.9..0-7-686-pae	4.9..0-7-686-pae
RAM	2 Giga Byte	2 Giga Byte	4 Giga Byte
CPU	3.00GHz (disabling other core processors)	900@2.20GHz	i3-2310M @ 2.10GHz
Openssl server	Version 1.0.2g	Version 1.0.2g	Version 1.0.2g

5.4. Testing scenario

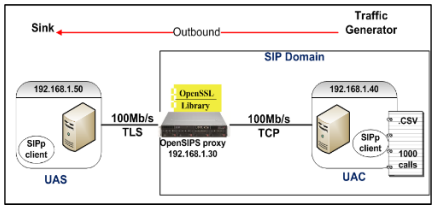
The workload is a SIP call generated from the SIPp client toward the SIP server. The workload is increased dramatically starting from 10 calls per second (cps) till the desired load 1000. This proxy runs under 5 bulk cipher algorithms and omitting the other cryptographic algorithms such as key exchange, signature, and message integrity. Testing is conducted based on 3 topologies which are inbound, multi-to-one, and outbound proxy by evaluating the following parameters:

- CPU utilization (using one processor and disabling other cores).
- Throughput (measures kbps).
- Call response time (round trip SIP message).

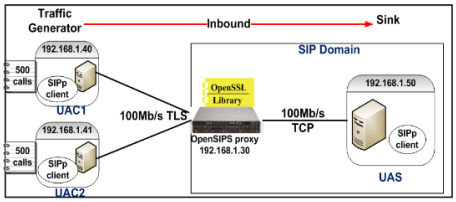
Initially, test runs for 1000 calls in prior to ramp up the server machine before the actual testing is taken. Also, results are obtained from the average of three consecutive test runs. Figure 3 shows testing topologies.



(a) Inbound.



(b) Outbound.



(c) Heterogeneous Clients (Multi-to-One).

Fig. 3. Test environment for SIP proxy.

6. Experimental Results

The results have been conducted from different topologies and configuration scenarios. The results obtained from different overheads caused by the heavy calculation of cipher algorithms. From which, there will be 12 tests to carry out during this research obtaining various results regarding the OpenSIPS proxy over different cipher algorithms.

6.1. Inbound proxy

Figure 4 shows the CPU usage percentage for the inbound proxy scenario using five bulk cipher algorithms. Each bar presents how much CPU usage each algorithm costs. It is obviously seen that the AES cipher algorithm cost the OpenSIPS server to perform with highest CPU utilization that is due to the fact that the AES requires high process for encryption by using 10 rounds with different 128-bit key which are originally obtained from the initial AES key. SEED and 3DES are both run with normal CPU usage to hit around 1.8%. When Camellia is used, OpenSIPS reports the lowest CPU usage while operating this algorithm, that is due to the normal four 8×8-bit S-boxes is used with input and output logical operation and affine transformations. The most significant CPU cost components is an AES algorithm, resulting higher security concern [31].

Throughput performed with opposite perspectives compared to CPU usage section. From Fig. 5, AES wins over other algorithms with a throughput 78.81% Kbps. 3DES performs at the fair position in both sections CPU and throughput that is because the normal key operation that been used in this algorithm [32]. With no doubt, RC4 algorithm processed with lower throughput in this section that is from the fact that the RC4 is a stream cipher. Stream cipher converts text by taking 1 byte (8 bits) of the plain text at a time while other algorithms (block ciphers) take the entire block (64 bits or more) at a time. In contrast to CPU usage, Camellia algorithm performs in satisfactory level to reach 78.79 Kbps which almost close to AES. As can be seen, the AES and the Camellia algorithms with inbound proxy scenario enjoy the most obvious boosts in throughput, by about 78.81 and 78.79 Kbps respectively.

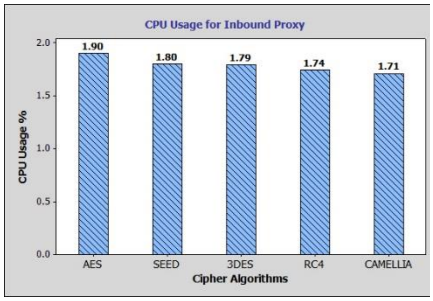


Fig. 4. CPU percentage usage for inbound proxy.

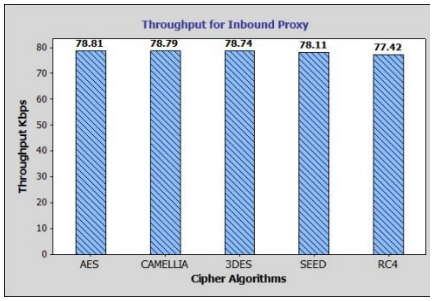


Fig. 5. Throughput measure for inbound proxy.

The high performance of Camellia is from the nature design of its logical operations and table lookups that aims to provide high-speed implementations on hardware and software systems. Other benchmark works have adopted same claim of Camellia performance. Aoki et al. [33] recommended Camellia in hardware implementation to occupy only 7.875 K gates using a 0.11 μm when using 128-bit block ciphers.

Regarding the Call Response Time (CRT) calculation, from Fig. 6 readers clearly notice that AES algorithm has longest response time from source to destination and back to source again. That is because again the AES requires high load processing to include 10 rounds with different 128-bit key, each round calculates its key from the initial key. Despite of Camellia perform well in CPU usage and throughput, it costs the OpenSIPS proxy to handle its calls with long response time to reach 30 ms. RC4 and 3DES are both perform in acceptable response time in the range of 20ms. Whereas SEED algorithm requires less response time to handle its calls with optimum value to reach 17ms which consider almost idle. Other researchers have also claimed the acceptable performance of RC4 such as in [34] as it uses two parallel outputs that however decrease the time consumption. Also, Jindal and Singh [26] have turned the attention to the additional byte generated from Key2 phase and xored which enhance the performance in term of time.

All the algorithms evaluated in this scenario are still in the range of acceptable response time which is 150ms (37, 30, 28, 22, and17 for AES, Camellia, RC4, 3DES, and SEED respectively), stated in ITU-T G.114 recommendation and SIP server benchmark [1]. Clearly can notice that SEED algorithm wins over other algorithms by accomplishing its call setup with fastest response time which in turn gain the privilege of using security with acceptable performance.

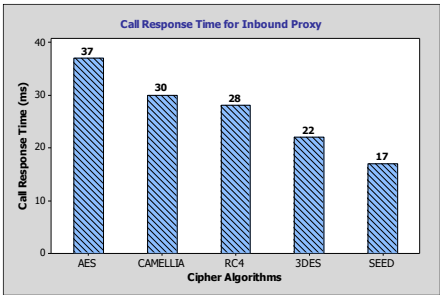


Fig. 6. Call response time for inbound proxy

6.2. Outbound proxy

In this scenario the OpenSIPS proxy acts as UAC where it has to establish TLS connection with the next hop (server or client) see Fig. 4. From Fig. 7, one can notice that the CPU usage in this scenario with AES is about 19% of that one from inbound scenario. As mentioned earlier that the AES is processed with long serious of operations to encrypt the text. However, at the previous section the AES used to decrypt the incoming messages but in this section the encryption is used. Camellia algorithm consumes low CPU sources compared to the other algorithms, 2.09% of CPU. RC4 again utilized low CPU cycles in outbound proxy that is due to the small code size which is in range of 1 KB. These measurements are not far from [35] results when they use the 8-bit processor Intel 8051: they claimed the speed-optimized when the key setup takes 43246 clock cycles compared to other setups.

An interesting observation from Fig. 7 is the cost of CPU consumption to all algorithms, which are substantially higher than in the previous scenario. This is from the fact that proxy in the outbound mode acts as TLS client and required to verify the certificates presented by the UAS [32], which was not present in the inbound mode.

Figure 8 present the OpenSIPS proxy throughput when the server acts as an outbound proxy. Clearly, RC4 algorithm comes up with low throughput which is 77.52 Kbps that is because the RC4 is a stream algorithm and OpenSIPS proxy in this section is sending SIP messages toward the client as 1 Byte each time. Nevertheless, this is not a drawback for this algorithm in all ways because throughput in this case is measured in 1 second beside RC4 performs well with CPU usage. We have to mention here that RC4 is considered as a low security algorithm and there is a recommendation to avoid this algorithm [26]. Interestingly, AES algorithm records sufficient throughput against other algorithms to hit 78.98 Kbps. That however made AES as a good choice algorithm to be used in SIP security using TLS. Camellia also performs in an optimum level of throughput with amount of data 78.89 kbps to be higher with 1 kb from the next algorithm, 3DES.

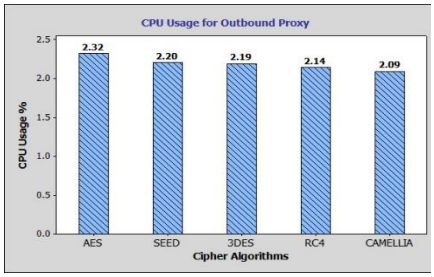


Fig. 7. CPU Percentage usage for outbound proxy.

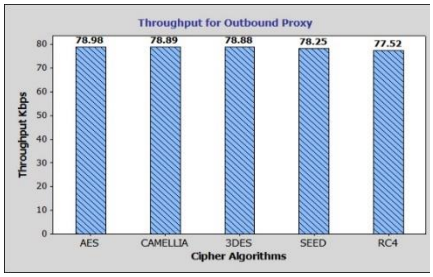


Fig. 8. Throughput measure for outbound proxy.

In this scenario the response time is calculated from the round trip starting from the OpenSIPS proxy toward the next hop then back track to proxy again. It is clearly notice that the AES registered the worst position in Fig. 9 that is due to the extra process calculations that been occurred in the encryption/decryption sessions.

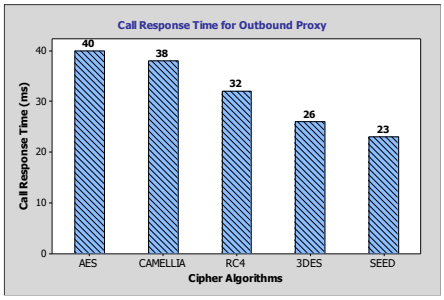


Fig. 9. Call response time for outbound proxy.

There is a (3 ms) of time differ between AES algorithm from the previous scenario associated to the key verification process that occur in the server side. Camellia is faster than AES in (2 ms) but is provide low performance in other calculations such as CPU usage and throughput. Unlike the CPU usage and throughput, response time with SEED algorithm records the best in this scenario which is faster than AES around 17 ms. Table 3 presents all the data collected from the testing evaluation.

Table 3. Outbound SIP proxy performance with three parameters.

Algorithms	CPU (%)	Throughput (Kbps)	CRT (ms)
AES	2.32	78.98	40
SEED	2.20	78.25	23
3DES	2.19	78.88	26
RC4	2.14	77.52	32
Camellia	2.09	78.89	38

6.3. Heterogeneous clients (Multi-to-One)

In this experiment, two different SIPp clients generate traffic toward OpenSIPS proxy. The aim of this experiment is to examine SIP server over TLS scalability by handling 1000 callers from different clients. The topology in this experiment has two source nodes (each generate 500 calls) and one destination node (sink) using TLS across five different cipher algorithms. Note that both clients are sending the traffics simultaneously. In this scenario, Call Response Time is omitted and cannot be calculated from two different SIPp clients.

Figure 11 shows the CPU usage of the OpenSIPS proxy for 5 variant cipher algorithms. Clearly we can notice that the SEED algorithm cause OpenSIPS proxy to use 3.28% of CPU, the extra usage is come from the multi-TLS threat established between the two clients (source) and the OpenSIPS proxy (destination). In this scenario SIP proxy have to manage multi-TCP sockets to establish TLS connection individually. AES has added additional 1.21% of CPU usage to SIP proxy over the one in inbound scenario; however, the said extra CPU usage is to manage the extra connection opened between end-to-end devices. Regarding to Camellia algorithm, it still consumes lowest CPU among other algorithms to hit 3.07%. 3DES copes with this scaling in a sufficient way by improving its position comparing to Fig. 4.

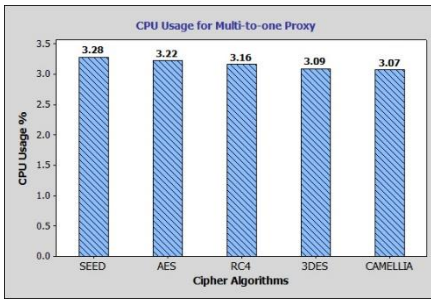


Fig. 10. CPU Percentage usage for multi-to-one proxy.

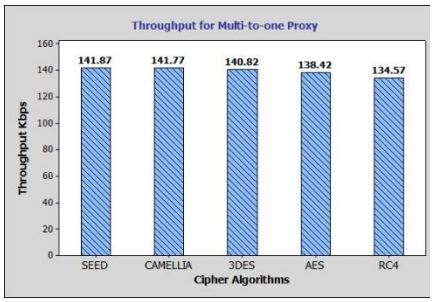


Fig. 11. Throughput measure for multi-to-one proxy.

It is fair to say that the SIP server throughput with two clients behaves like inbound scenario with extra bit of data associated to the connectivity setup from two devices. For instance, RC4 algorithm caused SIP proxy to receive calls extra data to reach 134.57 Kbps compared to 77.42 Kbps from one client scenario. From here we can conclude that two clients do not mean double of throughput amount. The extra amount is come from the TCP initial handshake for connection establishment. Worth to note that SIP proxy degrades its performance with AES algorithm when two scaling client is considered. AES in this scenario performs with 138.42 Kbps whereas SEED algorithm causes the proxy to gain 141.87 Kbps

due to the simple calculated operation for its encryption/decryption. From the scaling scenario there are several findings arisen here, throughput do not double by client increasing nor do algorithms behave same with single threat.

7. Conclusion

Due to the ever-increasing demand of securing SIP communications, SIP service providers relay on TLS as a transport protocol to transmit SIP signalling messages securely. Several researches have been conducted on TLS protocol concentrating on security level; however, insufficient researches have been done on performance impact.

This paper studied the impact of deploying TLS cipher algorithms over SIP server. OpenSIPS server is chosen as a SIP proxy along with OpenSSL to support TLS library. From TLS, five bulk cipher algorithms are studied in this research to show how much each algorithm cost SIP server in term of CPU usage, Throughput, and Call Response Time.

In term of experiment, OpenSIPS proxy is implemented over three different scenarios each of which provides variant result finding. Through the experiments, TLS utilized one set of cipher suite string includes RSA for key exchange algorithm and SHA for message integration and authentication. AES cipher algorithm cost OpenSIPS proxy 1.9% of CPU usage (which is the highest among other algorithms) due to the fact that AES requires high process to encrypt SIP messages by using 10 rounds with different 128-bit key each time. However, Camellia algorithm performs out of the rest algorithm to register only 1.71% of SIP server's CPU.

Through all scenarios, RC4 witnessed to record insufficient throughput over all experiments because it is known to be stream cipher which is in fact converts text by encrypting 1 byte (8 bits) of the plain text at a time. Nevertheless, AES consume high CPU usage, it can perform great when throughput is considered when inbound proxy scenario is used. While for outbound scenario, SIP proxy acts as TLS client and it required to verify TLS connection certificates presented by the UAS. This as an extra process caused the proxy to add negligible amount of CPU usage and throughput data over other scenarios.

Finally, heterogeneous client scenario shows that SEED algorithm causes the proxy to gain 141.87 kbps which is higher from one client due to the extra TCP connection setup and socket management for each thread. Out of security talk, AES is somewhat performed better over other algorithms especially in throughput. For devices with limited/low resources such as CPU, Camellia algorithm will be a proper choice as it records ever lower CPU usage through the three scenarios. Future direction of this research can curve toward SIP registration with authentication using the one/all algorithms.

Nomenclatures

F	Function
Ki	Sub-key
V	Version

Abbreviations

3DES	Triple Data Encryption Standard

AES	Advanced Encryption Standard
CA	Certification Authority
CPS	Call Per Second
CPU	Central Processing Unit
CRT	Call Response Time
DH	Diffie-Hellman
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol Version 4
ITU	International Telecommunication Union
Kbps	Kilo Bit Per Second
MAC	Message Authentication Code
Ms	Millisecond
PKC	Public Key Cryptography
QoS	Quality of Service
RC4	Rivest Cipher 4
RTP	Real Time Protocol
S-Box	Substitution Box
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SIPp	Session Initiation Protocol Performance
SRTP	Secure Real-Time Transport Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
XML	Extensible Markup Language

References

1. Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A.; Peterson, J.; and Sparks, R. (2002). SIP: Session initiation protocol. RFC 3261, Retrieved March 22, 2018, from https://tools.ietf.org/html/rfc3261 .
2. Boucadair, M.; Kaplan, H.; Gilman, R.; and Veikkolainen, S. (2013). The session description protocol (SDP) alternate connectivity (ALTC) attribute. RFC 6947, Retrieved March 22, 2018, from https://tools.ietf.org/html/rfc6947.
3. Ali-Ahmad, H.; Munir, K.; Bertin, P.; Guillouard, K.; Ouzzif, M.; and Lagrange, X. (2016). Processing loads analysis of distributed mobility management and SIP-based reachability. *Telecommunication Systems*, 63(4), 681-696.
4. Shen, C.; Nahum, E.; Schulzrinne, H.; and Wright, C. (2012). The impact of TLS on SIP server performance: Measurement and modeling. *IEEE/ACM Transactions on Networking*, 20(4), 1217-1230.
5. Kaji, T.; Hoshino, K.; Fujishiro, T.; Takata, O.; Yato, A.; Takeuchi, K.; and Tezuka, S. (2006). TLS handshake method based on SIP. *Proceedings of the International Multiconference on Computer Science and Information Technology*. Wisla, Poland, 467-475.

6. Ring, J.; Choo, K.; Foo, E.; and Looi, M. (2006). A new authentication mechanism and key agreement protocol for SIP using identity-based cryptography. *Proceedings of Asia Pacific Information Technology Security Conference* (*AusCERT*). Gold Coast, Australia, 57-72.
7. Delignat-Lavaud, A.; Fournet, C.; Kohlweiss, M.; Protzenko, J.; Rastogi, A.; Swamy, N.; Zanella-Béguelin, S.; Bhargavan, K.; Pan, J.; and Zinzindohoue, J. (2017). Implementing and proving the TLS 1.3 record layer. *Proceedings of IEEE Symposium on Security and Privacy*. California, USA, 463-482.
8. Khudher, A.A. (2019). SIP aspect of IPv6 transitions: current issues and future directions. *Journal of Engineering Science and Technology* (*JESTEC*), 14(1), 448-463.
9. Shen, C.; Nahum, E.; Schulzrinne, H.; and Wright, C. (2010). The impact of TLS on SIP server performance. *In Principles, Systems and Applications of IP Telecommunications*, 59-70.
10. Shen, C.; Nahum, E.; Schulzrinne, H.; and Wright, C. (2012). The impact of TLS on SIP server performance: Measurement and modeling. *IEEE/ACM Transactions on Networking*, 20(4), 1217-1230.
11. Müller, S.; Bermbach, D.; Tai, S.; and Pallas, F. (2014). Benchmarking the performance impact of transport layer security in cloud database systems. *Proceedings of IEEE International Conference on Cloud Engineering*, Bosten, USA, 27-36.
12. Adrianto, D.; and Lin, F. (2015). Analysis of security protocols and corresponding cipher suites in ETSI M2M standards. *Proceedings of IEEE* 2*nd World Forum on Internet of Things* (*WF-IoT*), Milan, Italy, 777-782.
13. Zhao, L.; Iyer, R.; Makineni, S.; and Bhuyan, L. (2005). Anatomy and performance of SSL processing. *In IEEE International Symposium on Performance Analysis of Systems and Software*, Texas, USA, 197-206.
14. Khudher, A.A.; Aboalmaaly, M.F.; and Naeem, A.N. (2013). Telephone number addressing for SIP peering within inter-domain voice communication. *Journal of Advances in Information Sciences and Service Sciences*, 5(11), 178-189.
15. Khudher, A.A.; and Ramadass, S. (2015). I-tnt: phone number expansion and translation system for managing interconnectivity addressing in sip peering. *Journal of Engineering Science and Technology* (*JESTEC*), 10(2), 174-183.
16. Keerthi, V.K. (2016). Taxonomy of SSL/TLS attacks. *International Journal of Computer Network and Information Security*, 8(2), 15-24.
17. Dierks, T.; and Christopher A. (1999). The TLS protocol version 1.0. RFC 2246, Retrieved March 22, 2020, from https://tools.ietf.org/html/rfc2246.
18. Han, S.; Kwon, H.; Hahn, C.; Koo, D.; and Hur, J. (2016). A survey on MITM and its countermeasures in the TLS handshake protocol. *Proceedings of* 8th *International Conference on Ubiquitous and Future Networks*, (*ICUFN*), Vienna, Austria, 724-729.
19. Cai, J.; Huang, X.; Zhang, J.; Zhao, J.; Lei, Y.; Liu, D.; and Ma, X. (2018). A handshake protocol with unbalanced cost for wireless updating. *IEEE Access*, 6, 18570-18581.
20. Abualhaj, M.; Al-tahrawi, M.; and Al-khatib, S. (2019). Performance evaluation of voip systems in cloud computing. *Journal of Engineering Science and Technology* (*JESTEC*), 14(3), 1398-1405.

21. Ismoyo, D.; and Wardhani, R. (2016). Block cipher and stream cipher algorithm performance comparison in a personal VPN gateway. *Proceedings of International Seminar on Application for Technology of Information and Communication* (*ISemantic*), Semarang, Indonesia, 207-210.
22. Thakur, J.; and Kumar, N. (2011). DES, AES and blowfish: symmetric key cryptography algorithms simulation based performance analysis. *International Journal of Emerging Technology and Advanced Engineering*, 1(2), 6-12.
23. Koko, S.; and Babiker, A. (2015). Comparison of various encryption algorithms and techniques for improving secured data communication. *Journal of Computer Engineering*, 17(1), 62-69.
24. Lu, J.; Wei, Y.; Fouque, P.; and Kim, J. (2012). Cryptanalysis of reduced versions of the camellia block cipher. *IET Information Security*, 6(3), 228-238.
25. Kitsos, P.; and Skodras, A. (2011). An FPGA implementation and performance evaluation of the seed block cipher. *Proceedings of* 17*th International Conference on Digital Signal Processing* (*DSP*), Corfu, Greece, 1-5.
26. Jindal, P.; and Singh, B. (2017). Optimization of the security-performance tradeoff in RC4 encryption algorithm. *Wireless Personal Communications*, 92(3), 1221-1250.
27. Bogdan A. (2020). OpenSIPS the new breed of communication engine. Retreived August 8, 2020, from https://opensips.org/.
28. Goncalves, F.E.; and Iancu, B.A. (2016). *Building telephony systems with OpenSIPS* (2nd ed.). Packt Publishing Ltd. Birmingham, United Kingdom.
29. Richard G. (2020). SIPp Open Source traffic generator for the SIP protocol. Retreived August 10, 2020, from http://sipp.sourceforge.net/.
30. Khudher, A.A.; Beng, L.Y.; and Ramadass, S. (2013). A comparative study of direct and indirect static peering for inter-domain SIP calls. *Proceedings of IEEE International Conference on RFID-Technologies Application*. Johor, Malaysia, 1-5.
31. Leau, Y.; Khudher, A.A.; Manickam, S.; and Al-Salem, S. (2017). An adaptive assessment and prediction mechanism in network security situation awareness. *Journal of Computer Science*, 13(5), 114-129.
32. Praptodiyono, S.; Santoso, M.; Firmansyah, T.; Abdurrazaq, A.; Hasbullah, I. H.; and Osman, A. (2019). Enhancing IPsec performance in mobile IPv6 using elliptic curve cryptography. *Proceedings of* 6*th International Conference on Electrical Engineering, Computer Science and Informatics* (*EECSI*), Bandung, Indonesia, 186-191.
33. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., And Tokita, T. (2001). Camellia: A 128-Bit Block cipher suitable for multiple platforms. *In Proceedings of the Selected Areas in Cryptography* (*SAC'*00), *D. Stinson and S. Tavares*, Springer-Verlag, 39-56.
34. Worley, J., Worley, B., Christian, T., And Worley, C. (2001). AES Finalists on PA-RISC and IA-64:Implementations & performance. *In Proceedings of the* 3*rd AES Conference* (*AES*3), 57-74.
35. Hachez, G., Koeune, F., And Quisquater, J.-J. (1999). cAESar results: Implementation of four AES candidates on two smart cards. *In* 2*nd AES Candidate Conference* (*AES*2), 95-108.