The SSL Hardware Acceleration for Rota Model and Cryptographic Approach

Prof. Kavitha S Patil[1], Dr. Indrajit Mandal[2], Dr Seetharam K[3]
[1]*Assistant Professor: ISE, Atria Institute of Technology, Bangalore, India*
[2]*Scientist, TCS, Bangalore, India*
[3]*Professor, CSE, Bangalore, India*

***Abstract*- Based on data sensitivity over the networks at transport layer, deploying the cryptographic[24] approach by accelerating the SSL[15][11] is one the most complicated area, over the years this issue of SSL upgration and at TLS[4][5][22] is getting problematic for proper metrics. Coming to cryptography and security[24] the repetition of traditional approaches is into the picture to broad cast the data based on the discovered LOOK UP table[23]. So, to overcome the challenges this work discovered a new model DSSETA(Dynamic Secure Socket Layer Enability over Transport layer with Acceleration). For security this work discovered the ROTA (Regex)[base64] algorithm for encryption and decryption. Accessing of the LOOK UP table[23] is with a new and innovative approach based on the kernel level interfaces with neighboring and with routing addresses and with valid metric access dynamically which can be enabled for broadcast of data with valid key break through approach. ROTA[base64] is a kind of REGEX model of approach which will change the 6 bits break through to fully qualify to pad dynamic character(s) to have satisfactory bit quantization for decryption**

Keywords: **Look up table, encryption, SSL,TLS, routing, cryptography, metrics**

I.INTRODUCTION

In considering with the nature of SSL/TLS [4][5][22] matches which gives extension and backward adoptability between various versions. So, by deploying the various cryptographic approaches[24] with the same versions of SSL/TLS [4][5][22] will lead to damage to transport of the data with loss of sensitive information. So by maintaining proper and valid LOOK UP table[23] which should accelerate the migration of versions with the current metrics with various environments. DSSETA will acts as a controller for migration of versions at transport layer by giving proper acceleration model to enable and disable with respect to various parametric and with proper attributes. So by enabling this kind of acceleration, the security[24] will be increased with auto initiative mechanism. First DSSETA will starts fetching the hardware offset of the application layer(for sample :X47F0Fxxx) by targeting the proper DLL with current environment(normal OS or VMware, cloud era) and starts controlling the all the layer till session layer which is pre layer to transport layer. So by enabling the proper LOOK UP[23] metrics with the current parameters the security[24] deployment begins with SSL[15][11] enability. By putting the TLS[4][5][22] with SSL now the access to data will be flown in the cryptographic[24] way. Currently TLS(Transport layer security) [4][5][22] is trend and predominant way to achieve the internet security. So DSSETA will starts to accelerate the cryptographic security[24] by enabling the SSL[15][11] at TLS[4][5][22] with iterative LOOK UP table[23] access. The ROTA(Regex)[base64] security is enabled after the acceleration. Rota[base64] is a model of approach to break the data into 6-bit framing with neighboring bits as partner. With odd ends data this approach will keep putting "=" to satisfy the 6-bit framing partners.

Architecture:

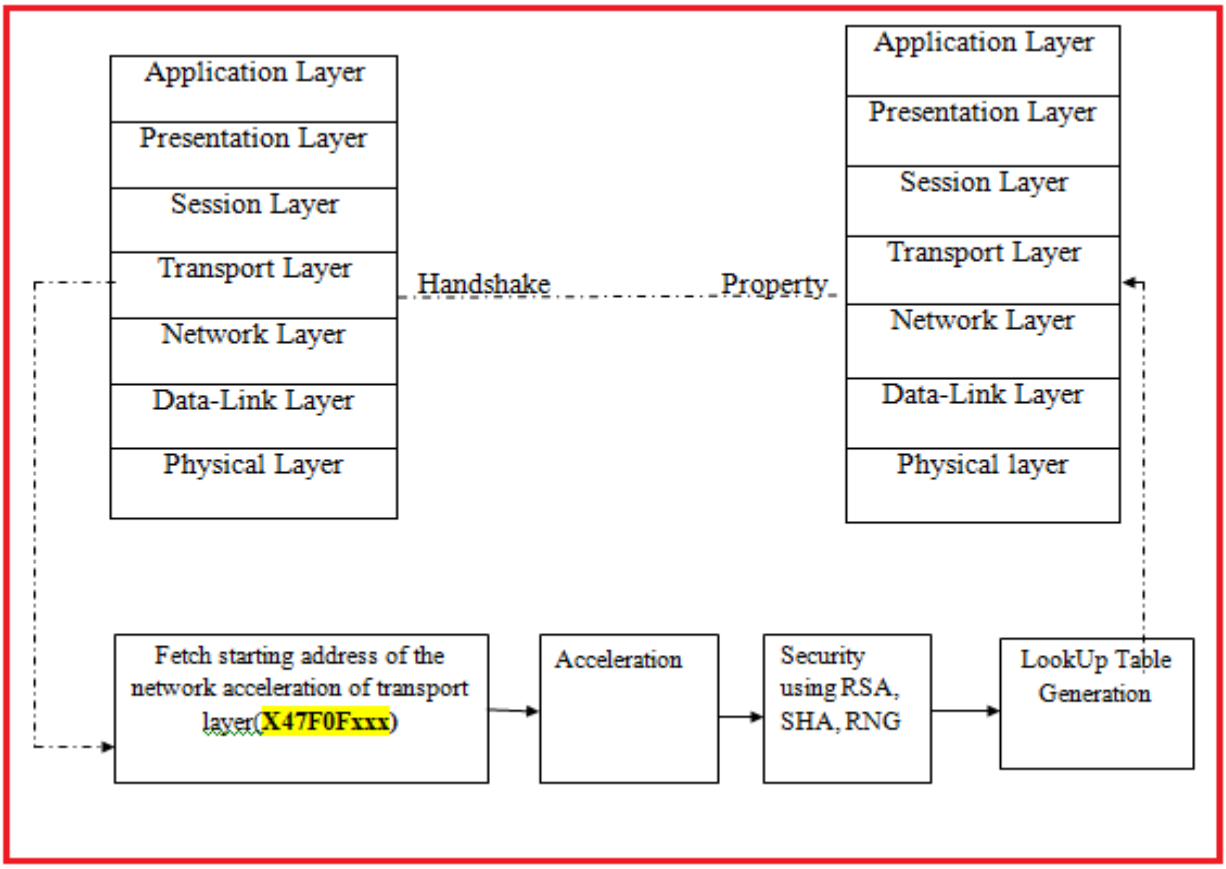


Fig 1:Integrated Architecture Containg three Approaches

II LITERATURE SURVEY AND BACKGROUND ON SSL/TLS

The protocol was first developed by Netscape Communications when Mosaic was released in 1993. In 1994 the first version was finished but due to major flaws in the protocol it was only used internally and never was released. Later that same year SSL[15][11] was improved upon and launched with the web browser Netscape Navigator as SSL version 2. Initially SSL[15][11] was patent protected but the patent was given away and made free to use. SSL[15][11] 3.0 was released in 1995 in order to correct security issues present in SSL 2.0 . For example, complementing Message-Digest algorithm 5 (MD5)[25] with Secure Hash Algorithm 1 (SHA-1)[25] as the cryptographic hash function. In parallel to Netscape developing and releasing SSL[15][11], Microsoft had been working on their own version called Private Communication Technology (PCT). Even though interoperability with SSL[15][11] was supported, when another protocol called Secure Transport Layer Protocol (STLP)[26] was proposed it was clear that a single standardized protocol was needed. The Internet Engineering Task Force (IETF)[4][5] set up a work group dedicated to this and from that point the protocol would be known as TLS[4][5][22].

Turner and Polk on behalf of IETF[4][5] ruled SSL[15][11] 2.0 not secure enough and that TLS[4][5][22] never should negotiate use of SSL[15][11] 2.0. Usage of MD5[25], no protection of handshake messages, the message integrity and encryption using the same key, and weakness in the session making it easy to terminate by a man-in-the-middle was named the major reasons. In June 2015 Barnes, Thomson, Pironti, and Langley deprecated the SSL[15][11] 3.0 protocol as well, stating that it must not be used. No suitable record protection mechanism, key exchange vulnerability during renegotiation and session resumption, cryptographic[24] primitives relying on SHA-1 and MD5[25], and the inability to

adapt new features from newer protocols was named the major reasons. None of these flaws are present in any of the TLS[4][5][22] implementations.

The improved TLS[4][5][22] 1.0 was released but was still very similar to SSL[15][11] 3.0 and could basically be view as SSL 3.1 but under a single standardized protocol. A much bigger update was made when TLS[4][5][22] 1.1 was released in 2006. One of the big changes was the change to the Initialization Vector (IV) making it explicit instead of implicit. This is to protect against Cipher Block Chaining (CBC) attacks Just two years later in 2008 the TLS 1.2 was released. The MD5/SHA-1[25] combination in both the Pseudorandom Function (PRF) and digitally signed element was replaced with a single hash. Also, the extensions definition and Advanced Encryption Standard (AES) cipher suites being merged in are named major differences.

A.Flow:

Algorithms and pseudo codes:

SSL Secure Enablity with hand shaking:

SSL[15][11] (Secure Sockets Layer) or more correctly TLS[4][5][22] (Transport Layer Security) is an important component in the secure delivery of web applications. It provides for authentication (website to client and optionally client to website) and protects the traffic between clients and sites using encryption.

However, this protection comes at a cost as the computational overhead involved in setting up each client session is significant. Using a load balancer to offload the SSL[15][11] processing removes this overhead from the web servers, hardware acceleration shown in the Fig 2 and frees up resources for web application related tasks.

Fig 2: SSL Acceleration

SSL[15][11] protocol, does its fantastic job of securing communication over the wire, with the help of multiple layers of protocols, above TCP(And After Application Layer).Always keep in mind that, although HTTP protocol is the protocol, which highly makes use of SSL, to secure communication. SSL[15][11] is an application layer independent protocol shown in the Fig 3. So you can use that with any application layer Protocol.

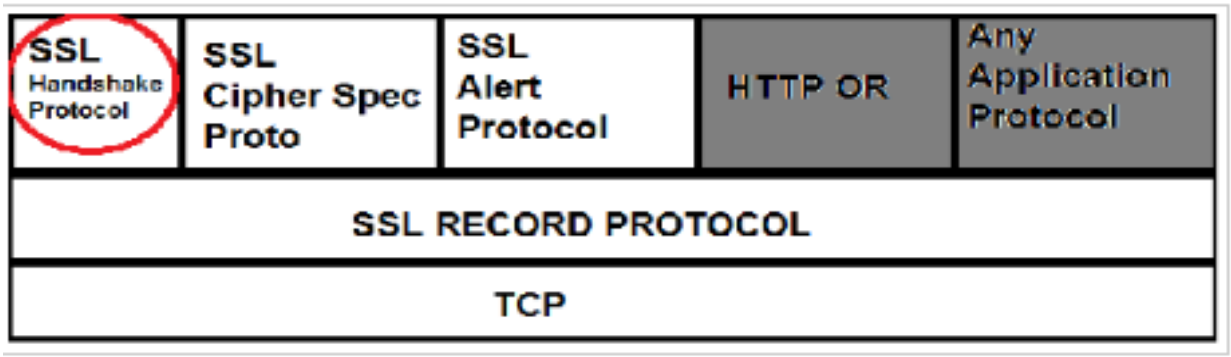


Fig 3: Layer representation of SSL Handshake Protocol

Client Hello message content in SSL/TLS
Fig 4 shows the SSL handshake protocol where following steps will be done

SSL VERSION NUMBER : the client sends a list of ssl version it supports. And priority is given to the highest version it supports

Random Data Number : Its made up of 32 bytes. 4 byte number made up from client's date & time plus 28 byte randomly generated number(this will be used with server's random value made of date & time for generating the "master secret", from which encryption key will be derived).

SESSION ID: In order to enable client's resuming capabilities this session ID is included.

CIPHER SUITS: RSA[27] algorithm is used for the initial key exchange which will be done using public key cryptography. And SHA[25] is used for MAC and hashing. And also sends the encryption algo's supported by the client like DES for example.

Compression Algorithm: this will include compression algorithms details, if used.

Server Hello message in SSL/TLS
Version Number: Server selects an ssl version thats supported by both the server and the client, and is the highest version supported by both of them

Random Data: the server also generates a random value using the server's date and time plus a random number of 28bytes. Client will use this random value and its own random value to generate the "master key"

Sesssion ID: There are three possiblities, with regard to the session id. It all depends on the type of client-hello message. If the client requires to resume a previously created session, then both the client and server will use the same session ID. But, if the client is initiating a new session, the server will send a new session ID. Sometimes a null session ID is also used, where server will never support resuming the session, so no session id's are used at all.
Cipher Suits: Similar to the version number selected by the server, the server will select the best cipher suite version supported by both of them.

Certificate: The server also sends a certificate, which is signed and verified by a Certificate Authority, along with the public key(Content encrypted with public key can only be opened with a corresponding private key. In this case, only the server can unlock it because, the server has the private key for its public key).
A certificate signed by a certificate authority(a trusted third party), consists the complete information about the company using that certificate. The certificate identity of many well known certificate authority is made avialable to the web browser. Whenever a certificate is received by the client's browser, it is verified with the one it has from the certificate authority. So this proves that, that the server which claims, that it is "example.com" is infact correct.

Server Key Exchange: this step is taken by the server, only when there is no public key shared along with the certificate. If this key is used, this will be used to encrypt the "Client Key Exchange Method"

Client Certificate request: This is seldom used, because this is only used, when the client also needs to get authenticated, by a client certificate.

Server Hello Done: this message from the server will tell the client, that the server has finished sending its hello message, and is waiting for a response from the client.

Response from the client to server's hello message:
Client Certificate: The client sends a client certificate back to the server. This step is only used when a client certificate is requested by the server(through the server hello message).

Client Key Exchange: This message is only sent, after the client calculates, the premaster secret with the help of the random values of both the server and the client(Which was shared by both the server and the client through the hello message).
"Client Key exchange" message, is sent by encrypting it with the server's public key, which was shared through the hello message. This message can only be decrypted with the server's private key. If successful, the server is authenticated.

The above explained things is depicted in the below diagram.

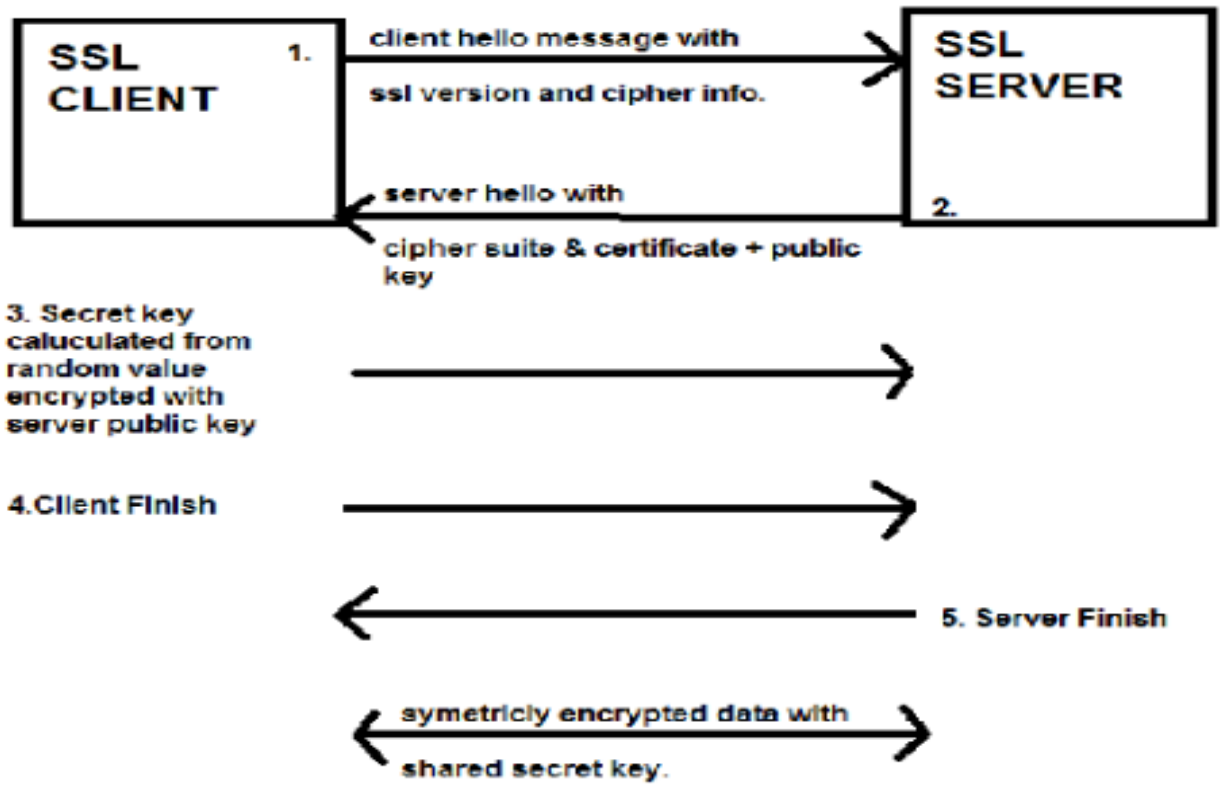


Fig 4: SSL Handshake

the client will also send the SSL[15][11] protocol version once again along with the "client key exchange" method, so that the server can verify, this version with the previous one send, so as to prevent a man in the middle from changing the protocol version.

Mathematical model of SSL Hand shaking:

Client Hello

TLS[4][5][22] wraps all traffic in “records” of different types. We see that the first byte out of our browser is the hex byte 0x16 = 22 which means that this is a “handshake” records shown in the Fig 5

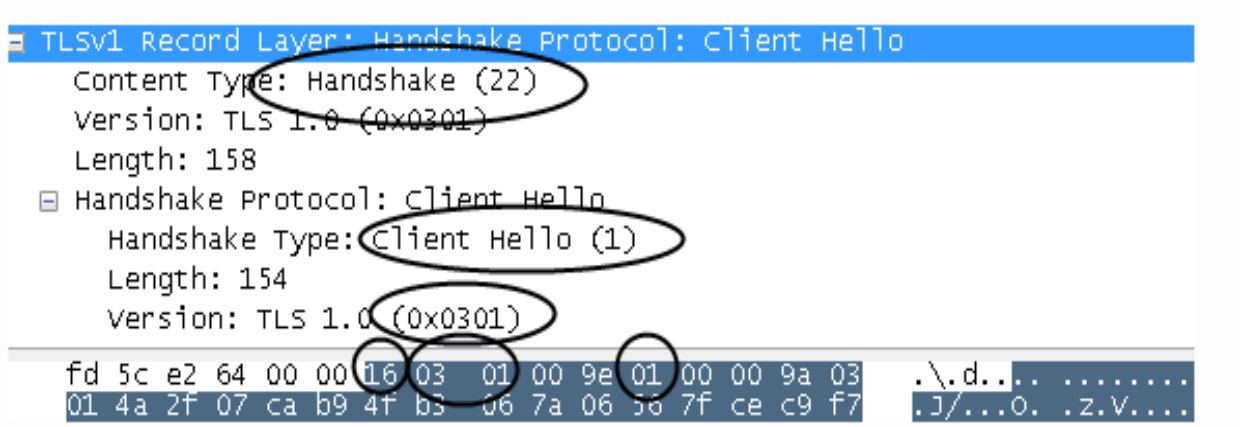


Fig 5: Handshake Record

Server Hello

Server replies with a handshake record that’s a massive two packets in size (2,551 bytes). The record has version bytes of 0x0301 meaning that Amazon agreed to our request to use TLS[4][5][22] 1.0. This record has three sub-messages with some interesting data shown in the Fig 6:

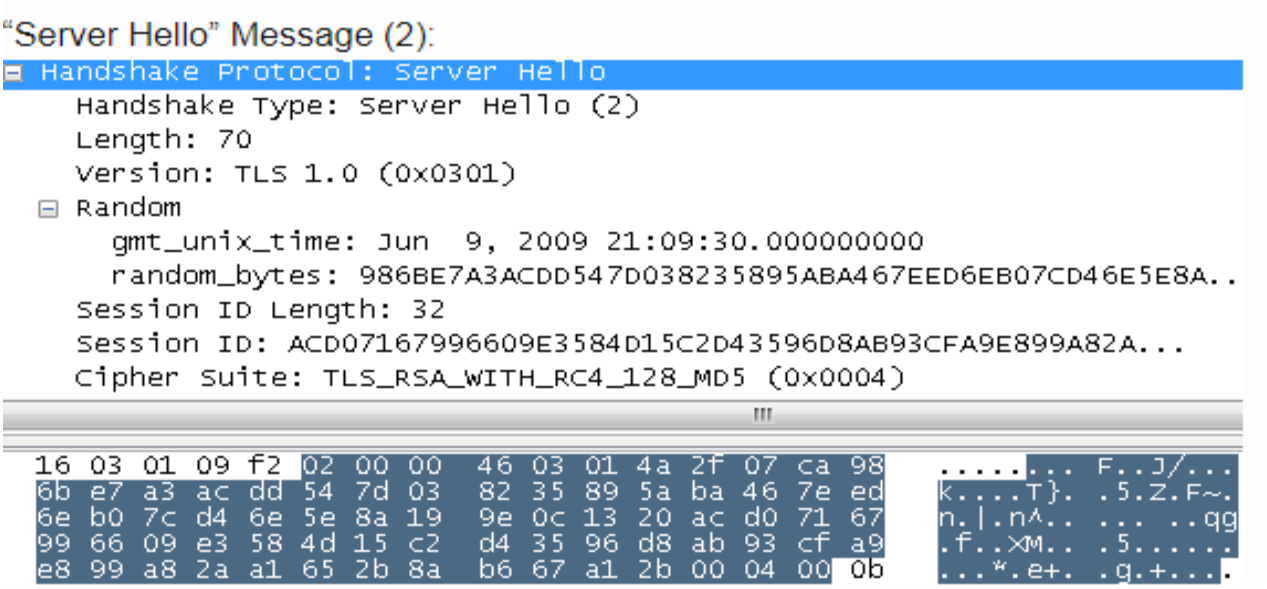


Fig 6: Server Hello Message

SSL handshaking signature verification:

People sometimes wonder if math has any relevance to programming. Certificates give a very practical example of applied math. Amazon's certificate tells us that we should use the RSA[27] algorithm to check the signature. RSA[27] was created in the 1970's by MIT professors Ron Rivest, Adi Shamir, and Len Adleman who found a clever way to combine ideas spanning 2000 years of math

development to come up with a beautifully simple algorithm:

You pick two huge prime numbers "p" and "q." Multiply them to get "n = p*q." Next, you pick a small public exponent "e" which is the "encryption exponent" and a specially crafted inverse of "e" called "d" as the "decryption exponent." You then make "n" and "e" public and keep "d" as secret as you possibly can and then throw away "p" and "q" (or keep them as secret as "d"). It's really important to remember that "e" and "d" are inverses of each other.

Now, if you have some message, you just need to interpret its bytes as a number "M." If you want to "encrypt" a message to create a "ciphertext", you'd calculate:

C ≡ Me (mod n)

This means that you multiply "M" by itself "e" times. The "mod n" means that we only take the remainder (e.g. "modulus") when dividing by "n." For example, 11 AM + 3 hours ≡ 2 (PM) (mod 12 hours). The recipient knows "d" which allows them to invert the message to recover the original message:

Cd ≡ (Me)d ≡ Me*d ≡ M1 ≡ M (mod n)

Just as interesting is that the person with "d" can "sign" a document by raising a message "M" to the "d" exponent:

Md ≡ S (mod n)

This works because "signer" makes public "S", "M", "e", and "n." Anyone can verify the signature "S" with a simple calculation:

Se ≡ (Md)e ≡ Md*e ≡ Me*d ≡ M1 ≡ M (mod n)

Public key cryptography algorithms like RSA[27] are often called "asymmetric" algorithms because the encryption key (in our case, "e") is not equal to (e.g. "symmetric" with) the decryption key "d". Reducing everything "mod n" makes it impossible to use the easy techniques that we're used to such as normal logarithms. The magic of RSA[27] works because you can calculate/encrypt C ≡ Me (mod n) very quickly, but it is really hard to calculate/decrypt Cd ≡ M (mod n) without knowing "d." As we saw earlier, "d" is derived from factoring "n" back to its "p" and "q", which is a tough problem.

Rota(Regex): This is a set of identical binary to strings(text) encoded models that indicates binary data with ASCII text format by converting/translating to RADIX-64 notations.

Normally Rota[base64] models are widely used whenever there is a need of binary data encoding that needed to be encoded and transferred over the

networks which are framed to deal with string based data. This step is to ensure that data has to remain without alteration during broadcast. Rota[base64] is ordinarily utilized as a part of a Rota[base64] encoding takes the first double information and works on it by partitioning it into tokens of three bytes. A byte comprises of eight bits, so Rota[base64] takes 24bits altogether. These 3 bytes are then changed over into four printable characters from the ASCII standard.

The calculation's name Rota[base64] originates from the utilization of these 64 ASCII characters. The ASCII characters utilized for Rota[base64] are the numbers 0-9, the letter sets 26 lowercase and 26 capitalized characters in addition to two additional characters "+" and '/'.

The initial step is to take the three bytes (24bit) of parallel information and split it into four quantities of six bits. Since the ASCII standard characterizes the utilization of seven bits, Rota[base64] just uses 6 bits (comparing to 2^6 = 64 characters) to guarantee the encoded information is printable and none of the uncommon characters accessible in ASCII are utilized. The ASCII change of 3-byte, 24-bit gatherings is rehashed until the entire succession of unique information bytes is encoded. To guarantee the encoded information can be appropriately printed and does not surpass the farthest point.

At the point when the quantity of bytes to encode is not distinct by 3 (that is, if there are just a single or two bytes of contribution for the last 24-bit square), then the accompanying activity is performed: Add additional bytes with esteem zero so there are three bytes and play out the change to base64. On the off chance that there was just a single huge information byte, just the initial two Rota[base64] digits are picked (12 bits), and if there were two critical info bytes, the initial three Rota[base64] digits are picked (18 bits). "=" characters may be added to make the last piece contain four base64 characters.

Example:

Text content	***M***		***a***	***n***
ASCII	***77***		***97***	***110***
Bit pattern	***0 1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0***			
Index	***19***	***22***	***5***	***46***
Base64-encoded	***T***	***W***	***F***	***u***

Padding:

The '==' sequence indicates that the last group contained only 1 byte, and '=' indicates that it contained 2 bytes.

Exapmple1:

Input:
any carnal pleasure.
Output: YW55IGNhcm5hbCBwbGVhc3VyZS4=

Exapmple2:

Input:
any carnal pleasure
Output:
YW55IGNhcm5hbCBwbGVhc3VyZQ==

Rota (encryption) algorithm:
Input → raw string
Output → base64 encoded format
Step1: initialization
∫ALPHABET = $\sum_0^{25} CAPS(65-90)$ + $\sum_0^{25} SMALL(97-122)$ +
$\sum_0^{9} NUMBERS(48-57)$ + $\sum_0^{2} SPC(43,67)$
// all "ALPHABET" CONTAINS ASCII values for capital letters, small letters, single digit numbers, '+' and '/' characters.

Step2:
Functionality:
To convert alphabets to ASCII codes
Input ← all available characters
Output ← all equivalent ASCII values
n ← 0
B0 ← 0
B1 ← 0
B2 ← 0
$\sum_0^n buff[]$ ← 0
$\sum_0^n ar$ ← 0
i ← 0
Iteration ← 0
Loop statement:
Count=0
For each C in ALPHABET
toInt (count) = TOINT (ALPHABET (i))
Count++
End loop

Size=SIZE (buff)
Iteration ← (((size+2)/3)*4)
Do while n in iteration
B0 ← buff
B1 ← (i <size)? buff++: 0
B2 ← (i < size)? buff++: 0
Masking
Mask=0X3F
ar = ALPHABET [(b0>>2) & mask]
ar= ALPHABET [(b0<<4) | ((b1&0XFF)>>4)) & mask]
ar= ALPHABET [(b1<<2) | ((b2&0XFF)>>6)) & mask]
ar = ALPHABET [b2 & mask]
End while
Padding:
If (size % 3=1)
ar ← "="
ar ← "="
Else if (size % 3=2)
ar ← "="
else
size %3=0
End if

Decryption:
When decoding Rota[base64] Final text, 4 characters are typically converted back to 3 bytes. The only exceptions are when padding characters exist. A single '=' indicates that the 4 characters will decode to only 2 bytes, while 2 '='s indicates that the 4 characters will decode to only a single byte.
Example:
Input:
YW55IGNhcm5hbCBwbGVhcw==
Block with 2 '='s decodes to 1 character:

Output:
any carnal pleas

RotalFinal (decoding) algorithm:
Input ← string
Output ← $\sum_0^n buff(bytes)$

Initialization:
Buff ← 0
S ← string decode
N ← length of string
Mask ← 0XFF
If s [0] = '=='
Delta =2
Else if s [0] = '='
Delta =1
Else
Delta=0
End if
Loop
For I ← 0 step by 4 of in n
C0=Convert to Int [CharAt (i) in S]
C1=Convert to Int [CharAt (i+1) in S]
$buffer_i$ ← (c0<<2) | (c1>>4) & mask
C2 ← Convert to Int [CharAt (i+2) in S]
$buffer_{i+1}$ ← (c1<<4)|(c2>>2)&mask
C3=convert toInt [i+3]
$buffer_{i++}$ = (c2<<6)! c3&mask
End loop

B Lookup table with Dynamic Secure Socket Layer Enability over Transport layer with Acceleration(FPGA):

A FPGA is an integrated circuit containing logic blocks such as look-up tables (LUT) [23] and flip-flops. As opposed to application specific integrated circuit (ASIC), the FPGA[28] can be programmed by the user to realize custom-designed logic. Programming is accomplished using a hardware descriptive language (HDL), such as VHDL or Verilog. In the labs you have previously done, the FPGA[28] has been pre-pre-programmed to provide logic components such as AND gates and flip-flops and the student connected these components together. This experiment is designed to provide students with more insight into the structure of a field programmable gate array (FPGA)[28]. A look up table (LUT) [23] is a memory with a one-bit output that essentially implements a truth table where each input combination generates a certain logic output. The input combination is referred to as an address. The HDL synthesizer implements an AND gate or other simple logic function by programming the stored elements in a LUT[23]. In this experiment, you will program three pre-built LUTs[23] and develop a state machine to implement the fox-duck-corn game described below. When finished, you will be able to play the game. A man went on a trip with a fox, a duck and an open can of corn. He came upon a river and a tiny boat to cross the river but he could only take himself and one other - the fox, the duck, or the corn - at a time. He could not leave the fox alone with the duck or the duck alone with the corn. How does he get all safely over the river?

Procedure:

1. Fox Logic - Complete the following next state table shown in the Table 1 according to the game description Definitions: curM (current state of the man), curF (current state of the fox), inF (the control input of bringing the fox over the river), nextF (next state of the fox). Remember that the fox cannot move unless the man is on the same side of the river as the fox.

Table1: Next State Table of Fox Logic

Inputs (address)			Output
curM	curF	inF	nextF
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Duck & Corn Logic - Upon further examination of the game, it turns out that the next state table of the duck and the corn shown in the Table 2 are very similar. Design and write out the next state table of the duck and the corn. As in Part 2, program LUT-2[23] and LUT-3 with the truth table for the duck and corn respectively. Connect LUT[23] outputs and inputs to D-FFs to form a state machine.

Table2: Next State Table of Duck and Corn

Next state table of Duck and Corn

Inputs (address)			Output
curM	curD	inD	nextD
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Inputs (address)			Output
curM	curC	inC	nextC
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

C FPGAs[28] and Their Internal Architecture(Lookup table)

Field Programmable Gate Arrays (FPGAs)[28] offer a reconfigurable design platform which makes them popular among digital designers. Typical internal structure of FPGA[28] (Figure 8) comprises of three major elements:

- Configurable Logic Blocks (CLBs), shown as blue boxes in Figure 8 , are the resources of FPGA[28] meant to implement logic functions. Each CLB is comprised of a set of slices which are further decomposable into a definite number of look-up tables (LUTs) [23], flip-flops (FFs) and multiplexers (Muxes).
- Input/Output Blocks (IOBs) available at FPGA's[28] periphery facilitate external connections. These programmable blocks carry signals 'to' or 'from' FPGA[28] chip. Figure 1 shows IOBs as a set of rectangular boxes enclosed within the FPGA[28] boundary (violet colored outer box).
- Switch Matrix (shown as red-colored lines in Figure 8) is an interconnecting wire-like arrangement within FPGA[28]. These offer connectivity for the CLBs or provide dedicated low impedance, minimum delay paths (for example, global clock line).

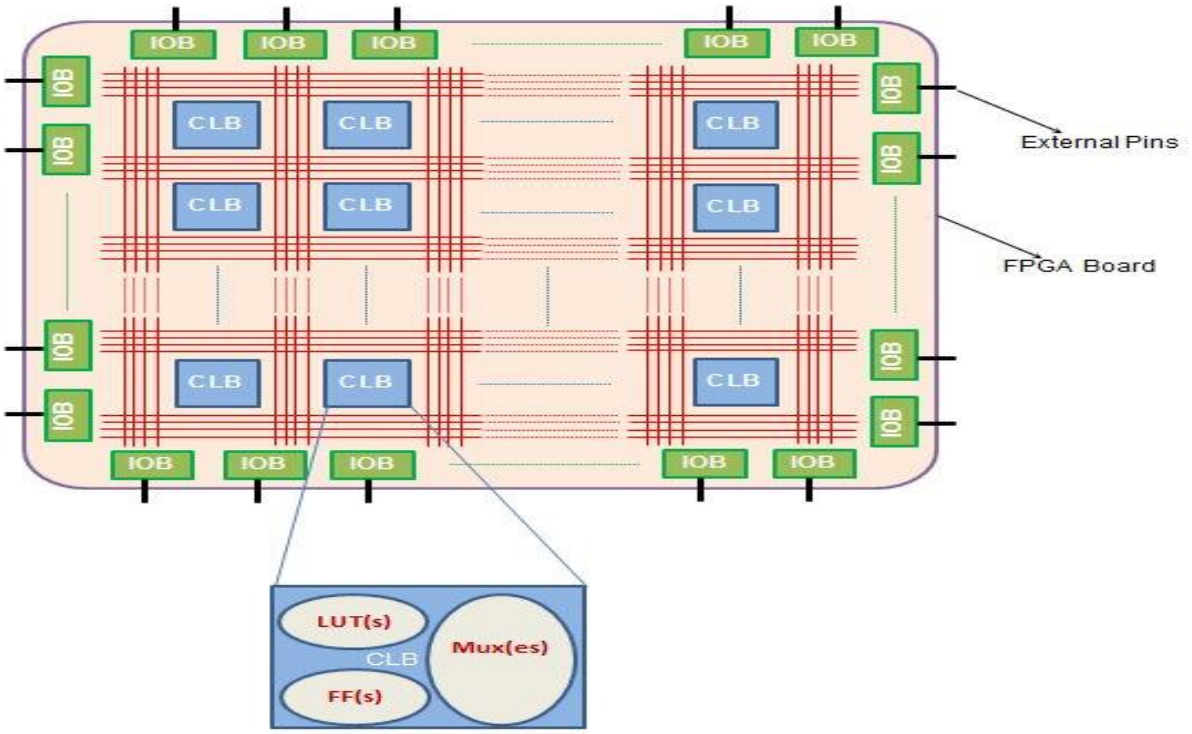


Fig 8: FPGA Internal Architecture

C Mathematical model for Lookup table:

Pre computation of lookup table

Input: $\bar{a}, \bar{b}$
Output: $\bar{a} \cdot \bar{b} \cdot R^{-1} \bmod n$
Precomputation: R^{-1} such that $R \cdot R^{-1} \equiv 1 \pmod{n}$
n' such that $R \cdot R^{-1} - n \cdot n' = 1$

1: $T \leftarrow \bar{a} \cdot \bar{b}$
2: $M \leftarrow T \cdot n' \bmod R$
3: $U \leftarrow (T + M \cdot n)/R$
4: **if** $U \geq n$ **then**
5: **return** $U - n$
6: **else**
7: **return** U
8: **end if**

III RESULTS AND ANALYSIS

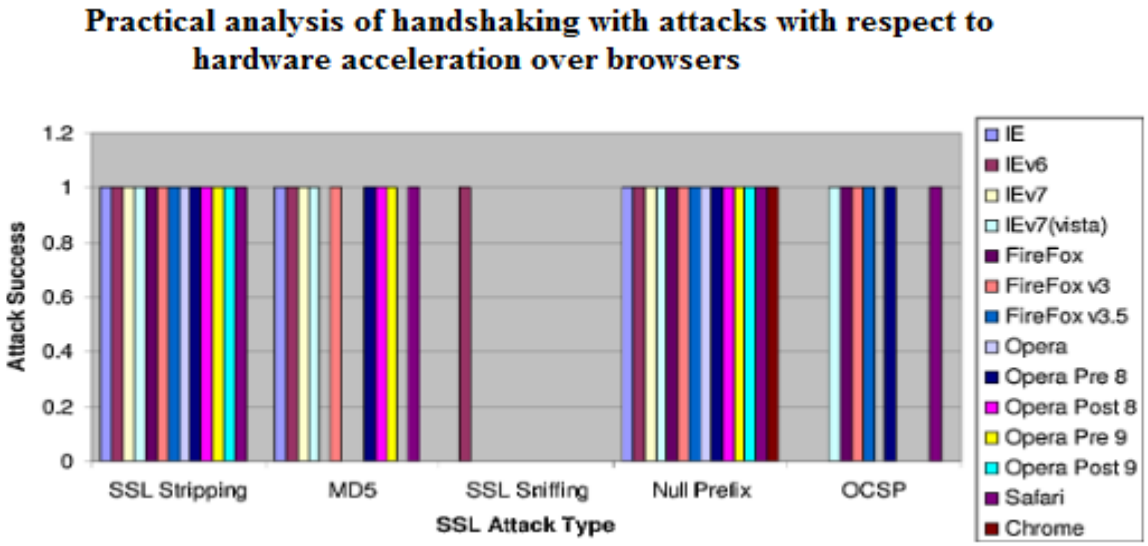
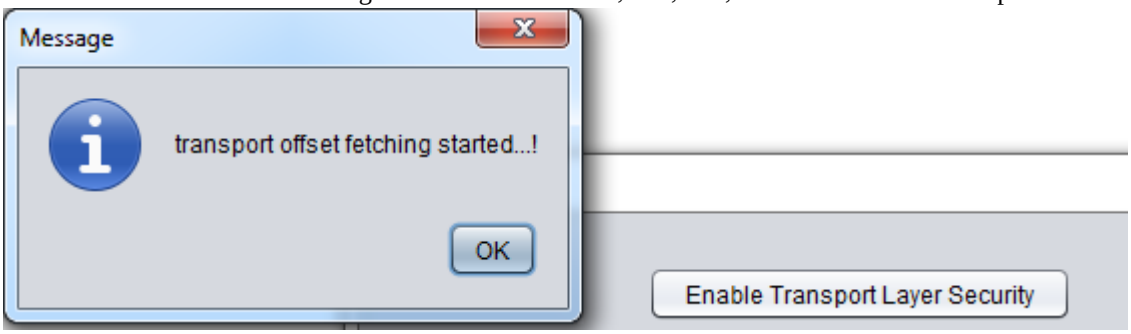
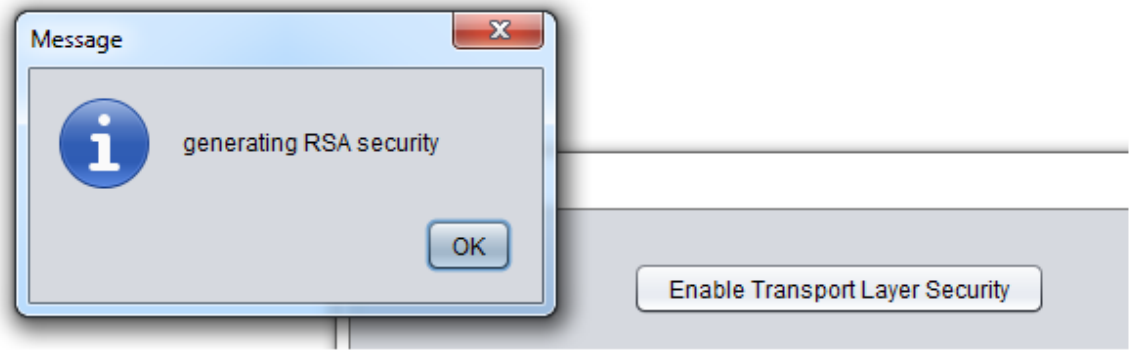


Fig 9:Practical analysis of handshaking with attacks with respect to hardware acceleration over browsers

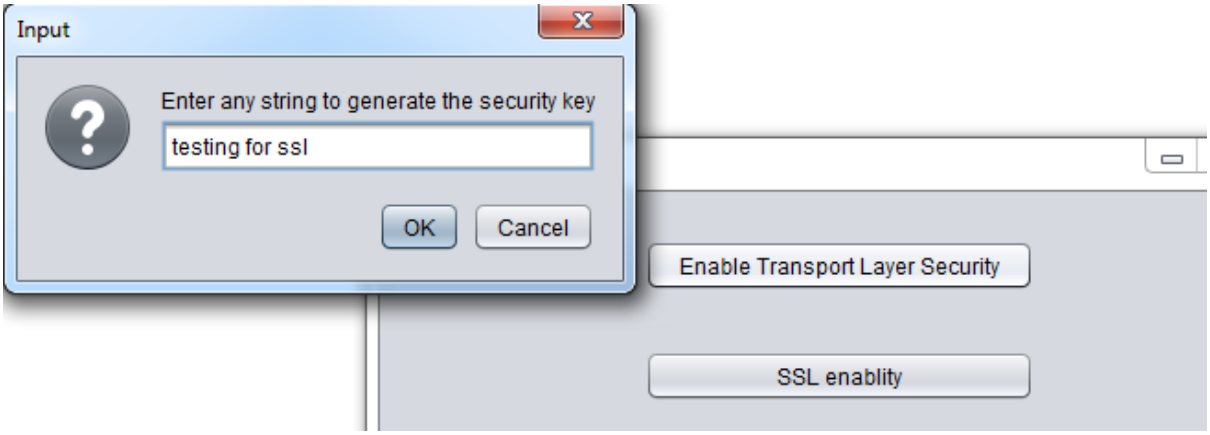
Possible Outcomes: The following screen shot from PO1 ,PO2,PO3,PO4 and PO5 shows the possible outcome.



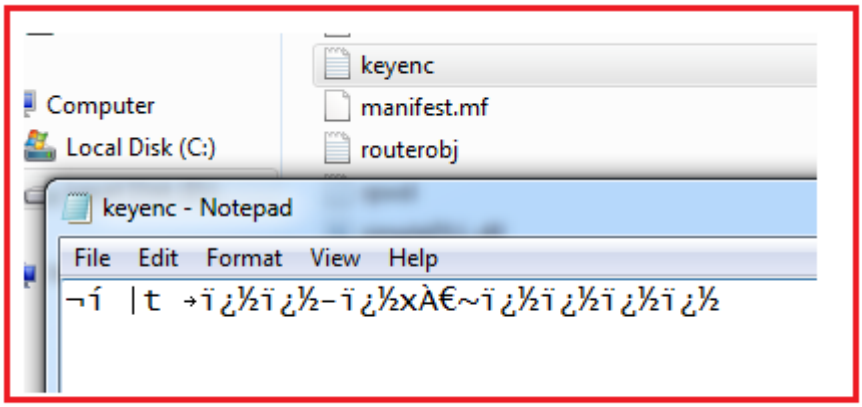
PO1:Transport Layer Hooked(LookUP Table Initiated)



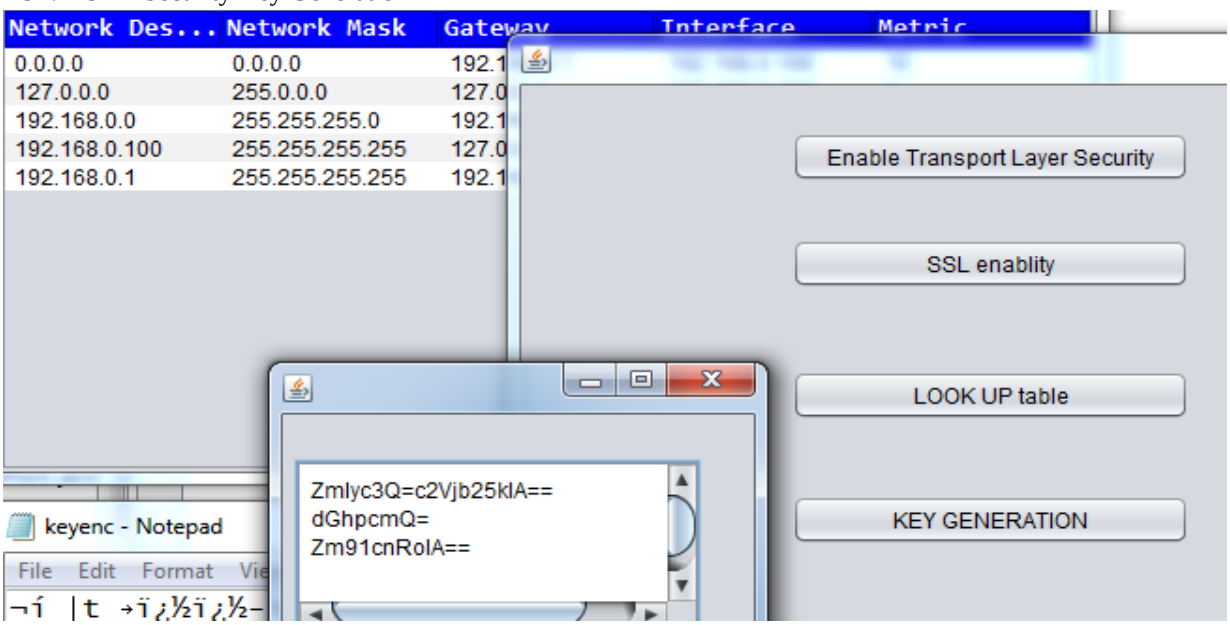
PO2: RSA Security Underwater ROTA Activation at Transport Layer



PO3: SSL/TLS Hardware Acceleration Enabled



PO4: ROTA Security Key Generation



PO5: Fully Qualified LookUp Table with Pre and Post Acceleration with Double ROTA Key Generations.

IV CONCLUSION

Intel SGX offers a unique opportunity to perform secure computation in otherwise untrusted environments. An integral part of Intel SGX is the ability to obtain an attestation on the properties of the enclave and its platform. Integrating remote attestation seamlessly with a standard secure channel protocol greatly simplifies the use of remote attestation in practice. We developed a library that conveniently encapsulates the attestation flow and verification behind a simple API. Using this interface, developers can rely on the added assurance remote attestation provides their application without having to deal with the intricacies of implementing it correctly.

REFERENCE

[1] Connolly, L., Lang, M. & Tygar, D. (2014). Managing Employee Security Behaviour in Organisations: The Role of Cultural Factors and Individual Values. ICT Systems Security and Privacy Protection: 29th IFIP TC 11 International Conference, SEC 2014, Marrakech, Morocco, June 2-4, 2014. Proceedings. pp. 417-430. DOI: 10.1007/978-3-642-55415-5_35.

[2] Datainspektionen. (2008a). Säkerhet för personuppgifter Datainspektionens allmänna råd. Available online: http://www.datainspektionen.se/Documents/fakta broschyr-allmannaradsakerhet.pdf.

[3] Datainspektionen. (2008b). Statliga myndigheters behandling av personuppgifter. Available online: http://www.datainspektionen.se/lagar-och-regler/personuppgiftslagen/eforvaltning/statliga-myndigheters-behandling-av-personuppgifter/

[4] Dierks, T. and Rescorla, E. (2006). The Transport Layer Security (TLS) Protocol Version 1.1. Internet Engineering Task Force (IETF). Available online: https://tools.ietf.org/html/rfc4346.

[5] Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. Internet Engineering Task Force (IETF). Available online: https://tools.ietf.org/html/rfc5246

[6] http://www.hackinglinuxexposed.com/articles/20020423.html

[7] http://www.extremetech.com/article2/0,3973,471936,00.asp

[8] http://www.securityfocus.com/archive/1/286290/2002-07-31/2002-08-06/0

[9] nFast Series, Thales. http://iss.thalesgroup.com/Products/.

[10] NITROX security processor, Cavium Networks. http://www.caviumnetworks.com/processor_security_nitrox-III.html.

[11] OpenSSL Engine. http://www.openssl.org/docs/crypto/engine.html.

[12] Researchers crack 768-bit RSA. http://www.bit-tech.net/news/bits/2010/01/13/researchers-crack-768-bit-rsa/1.

[13] ServerIron ADX Series, Brocade. http://www.brocade.com/products-solutions/products/application-delivery/serveriron-adx-series/index.page.

[14] Silicom Protocol Processor Adapter. http://www.silicom-usa.com/default.asp?contentID=676.

[15] SSL Acceleration Cards, CAI Networks. http://cainetworks.com/products/ssl/rsa7000.htm.

[16] The AMD Fusion Family of APUs. http://sites.amd.com/us/fusion/APU/Pages/fusion.aspx.

[17] Security Architecture for the Internet Protocol. RFC 4301, 2005.

[18] Netcraft SSL Survey. http://news.netcraft.com/SSL-survey, 2009.

[19] Netcraft Web Server Survey. http://news.netcraft.com/archives/2010/04/15/april_2010_web_server_survey.html, 2009.

[20] NVIDIA's Next Generation CUDATMCompute Architecture: FermiTM. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009.

[21] S. Agarwal, V. N. Padmanabhan, and D. A. Joseph. Addressing email loss with suremail: Measurement, design, and evaluation. In USENIX ATC, 2007.

[22] G. Apostolopoulos, V. Peris, and D. Saha. Transport Layer Security: How much does it really cost? In IEEE Infocom, 1999.

[23] Nen-Fu Huang, Shi-Ming Zhao. A Novel IP-Routing lookup Scheme and Hardware Architecture for Multigigabit Switching Routers.

In IEEE Journal on Selected Areas in Communications, 1999.

[24] S. Sridar, S. Smys. Intelligent Security Framework for IOT Devices Cryptography based End-to-End Security Architecture. In International Conference on Inventive Systems and Control, 2017.

[25] Anak Agung Putri Ratna, Prima Dewi Purnamasari, Ahmad Shaugi, Muhammad Salman. Analysis and Comparision of MD5 and SHA-1 Algorithm Implementation in Simple-O Authentication based Security System. In International Conference on QiR, 2013.

[26] A. ALexandrov, V. Monov. Method for WSN clock Syncronization based on Optimized SLTP Protocol. In 25th Telecommunication Forum,2017.

[27] Qing Liu, Yunfei Li, Lin Hao,Hua Peng. Two Efficient Variants of the RSA Cryptosystem. In International Conference on Computer Design and Applications, 2010.

[28] Chenguang Guo, Yanlong Zhang,Lei Chen, Tao Zhou,Xuewu Li, Min Wang, Zhiping Wen. A Novel Application of FPGA-based Partial Dynamic Reconfiguration System with CBSC. In VIII Southern Conference on Programmable Logic, 2012.