A Fast and Secure Way to Prevent SQL Injection Attacks using Bitslice Technique and GPU Support

Piyush Mittal

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

A Fast and Secure Way to Prevent SQL Injection Attacks using Bitslice Technique and GPU Support

Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

by

Piyush Mittal

(Roll 211CS2281)

under the supervision of

Prof. S.K. Jena

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

Dedicated to my mother
and
voice club members...

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, India. `www.nitrkl.ac.in`

Dr. Sanjay Kumar Jena
Professor

May, 2013

Certificate

This is to certify that the work in the thesis entitled *A Fast and Secure Way to Prevent SQL Injection Attacks using Bitslice Technique and GPU Support* by *Piyush Mittal*, bearing roll number 211CS2281, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering.* Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Sanjay Kumar Jena

Acknowledgment

This dissertation would not have been possible without the guidance and the help of several individuals in way or another contributed and extended their valuable assistance in the course of this study.

My utmost gratitude to Prof. S. K. Jena, my dissertation adviser whose sincerity and encouragement I will never forget. Prof. Jena has been my inspiration as I hurdle all the obstacles in the completion of this research work and has supported me throughout my project with patience and knowledge.

I also extend my thanks to the entire faculty of Dept. of Computer Science and Engineering, National Institute of Technology, Rourkela who have encouraged me throughout the course of Masters Degree.

I would like to thank all my seniors, especially Ravi Sankar Barpanda, Ashish Singh and all my classmates for their help and support during the course of work.

And finally thanks to God for his blessings with wisdom and to my parents and Voice Club members for their faith, patience that inspired me to walk upright in my life.

Piyush Mittal

m@piyushmittal.com

Abstract

Most of the web applications are associated with database as back-end so there are possibilities of SQL injection attacks (SQLIA) on it. Even SQLIA is among top ten attacks according to Open Web Application Security Project (OWASP) but still approaches are not able to give proper solution to this problem. Numbers of measures are also discovered to overcome this attack, but which measure is more convenient and can also provide fast access to application without compromising the security is also a major concern. Some existing approaches are good in security but they are not efficient to handle large user's requests. To overcome these two issues at the same moment Bitslice AES encryption and parallel AES encryption using CUDA are used to prevent this attack. Bitslice AES uses a non-standard representation, and view the processor as a SIMD computer, i.e. as 64 parallel one-bit processors computing the same instruction. As AES round functions are good candidate for parallel computations, AES encryption using CUDA gives tremendous encryptions per second and application response remains constant even if users requests increase.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To prevent SQL injection one of the common measure is to use client side validation where all the strings from the submitted form are checked corresponding to data type, length or for any other set of malicious characters [1]. Tokenization in client side validation is also one of the approach where data from all the fields are fetched and converted to tokens and later these tokens are used to match with original query [2]. But only client side validation is not enough as attacks can also be possible using cookies where modified cookie can contain malicious string. Even attacks can be possible if header is modified and it can bypass the submitted form [9]. To avoid such issues, encryption mechanism is proposed. This makes it possible to overcome the attacks by cookies, headers or even by second order injection like Trojan horse [10]. Encryption solved major security issues but if in a web form the numbers of fields are much more and also millions of users are accessing the form at the same moment then encryption of all fields will be a great overhead.

In this paper, Bitslice AES encryption technique is introduced, where if m fields are there in a web form and n users are accessing it simultaneously then (n x m) encryption can be done in parallel. This gives almost constant time to access the application without compromising the security.

1.1 Existed Encryption Mechanisms

Over the past years, there has been plenty of research going on in the both academic institutes as well as industries to prevent injection attacks [?]. Following are some encryption mechanism proposed by researchers which has been seen as more effective one.

1.1.1 "An Authentication Scheme using Hybrid Encryption" (Indrani Balasundaram, E.Ramaraj)-(2011)

They proposed an algorithm which uses both Advance Encryption Standard (AES) and Rivest-Shamir-Adleman(RSA) to prevent SQL injection attacks [12]. In this method a unique secret key is fixed or assigned for every client or user. On the server side server uses private key and public key combination for RSA encryption. In this method, two level of encryption is applied on login query:

- To encrypt user name and password symmetric key encryption is used with the help of user's secret key.
- To encrypt the query the scheme uses asymmetric key encryption by using public key of the server.

The disadvantages of this approach:

- It is not made for URL based SQL injection attacks.
- It is Very difficult to maintain every user secret key at server side and client side.
- There is no security mechanism at registration phase.
- Not efficient when large numbers of user accessing the application simultaneously.

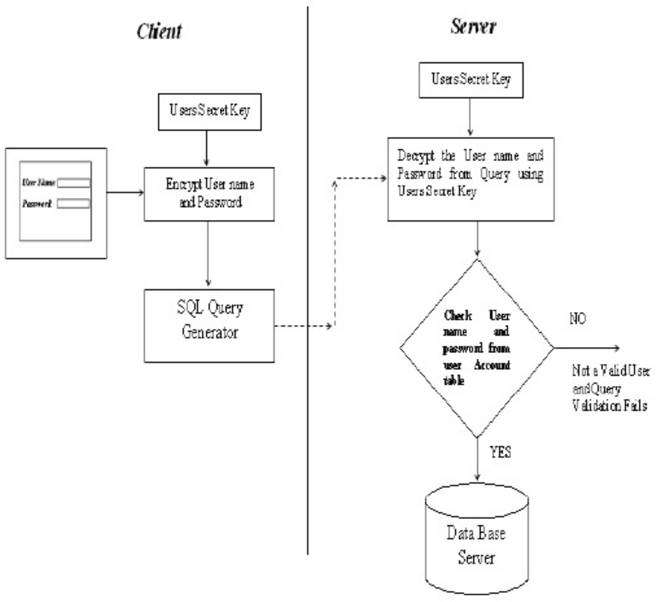

Client
Server
Users Secret Key
User Name
Password
Encrypt User name and Password
SQL Query Generator
Users Secret Key
Decrypt the User name and Password from Query using Users Secret Key
Check User name and password from user Account table
NO
Not a Valid User and Query Validation Fails
YES
Data Base Server

Figure 1.1: Login and Verification Phase

- Vulnerable to cache timing analysis.

1.1.2 "SQLrand: Preventing SQL Injection Attacks" (Stephen W.Boyd , Angelos D.Keromytis)

It uses randomization to encrypt SQL keywords [19]. However the main drawbacks of this approach are

- This needs an additional proxy.
- Computational overhead and the need to remember the keywords.

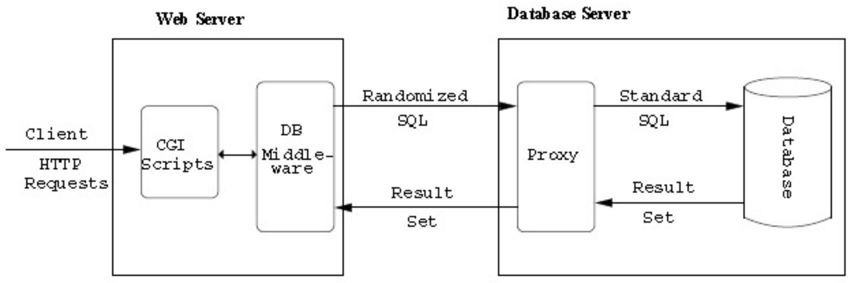


Figure 1.2: SQLrand System Architecture

1.1.3 "Random4: An Application Specific Randomized Encryption Algorithm to Prevent SQL Injection" (2012)

Their approach is based on SQLrand [5, 19] and randomization algorithm is used to convert the input into a cipher text incorporating the concept of cryptographic salt. However the main flaws in this approach are

- Use of lookup table that is not an efficient way.
- Can't handle second order SQL injection attack.

However in our approach all these drawbacks are concerned and proposed solution is secure and efficient.

1.1.4 SQL Injection Background

Why SQL Injection is a major Threat?

The vulnerability trends indicate that significant portion of the vulnerabilities are in applications [25]. Further Cross site scripting and SQL injection are prominent application vulnerabilities among those reported in applications. The SQL injection attacks pose greater risk due to the fact that they impact databases which are critical to any organization. From response perspective also, the remedial action need to be taken by the developer or programmer since the flaw need to be corrected by code level changes. This requires comparatively longer times to take corrective actions after SQL injection vulnerabilities are detected. The attack trends also indicate that SQL injection vulnerabilities are exploited during recent years, in mass scale on vulnerable web applications. Mass scale web intrusions were carried out by ASPROX botnet during 2008 and 2010 which resulted in infection of number of websites in a short span of time. ASPROX used specially formed malicious SQL queries to infiltrate vulnerable databases. In a typical attack ASPROX used Google queries to harvest vulnerable ASP pages and carried out SQL injection and subsequently inserting iFrame links into databases. These iFrames were used to conduct drive-by-download attacks wherein visitors of affected websites are redirected to malicious websites which are used to propagate malware on to users systems. Hence, SQL Injection could be very dangerous in many cases depending on the platform where the attack is launched and it gets success in injecting rogue users to the target system.

Impact of SQL Injection

As we already mentioned SQL injection attack is accomplished by providing data (inclusion of SQL queries) from an external source which is further used to dynamically construct a SQL query. The impact and consequences of SQL injection attacks can be classified as follows:

- Confidentiality: Loss of confidentiality is a major problem with SQL Injection attacks since SQL databases generally hold sensitive and critical information which could be viewed by unauthorized users as a consequence of successful SQL injection attack.
- Integrity: Successful SQL injection attack allows external source to make unauthorized modifications such as altering or even deleting information from target databases.

- Authentication: Poorly written SQL queries do not properly validate user names and passwords, which allows unauthenticated entity or attacker to connect to the affected database or application as an authenticated user, without initial knowledge of the password or even user name.
- Authorization: Successful exploitation of SQL injection vulnerability, allows attacker to change authorization information and gain elevated privileges if the authorization information is stored in the affected database In real world scenario, it is very hard to detect the SQL injection prior to its impact. In most number of scenarios, unauthorized activity is performed by the attacker through valid user credentials or by using inherent features of database application such as malicious modification of existing SQL Queries of web application that are accessing critical sections of the affected databases.

1.2 Motivation

Some existing encrytion mechanism to prevent sql injection attacks are good in security but they are not efficient to handle large user's requests. To overcome these two issues at the same moment, Bitslice AES encryption and parallel AES encryption using Compute Unified Device Architecture(CUDA) can provide a better solution to prevent this attack. Bitslice AES uses a non-standard representation, and view the processor as a SIMD computer, i.e. as 64 parallel one-bit processors computing the same instruction. As AES round functions are good candidate for parallel computations, AES encryption using CUDA gives tremendous encryptions per second and application response remains constant even if users requests increase.

1.3 Organization of thesis

Rest of this thesis is organized as follows. Chapter II discuses Bitslice AES technique. Chapter III introduces CUDA progamiing model and how parallel encryption is done using it. Chapter IV have details about the proposed mechanism i.e prevention of SQLIA using bitslice AES and AES on CUDA. Chapter V have implementation and results of the proposed mechanism. Finally, Chapter VI concludes the thesis.

Chapter 2

Bitslice AES

One of the non conventional but efficient ways to implement AES in software is Bit-slice. It involves converting the algorithm into a series of logical bit operations using AND, OR, XOR and NOT logic gates. When implemented on a microprocessor with a N-bit register width, each bit in the register acts as a 1 bit processor doing a different encryption, therefore N encryptions are done in parallel. This result in tremendous throughput [2, 3].

Also this implementation is free from cache timing attacks. Traditional ciphers make use of several tables to improve performance but that led to memory access pattern of the implementation and make it vulnerable to cryptanalysis [6, 13, 14]. However bitslice implementation is based on logical operation and free from cache timing attack.

2.1 Encryption Mechanism

In bit-slicing we use a non-standard representation and view the processor as a SIMD computer. In order to encrypt the input, first number of words required to store each AES input in memory are calculated. As shown in figure 3.4, if we have N bit microprocessor then we require 128/N words to store each AES input. So if we have 64 bit microprocessor then we need 128/64 i.e. two words to store each AES input.

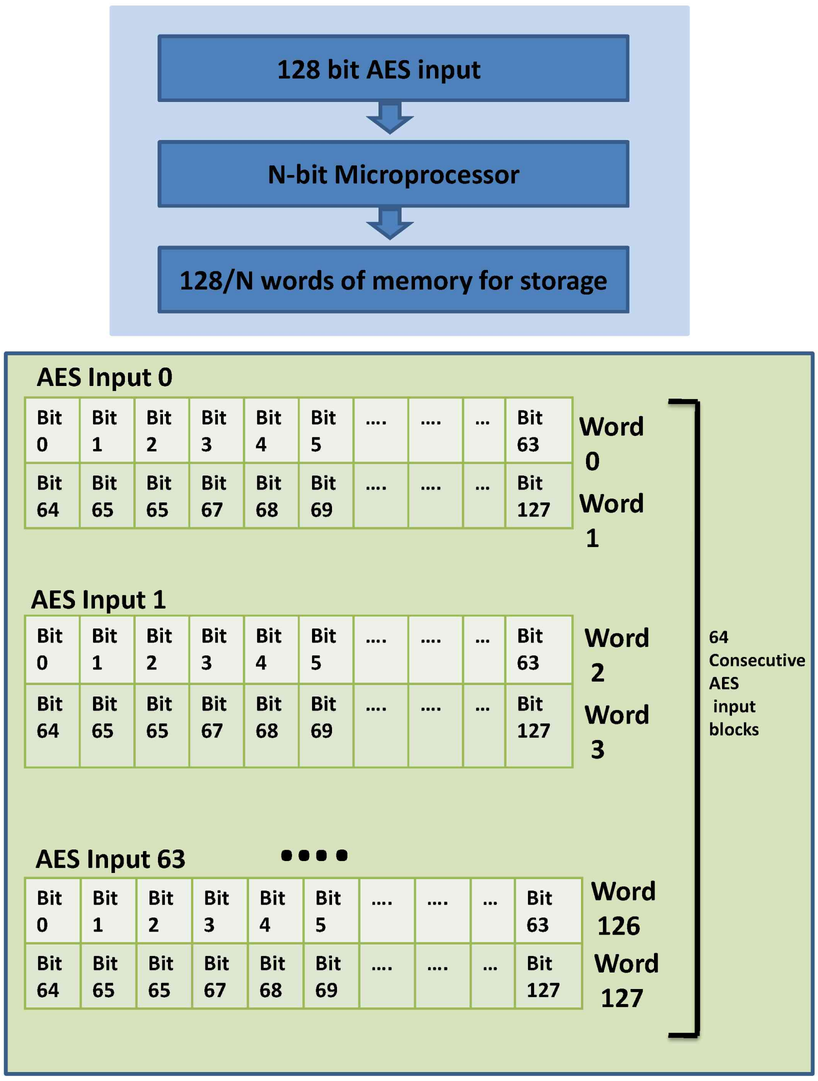

128 bit AES input
N-bit Microprocessor
128/N words of memory for storage
AES Input 0
Bit 0, Bit 1, Bit 2, Bit 3, Bit 4, Bit 5, ..., ..., ..., Bit 63
Bit 64, Bit 65, Bit 65, Bit 67, Bit 68, Bit 69, ..., ..., ..., Bit 127
Word 0
Word 1
AES Input 1
Word 2
Word 3
AES Input 63
Word 126
Word 127
64 Consecutive AES input blocks

Figure 2.1: Storage of AES input in memory for 64 bit processor

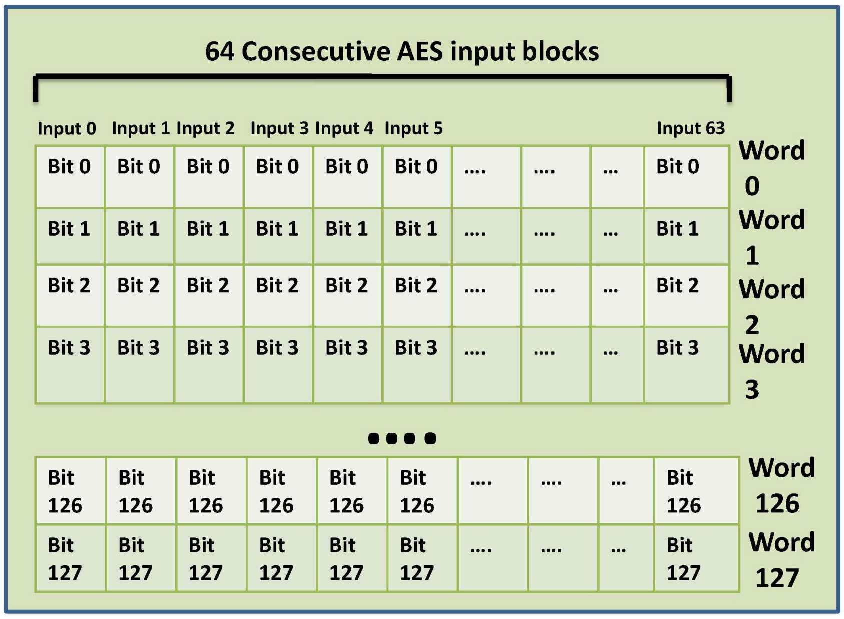


Figure 2.2: Rearrangement of bits in words

Here, 64 AES inputs are taken and therefore we need 128 words in memory to store these inputs. In each word 64 bits can be stored and therefore 128 AES input require 2 words. Now we need to rearrange bits in these words so that first bit of each word contains first AES input. Second bit of each word contain second AES input. Third bit of each word contain third AES input and so on. The final arrangement is shown in Figure 2.2. After rearrangement we can see that each bit layer of words have particular AES input and it forms the basis of parallel processing. All the AES operations are converted to logical operations. Now each layer of words in memory i.e. particular AES input is passed through Round function.

The first transformation in round function is Substitute byte. In substitute function, instead of using table look that led to the cache crypt analysis, Rijmen gave the idea that S Boxes can be implemented using combinational logics [6]. Eli Biham in his paper A Fast New DES Implementation in Software modified the S Box logic circuit and gate count of logic circuit is reduced to 100 gates [4]. Matthew Kwan's further modify the S box generation algorithm and gate count is reduced to average of 53.375 gates per S Boxes for DES [5]. Currently Roman Rusakov introduced the S box in just 44.125 gates per S-box and its implementation is used in John the Ripper cracker [15].

For AES as there is just one S Box, the gate count for it according to reference [3] is 132, so there is much scope to further reduce it. Source program of Bitsliced AES Sbox in x64 assembly language can be found from reference [16]. After substitute byte transformation, the shift rows operation is carried out by move operation where second row of the state is shifted left by eight bits, the third row by sixteen bits and the fourth row by twenty four bits. The final state after substitute byte and shift row is shown in figure 2.3. In mix column transformation we multiply the column of the state matrix with permutation of [2 3 1 1]. If multiplication leads to overflow then constant $1B_{16}$ is added. The Add round adds each word from memory to corresponding word of key in memory. Further equations of MixColumn, ShiftRows, AddRoundKey, Overflow constant and RoundKey(key) can be added at one place. Output for first byte can be summarized as follows:

$Bit'_0 = Bit_{40} + Bit_{80} + Bit_{120} + Key_0 + overflow$

$Bit'_1 = Bit_0 + Bit_{40} + Bit_{41} + Bit_{81} + Bit_{121} + Key_1 + overflow$

$Bit'_2 = Bit_1 + Bit_{41} + Bit_{42} + Bit_{82} + Bit_{122} + Key_2$

$Bit'_3 = Bit_2 + Bit_{42} + Bit_{43} + Bit_{83} + Bit_{123} + Key_3 + overflow$

$Bit'_4 = Bit_3 + Bit_{43} + Bit_{44} + Bit_{84} + Bit_{124} + Key_4 + overflow$

$Bit'_5 = Bit_4 + Bit_{44} + Bit_{45} + Bit_{85} + Bit_{125} + Key_5$

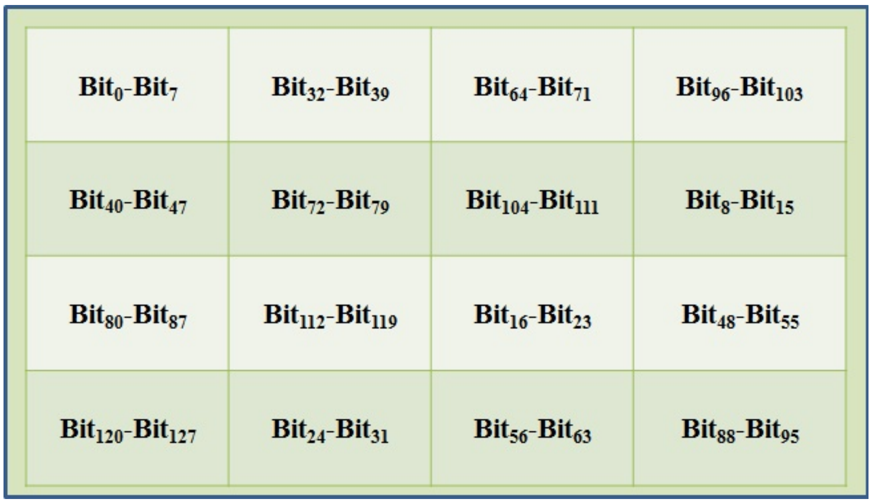
$Bit'_6 = Bit_5 + Bit_{45} + Bit_{46} + Key_{86} + Bit_{126} + Key_6$

$Bit'_7 = Bit_6 + Bit_{46} + Bit_{47} + Bit_{87} + Bit_{127} + Key_7$

where, $overflow = Bit_7 + Bit_{47}$

2.2 Summary

In this chapter, we have seen how AES encryption can be implemented using bitslicing technique. This implementation is also free from cache timing attack.In



Bit_0-Bit_7	Bit_{32}-Bit_{39}	Bit_{64}-Bit_{71}	Bit_{96}-Bit_{103}
Bit_{40}-Bit_{47}	Bit_{72}-Bit_{79}	Bit_{104}-Bit_{111}	Bit_8-Bit_{15}
Bit_{80}-Bit_{87}	Bit_{112}-Bit_{119}	Bit_{16}-Bit_{23}	Bit_{48}-Bit_{55}
Bit_{120}-Bit_{127}	Bit_{24}-Bit_{31}	Bit_{56}-Bit_{63}	Bit_{88}-Bit_{95}

Figure 2.3: Final state after substitute byte and shift row transformation

bit-slicing we use a non-standard representation and view the processor as a SIMD computer.It involves converting the algorithm into a series of logical bit operations using AND, OR, XOR and NOT logic gates. When implemented on a microprocessor with a N-bit register width, each bit in the register acts as a 1 bit processor doing a different encryption, therefore N encryptions are done in parallel. In the next chapter, we will see how AES encryption can be done using CUDA on GPU.

Chapter 3

CUDA PROGRAMMING MODEL

CUDA stands for (compute unified device architecture) and provide parallel computing platform and programming model that increases computing performance by harnessing the power of the graphics processing unit (GPU). It is best to solve the problems that are intensive to data parallel computations i.e. the same program is executed on many data elements in parallel with high arithmetic intensity, the ratio of arithmetic operations to memory operations.

At CUDA core, the following key abstractions are there a hierarchy of thread groups, shared memories, and barrier synchronizatio. All these for a programmer are just as minimal set of extensions to C.This helps to provide data parallelism and thread parallelism, along with coarse-grained data parallelism and task parallelism. Programmer can now partition the problem into coarse sub-problems which can be solved independently in parallel, and later into finer pieces which can be solved cooperatively in parallel. This decomposition allows threads to perform correctly while solving sub-problems, and also enables transparent scalability because every sub-problem can be solved on any of the available processor cores: Therefore we can execute compiled CUDA program on any number of processor cores, and physical

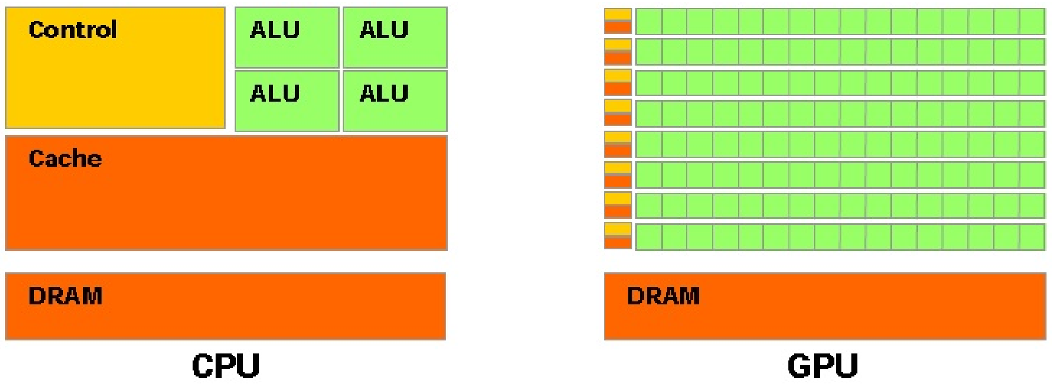


Figure 3.1: The GPU Devotes More Transistors to Data Processing

processor count is required by only the runtime system.

CUDA features a parallel data cache or on-chip shared memory with very fast general read and writes access, that threads use to share data each other. As opposed to regular C functions, CUDA has the capability to define C functions, also called kernels, that can execute N times in parallel by N different CUDA threads. The threads that executes the kernel is given a unique thread ID that can be accessed through in built-in threadIdx variable. threadIDx is a 3-component vector and it involves one-dimensional, two-dimensional or three dimensional index forming a one-dimensional, two-dimensional or three dimensional thread block. However the relation between thread and its ID is same for one dimensional block but different for two-dimensional and three dimensional block. For a two-dimensional block of size (Dx, Dy), the thread ID of a thread of thread of index (x, y) is (x+yDx) and for a three dimensional block of size (Dx, Dy, Dz) the thread ID of a thread of index (x, y, z) is (x+yDx+zDxDy). Threads in a block synchronize themselves using shared memory. We can also execute kernel by using multiple equally shaped threads blocks. Such blocks can be arranged in a one-dimensional or two-dimensional grid of thread blocks.

The CUDA threads shown in 3.2 are structured hierarchically, i.e. a maximum of 512 threads forming a thread block which can be 1-, 2- or 3-dimensional, and thread blocks forming a 1- or 2-dimensional grid. Another important concept is thread scheduling. When a kernel is launched, every stream multiprocessor (SM) is assigned a thread block. A warp, which consists of 32 threads, is executed simultaneously on a SM at any time [22]. Though the warp is not a concept of CUDA specification, good understanding to it can assist achieving high performance. One principle in optimizing a CUDA program is to keep all the threads in a warp working efficiently and none of them is doing useless work. Five types of memory are exposed to developers through CUDA, namely global memory, constant memory, texture memory, shared memory and registers. Shared memory and register are high speed on-chip memory; global memory, constant memory and texture memory all reside in the DRAM on board. Global memory is not cached while constant memory and texture memory are cached and therefore the later two can also be accessed very fast when there is no cache miss match. However, constant memory and texture memory are read-only. CUDA uses C programming language with a minimal extension which can greatly facilitate the learning. More information about the extension can be found in [21, 23, 24]. In order to run CUDA threads we need a separate device that behaves as a coprocessor to the actual host running the actual c program. Host can accommodate serial code but to execute parallel kernel we need to make use of device. We can also call parallel kernels in between the serial code.

3.1 AES on GPU

We need an efficient encryption kernel to fully utilize the parallel computing power of GPU. All the AES transformations are good candidate to run parallel on GPU. The first step before beginning the encryption process is to make cudaMalloc API calls to GPU device for allocating memory to plaintext, ciphertext, expanded key and AES tables [20].

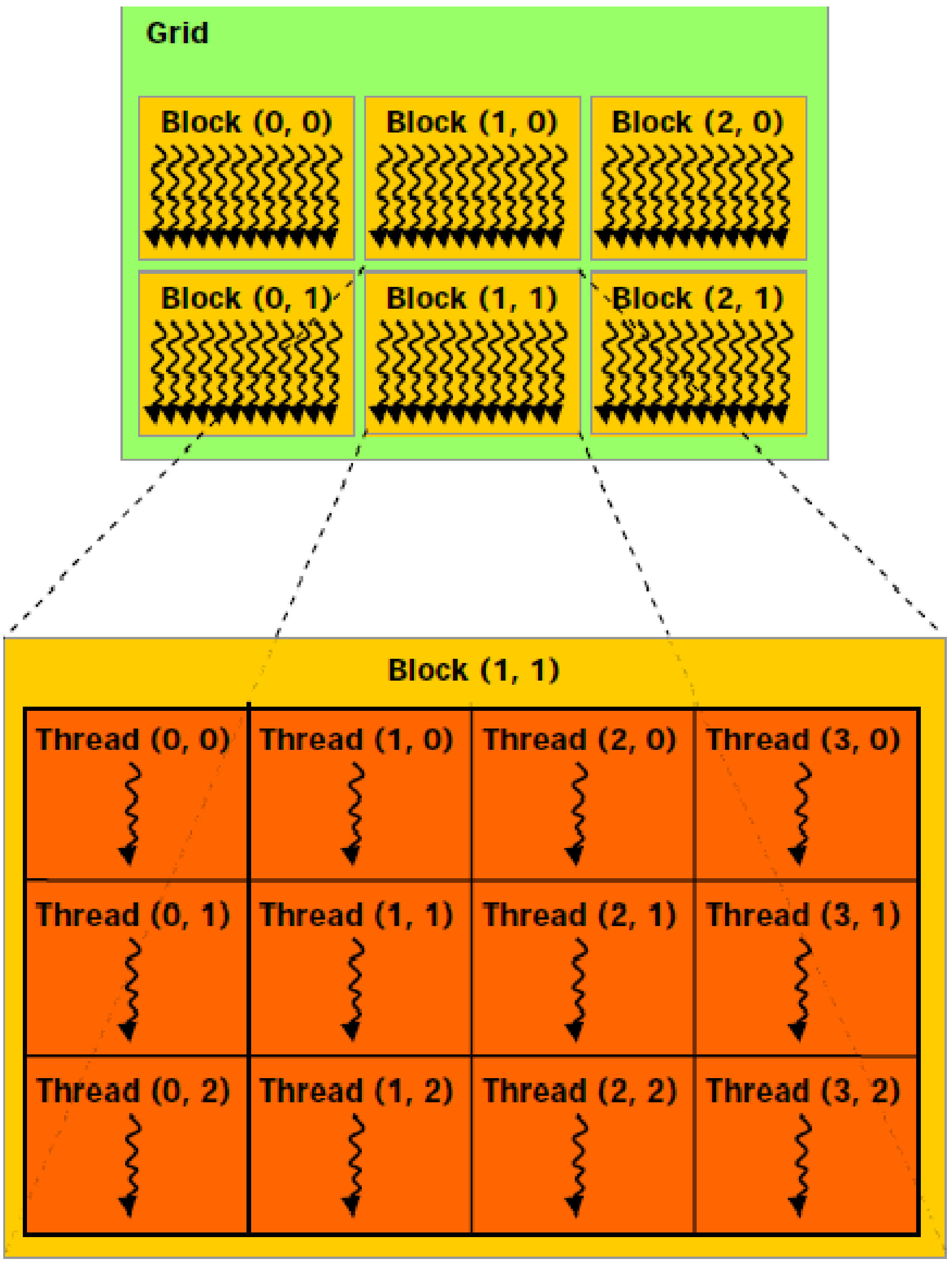


Figure 3.2: Grid of Thread Blocks

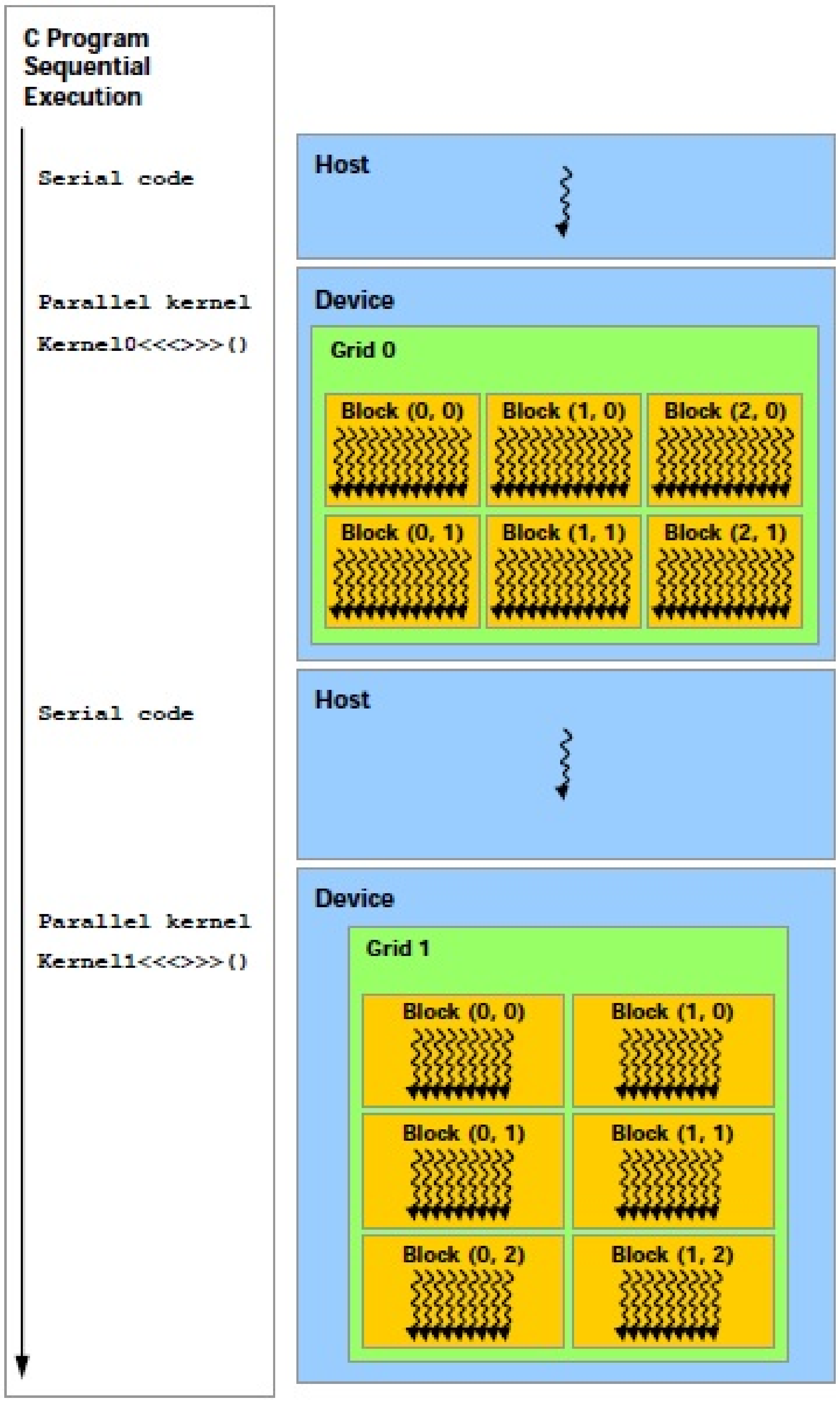


Figure 3.3: Serial and parallel code execution on host and device respectively

```
unsigned int* device-plaintext;
unsigned int* ciphertext;
unsigned int* device-expandedkey;
cudaMalloc((void**)&device-plaintext, sizeof(unsigned int)*(mem-length ));
cudaMalloc((void**)&ciphertext, sizeof(unsigned int)*(mem-length ));
cudaMalloc((void**)&device-expandedkey, sizeof(unsigned int)*44);
```

These buffers are created in the global memory or D-RAM of the GPU. Inputdata, expanded key, and the AES tables are then copied from the host to the device using the cudaMemcpy API call.

```
cudaMemcpy(device-plaintext,plaintext,
sizeof(unsigned int)*mem-length, cudaMemcpyHostToDevice) ;
 cudaMemcpy(device-expandedkey,expandkey,
sizeof(unsigned int)* 44, cudaMemcpyHostToDevice);
```

Now in order to frequently access the expanded key and plaintext both of them are loaded to shared memory by using the variable as follows :

```
__shared__ unsigned int shared-roundkey[44];
__shared__ unsigned int shared-plaintext[BLOCK-SIZE];
shared-rourndkey[thread-id] = device-roundkey[thread-id];
shared-plaintext[thread-id]   = device-plaintext[thread-id];
```

Where, share-plaintext and shared-roundkey are the variables of plaintext and expanded key respectively, which reside in the shared memory space of a thread block. thread-id refers to the thread ID.

To allow fast implementation of AES on processor we can make use of table lookups that can combine all the round transformations in a single set.

Fo the key addition and the MixColumn transformation, we have

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

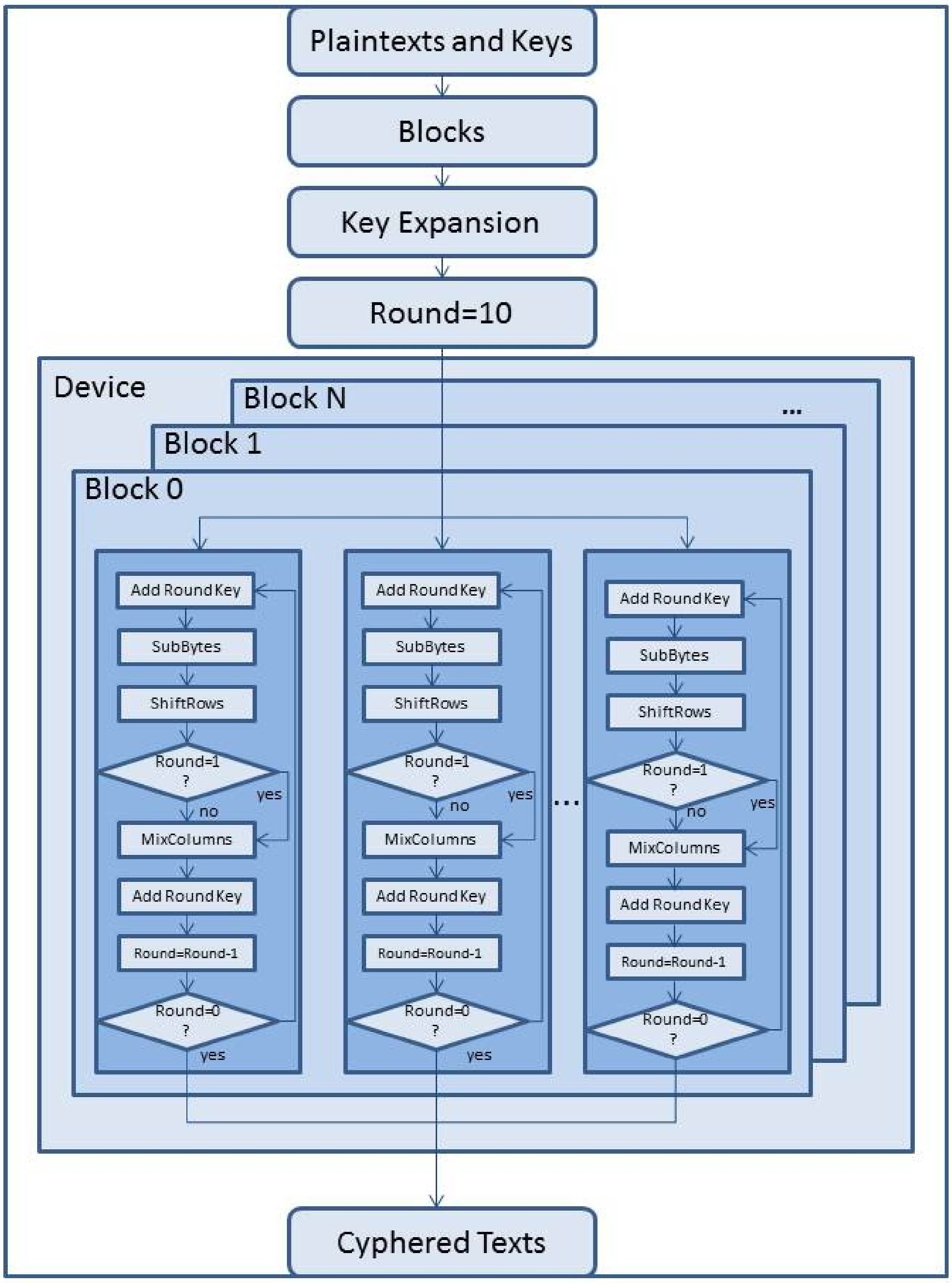


Figure 3.4: Parallel AES encryption

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

For the ShiftRow and the ByteSub transformations, we have:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-c1} \\ b_{2,j-c2} \\ b_{3,j-c3} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} b_{i,j} \end{bmatrix} = S\begin{bmatrix} a_{i,j} \end{bmatrix}$$

We define tables T0 to T3 :

$$T_0[a] = \begin{bmatrix} S[a] . 02 \\ S[a] \\ S[a] \\ S[a] . 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a] . 03 \\ S[a] . 02 \\ S[a] \\ S[a] . 03 \end{bmatrix}$$

$$T_2[a] = \begin{bmatrix} S[a] \\ S[a] . 03 \\ S[a] . 02 \\ S[a] \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] . 03 \\ S[a] . 02 \end{bmatrix}$$

These are 4 tables with 256 4-byte word entries and make up for 4KByte of total space. Using these tables, the round transformation can be expressed as:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-c1}] \oplus T_2[a_{2,j-c2}] \oplus T_3[a_{3,j-c3}]$$

3.2 Summary

In this chapter, we have seen how AES encryption can be done on GPU using CUDA. CUDA is best to solve the problems that are intensive to data parallel computations i.e. the same program is executed on many data elements in parallel with high arithmetic intensity, the ratio of arithmetic operations to memory operations. All the AES transformations are good candidate to run parallel on GPU. In the next chapter, we will see how Bitslice AES and AES encryption using CUDA can be used to defend SQL injection attack.

Chapter 4

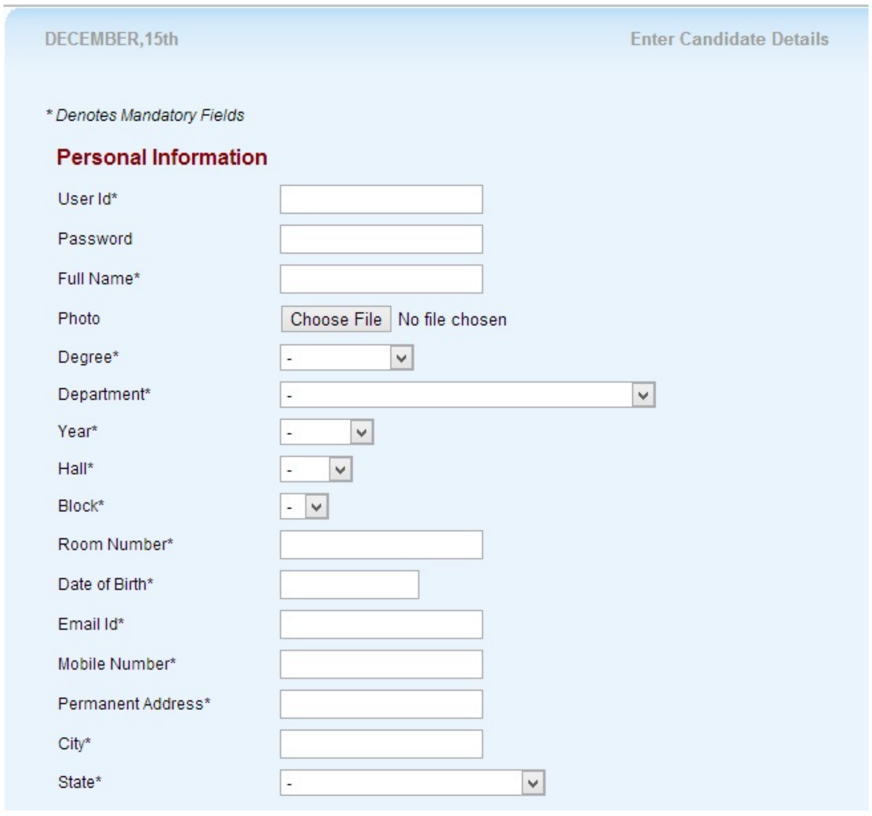
Proposed Mechanism

To defend the SQLIA we need to concern all phases of an application (i.e. Registration, Login and Search phase).

Registration Phase

In registration phase a user fills the form by relevant details like username, password, date of birth, address, and other data related to application. Now in order to avoid SQLIA we need to encrypt only those fields that we will use in conditional clause of Sql query because with these fields only there will be a chance of SQLIA. For encryption, we will use Bitslice AES. The 128 bit binary key which is equivalent to 16 ASCII character can be formed by appending first eight character of username and first eight character of password. If username or password is less than eight characters then binary bit 0 is appended to make it 128 bit long. Now using this key all the relevant fields are encrypted and stored in database.

As an example, we can see registration page from Figure 4.1 that only [userid, password] are used in Login page of Figure 4.2 and [degree, department, email id and mobile number] are used in search page of Figure 4.3, so we need to encrypt only these fields. After encryption these encrypted fields have to be stored in database.


DECEMBER,15th

Enter Candidate Details

** Denotes Mandatory Fields*

Personal Information

User Id*	
Password	
Full Name*	
Photo	Choose File No file chosen
Degree*	-
Department*	-
Year*	-
Hall*	-
Block*	-
Room Number*	
Date of Birth*	
Email Id*	
Mobile Number*	
Permanent Address*	
City*	
State*	-

Figure 4.1: Registration Page

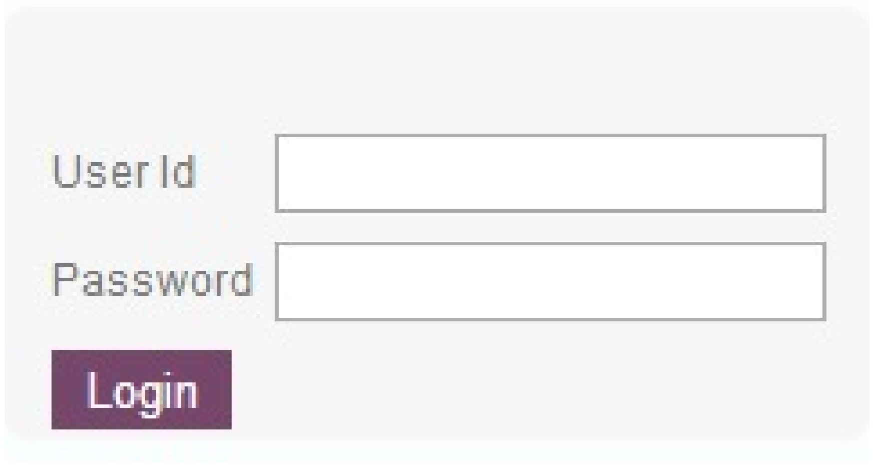


Figure 4.2: Login Page

Login Phase

In login phase a user need to enter username and password. Now we need to match these user credentials from the database but the username and password in database are encrypted because of registration phase. So before matching username and password with database encryption of both the fields with Bitslice AES is required. The key can be formed in the same way as in registration phase. Here there is no need to store secret key in database as done by Indrani et al. [12].

Search Phase

In search phase the user interacts more with database and can fetch any stored data based on some specific fields given in search form. As we know these specific fields are stored in encrypted form in the database so we need to encrypt these fields before firing SQL query.

4.1 Prevention of SQLIA using Bitslice AES

Most web application include login page with two basic fields ”user id” and ”password” that form a single SQL query as follow :

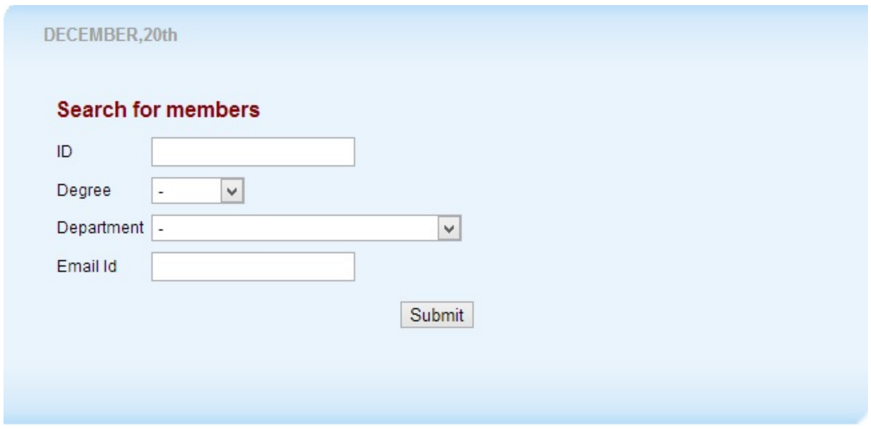


Figure 4.3: Search Page

SELECT "user" from "tablename" where userid= ' ' and password= ' ';

Here we need just two encryption one for user id and other for password but suppose after login in our web application we need more interaction with database and more number of field are present in other web pages.

Let us consider the search form in Figure 4.3, it has four fields. Some of the corresponding SOL queries to fetch relevant data from the table are as follow :

SELECT * from "tablename" where id = ' ';
SELECT * from "tablename" where degree = ' ' and dept = ' ';
SELECT * from "tablename" where eid =' ';

So we need to do four encryption before appending the fields in these sql queries. Now suppose if millions of users are accessing the search page at the same moment then in total it require (Four x Million) encryptions and hence we need a better solution to handle this situation.

We can solve this problem using Bit slice AES in a much efficient way, Suppose if M users are present and there are N fields in the webpage then N X M encryption can be done in parallel. To handle such requests we need to maintain a queue, where all the requests keeps on adding and later for encryption same as the number of bits of microprocessor string are fetched from queue. Each character string from the fields of web page is converted to corresponding 128 bit binary string that form the AES input. As each character takes 8 binary bits so there can be a maximum of 16 characters that can accommodate in 128 bit AES input. If number of characters are less than 16 then we can append 0 to make complete 128 bit binary string and if characters are more than 16 then we can do encryption using modern block ciphers like Electronic Code Book (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback Mode etc. Now for encryption, according to Figure 3.4, the first bit of first string will go to first bit of first word, the second bit of first string will go to second bit of first word and in similar way one hundred twenty eighth bit of string will go to one hundred twenty eighth bit of second word. Now the first bit of second string will go to first bit of third word, the second bit of second string will go to second bit of third word and in similar way one hundred twenty eighth bit of second string will go to one hundred twenty eighth bit of fourth word and so on. So in this way we can feed input to the bit slice AES and other operations can be carried out in similar way as mention in section II. After encryption SQL query is fired and the corresponding attribute is matched from database table. If match is appropriate then user can access the data from the table otherwise it is an injection attack. Again a new set of 64 strings are fetched from the queue and sent to encryption process.

4.1.1 Example

Let us consider the Figure 4.5, here N users are accessing the web form at the same time having four fields. As in 64 bit machine we can do 64 encryption in parallel so there can be maximum of sixteen user's data that can be processed at the same moment (16 users X 4 fields=64 encryption). First of all the requests from users are stored in queue then consecutive 64 strings are fetched from the queue. Then each string is converted to 128 bit binary string. As we can see from the figure that all the bits of string 0 are set to the first bits of all words. Second strings have all the bits on second bits of all the words and similarly for all other strings. After encryption a new set of 16 users data is taken and this process continues till all the

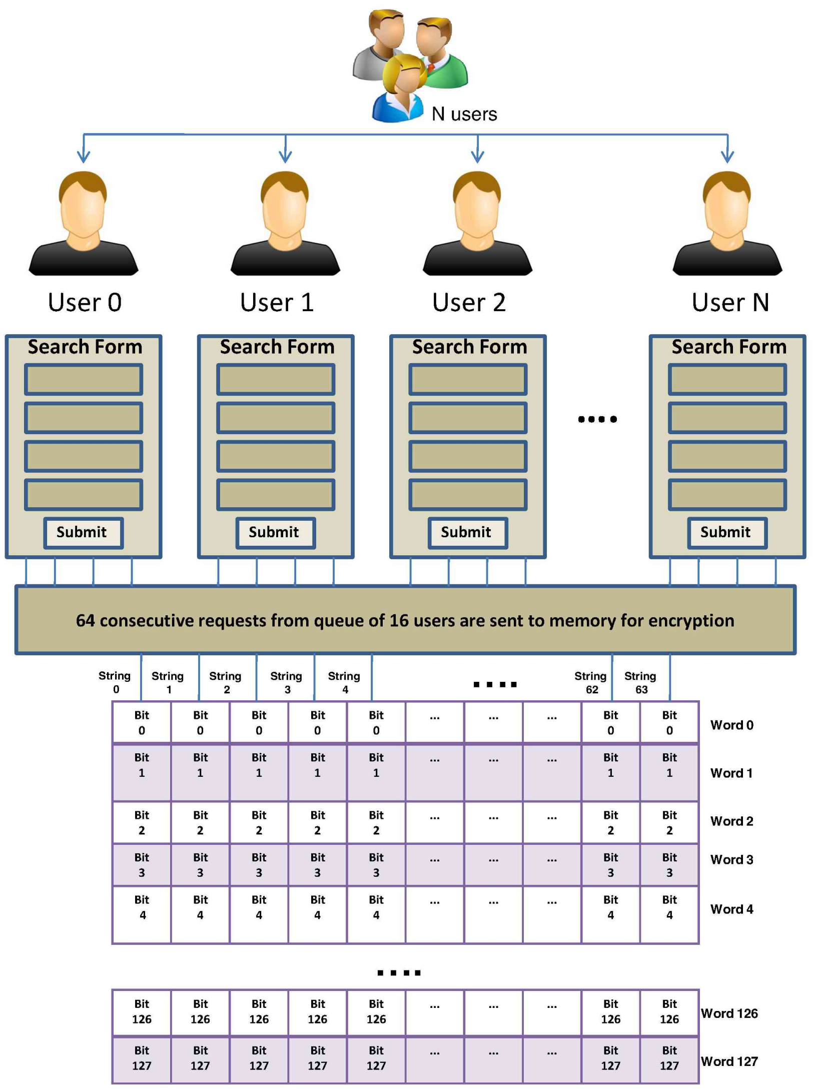


Figure 4.4: String storage in memory from sixteen users at a time from a web form of four fields

request has not finished.

4.2 Prevention of SQLIA using AES on CUDA

Most web application include login page with two basic fields ”user id” and ”password” that form a single SQL query as

```
SELECT "user" from "tablename" where userid=‘ ’ and password=‘ ’ ;
```

Here we need just two encryption one for user id and other for password but suppose after login in our web application we need more interaction with database and more number of field are present in other web pages.

Let us consider the search form in Figure 4.3, it has four fields. Some of the corresponding SOL queries to fetch relevant data from the table are as follow :

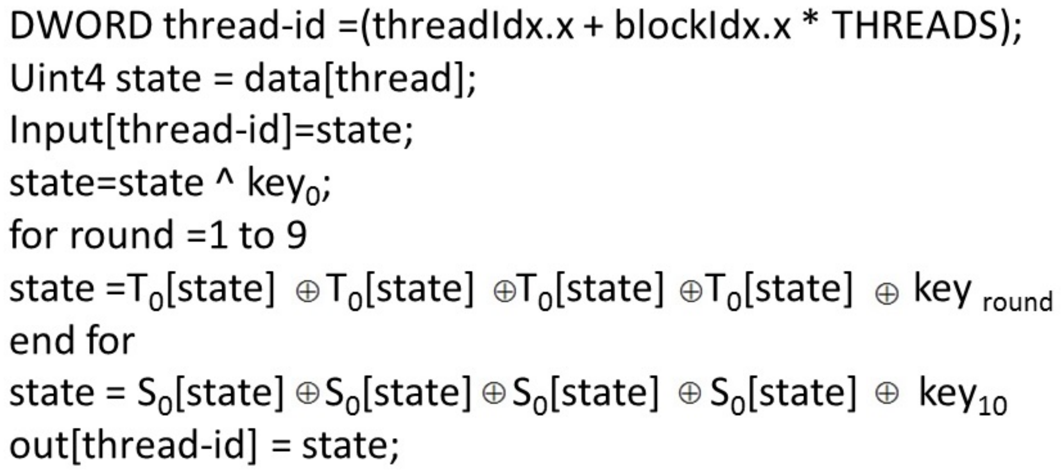
```
SELECT * from "tablename" where id = ‘ ’;
SELECT * from "tablename" where degree = ‘ ’ and dept = ‘ ’;
SELECT * from "tablename" where eid =‘ ’;
```

So we need to do four encryption before appending the fields in these sql queries. Now suppose if millions of users are accessing the search page at the same moment then in total it require (Four x Million) encryptions and hence we need a better solution to handle this situation.

We can solve this issue using AES encryption on GPU in a much efficient way, Suppose if M users are present and there are N fields in the webpage then N X M encryption can be done in parallel. To handle such requests we need to maintain a queue, where all the requests keeps on adding and later for encryption same as the maximum number of GPU thread of CUDA block are fetched from queue. Each character string from the fields of web page is converted to corresponding 128 bit binary string that form the AES input. As each character takes 8 binary bits so there can be a maximum of 16 characters that can accommodate in 128 bit AES input.

If number of characters are less than 16 then we can append 0 to make complete 128 bit binary string and if characters are more than 16 then we can do encryption using modern block ciphers like Electronic Code Book (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback Mode etc. After encryption SQL query is fired and the corresponding attribute is matched from database table. If match is appropriate then user can access the data from the table otherwise it is an injection attack. Again a new set of strings are fetched from the queue and sent to encryption process.

In the table-lookup, Each T table contains 256 4-byte word entries and these 4 T tables make up for 4KByte of total space. In CUDA-enabled GPU, it includes the global memory, shared memory, texture memory and constant memory. Among them, the global memory can be configured up to 4 GBytes. Given the flexibility of the memory model, it is possible to efficiently use the four T-lookup tables. Besides, the core of CUDA-enabled GPU is a scalar processor. Therefore, there is no need to combine instructions in vector operations in order to get the full processing power. Furthermore, the 32 bit logical exclusiveor operation can be executed natively on CUDA-enabled GPU. The pseudo-code of kernel function of our parallelized AES algorithm implemented on GPU.



```
DWORD thread-id =(threadIdx.x + blockIdx.x * THREADS);
Uint4 state = data[thread];
Input[thread-id]=state;
state=state ^ $key_0$;
for round =1 to 9
state =$T_0$[state] ⊕$T_0$[state] ⊕$T_0$[state] ⊕$T_0$[state] ⊕ $key_{round}$
end for
state = $S_0$[state] ⊕$S_0$[state] ⊕$S_0$[state] ⊕ $S_0$[state] ⊕ $key_{10}$
out[thread-id] = state;
```

From figure , we can see that every GPU thread of CUDA block performs 4 table lookups and 4 32-bit exclusive-or operations in each AES round on a state.

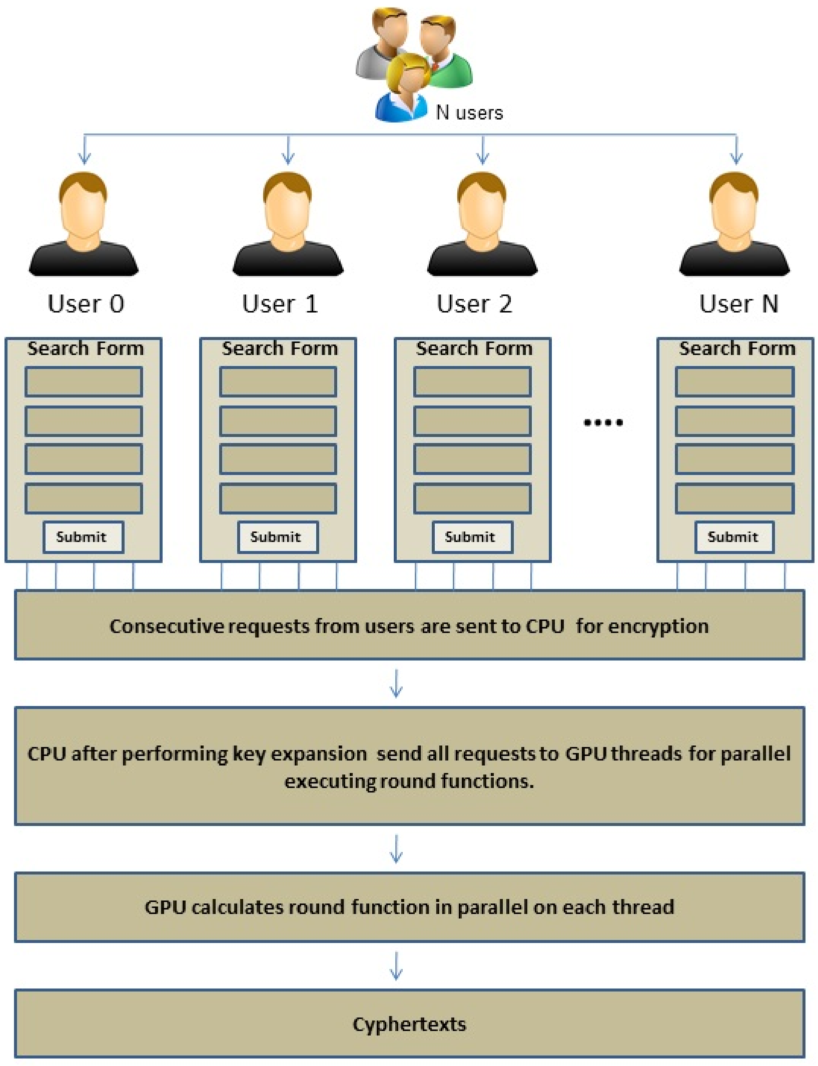

N users
User 0
User 1
User 2
User N
Search Form
Submit
....
Consecutive requests from users are sent to CPU for encryption
CPU after performing key expansion send all requests to GPU threads for parallel executing round functions.
GPU calculates round function in parallel on each thread
Cyphertexts

Figure 4.5: Encryption process of N users using CUDA Architecture

Besides, two arrays of 1KB shared memory are used for the input, reading data from the first and saving results of each AES round to the second one. Then the arrays are swapped for the successive round. This strategy allows to complete the encryption of the input block without exiting the kernel. So it is done without using the CPU to manage an external for loop to launch sequentially all the AES rounds. At the end of the computation, the resulted output data is written again in the global memory and then returned to the CPU.

4.3 Summary

In this chapter, we have seen how Bitslice AES and AES encryption using CUDA can be used to prevent SQL injection attack. The prevention process considered all the phases of the application i.e Registeration, Login and Search phase.

Chapter 5

Implementation and Results

5.1 Implementation and Results using Bitslice AES

In this section, we have considered three different systems as listed in table 5.1. In order to get the benchmark of the performance of Bitslice AES the maximum encryption per second on three machines is as shown in table 5.2.

Table 5.1: Machines Used for testing

Types	Microprocessor	CoreSpeed	Memory	O.S	Compiler
1	Core i7-2600K	3.40 GHz	4GBytes	Linux	gcc 4.6.1-9ubuntu3
2	Celeron E3200 oc	4.00 GHz	2GBytes	Linux	gcc 4.6.2 SUSE
3	Core 2 Duo	3.15 GHz	2GBytes	Linux	gcc 4.6.2

For type 1 machine, we can see that maximum of 5802K encryptions can be done in parallel and for Type 2 and Type 3, 4463K and 3500K encryptions .

Table 5.2: Encryption per Second for different machines

Machine	Encryptions per second
Type 1	5802K
Type 2	4463K
Type 3	3500K

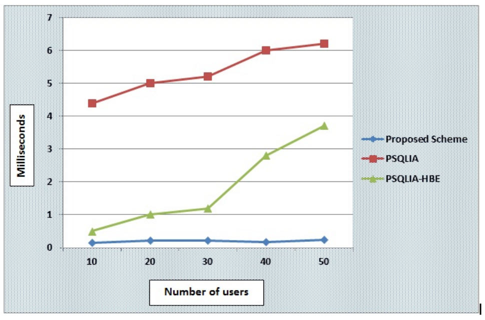


Figure 5.1: Runtime Analysis

Therefore, Bitslice AES encryption reduces much overhead of encryption process while preventing SQLIA.

Also to analyze Bitslice AES encryption, we have compared our proposed scheme with two of the existing schemes. The first scheme PSQLIA [17] is based on hash functions and the second scheme PSQLIA-HBE [18] is based on Elgamal cryptosystem. From the graph 5.1, we can see that with the increase in number of users the time for encryption also increases for both PSQLIA and PSQLIA-HBE scheme. But in proposed scheme the time to encrypt is much less then compared scheme and also it is approximately constant.

5.2 Implementation and Results using CUDA

In this section, we have considered three different systems as listed in table 5.3. In order to get the benchmark of the performance of AES on GPU using CUDA the maximum encryption per second on three machines is as shown in table 5.4.

Table 5.3: Machines Used for testing

Types	Microprocessor	Memory	GPU	O.S
1	i3	4GB 1333MHz	GeForce 9800GT	Linux
2	Core 2 Duo	2GB 2GHz	GeForce 9600m	Linux
3	i5	1 GB	GTX 460 1024M	Linux

Table 5.4: Encryption per Second for different machines

Machine	Encryptions per second
Type 1	5745K
Type 2	1795K
Type 3	10527K

For type 1 machine, we can see that maximum of 5745K encryptions can be done in parallel and for Type 2 and Type 3, 1795K and 10527K encryptions. This achieves a great performance and SQLIA attack can be handled very efficiently.

Chapter 6

Conclusion and Future Work

In this thesis, how Bitslice AES and AES encryption using Cuda can be used to prevent SQL injection attack is presented. This makes it possible to handle large users request in a much efficient manner without compromising the security. Bitslice AES technique is also free from cache timing attack and second order injections as well. Here we have also compared the performance of our proposed system with two of other techniques and found that even if number of users increases for accessing the web form but time to execute the requests always approximately constant. Still there is much possibility to enhance this work by making bitslice AES to work on GPU and other parallelization techniques.

Dissemination

Conference

- Piyush Mittal, Sanjay Kumar Jena, *A Fast and Secure Way to Prevent SQL Injection Attacks*, Proceedings of 2013 IEEE International Conference on Information and Communication Technologies (ICT 2013), Tamil Nadu, India.

Bibliography

[1] Rahul Johari, Pankaj Sharma. *A Survey On Web Application Vulnerabilities(SQLIA, XSS)Exploitation and Security Engine for SQL Injection*, 2012 International Conference on Communication Systems and Network Technologies, pp 453-458.

[2] NTAGW ABIRA Lambert, KANG Song Lin, *Use of Query Tokenization to detect and prevent SQL Injection Attacks*, 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), pp 438-440.

[3] Chester Rebeiro, David Selvakumar, A.S.L Devi, *Bitslice Implementation of AES*, International Conference, CANS 2006, Suzhou, China, December 8-10, 2006; Proceedings, LNCS, Springer, Vol 4301, pp 203-212.

[4] Eli Biham, *A Fast New DES Implementation in Software*, Fast Software Encryption 4th International Workshop, FSE97, 1997; Proceedings, LNCS, Springer, Vol 1267, pp 260-271.

[5] M. Kwan, *Bitslice implementation of DES*, http://www.darkside.com.au/bitslice

[6] V. Rijmen, *Efficient Implementation of the Rijnadael Sbox*, http://citeseer.ist.psu.edu/rijmen00efficient.html

[7] Srinivas Avireddy,Varalakshmi Perumal, Narayan Gowraj, Ram Srivatsa Kannan, Prashanth Thinakaran, Sundaravadanam Ganapathi, Jashwant Raj Gunasekaran, Sruthi Prabhu, *Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection*, 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp 1327-1333.

[8] J. Daemen, V. Rijman, *AES Proposal: Rijndael*, Version 2, AES submission, 1999, http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf.

[9] Chad Dougherty, *Practical Identification of SQL Injection Vulnerabilities*, 2012 Carnegie Mellon University. Produced for US-CERT

[10] Lori Mac Vittie, *SQL Injection Evasion Detection*, 2007 white paper, F5 Networks, Inc.

[11] A S Yeole, B B Meshram, *Analysis of Different Technique for Detection of SQL Injection*, International conference and Workshop on Emerging Trends in Technology, ACM New York, pp 963-966.

[12] Indrani Balasundaram, E.Ramaraj, *An Authentication Mechanism to prevent SQL Injection Attacks*, 2011 International Journal of Computer Applications, Volume 19 No.1

[13] D.A.Osvik, A. Shamir, E. Tromer, *Cache attacks and Countermeasures: the case of AES*, 2005, http://eprint.iacr.org/2005/271.pdf.

[14] D.J. Bernstein, *Cache timing attacks on AES*, 2005, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[15] http://www.openwall.com/lists/john-users/2011/06/22/1

[16] http://www.networkdls.com/articles/sbox.pdf

[17] Shaukat Ali, Azhar Rauf, and Huma Javed, *SQLIPA: An Authentication Mechanism Against SQL Injection*, 2009, European Journal of Scientific Research, ISSN 1450-216X Vol.38 No.4, pp 604-611.

[18] Indrani Balasundaram, E.Ramaraj, *An Authentication Scheme for Preventing SQL Injection Attack Using Hybrid Encryption (PSQLIA-HBE)*, 2011 European Journal of Scientific Research,ISSN 1450-216X Vol.53 No.3 (2011), pp.359-368

[19] Stephen W.Boyd , Angelos D.Keromytis "SQLrand: Preventing SQL injection Attacks

[20] Deguang Le, Jinyi Chang, Xingdou Gou, Ankang Zhang, Conglan Lu, *Password Recovery for RAR Files Using CUDA*, 2010 2nd International Conference on Computer Engineering and Technology

[21] Guang Hu, Jianhua Ma, Benxiong Huang, *Password Recovery for RAR Files Using CUDA*, 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing

[22] NVIDIA Corporation, NVIDIA CUDA Programming Guide Version 2.1, 2008.

[23] NVIDIA Corporation, NVIDIA CUDA Reference Manual Version 2.0, 2008.

[24] NVIDIA Corporation, The CUDA Compiler Driver NVCC Version 2.1, 2008.

[25] Rahul Johari, Pankaj Sharma, *A Survey On Web Application Vulnerabilities(SQLIA,XSS)Exploitation and Security Engine for SQL Injection*, 2012 International Conference on Communication Systems and Network Technologies

[26] Ke Wei, M. Muthuprasanna, Suraj Kothari ,*Preventing SQL Injection Attacks in Stored Procedures*, Proceedings of the 2006 Australian Software Engineering Conference (ASWEC06).

[27] Mehdi Kiani, Andrew Clark and George Mohay,*Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks*, The Third International Conference on Availability, Reliability and Security

[28] A S Yeole, B B Meshram*Analysis of Different Technique for Detection of SQL Injection*, International Conference and Workshop on Emerging Trends in Technology (ICWET 2011) TCET, Mumbai, India

[29] Qian XUE, Peng HE,*On Defense and Detection of SQL SERVER Injection Attack*, The Fund for Optional Scientific Research of Shannxi College of Communication Technology (NO. YJ10002)

[30] Daniel J. Bernstein1, Peter Schwabe2,*New AES software speed records*,

[31] Prasant Singh Yadav, Dr pankaj Yadav, Dr. K.P.Yadav,*A Modern Mechanism to Avoid SQL Injection Attacks in Web Applications*, IJRREST: International Journal of Research Review in Engineering Science and Technology, Volume-1 Issue-1, June 2012

[32] Chad Dougherty,*Practical Identification of SQL Injection Vulnerabilities*, 2012 Carnegie Mellon University. Produced for US-CERT, a government organization.

[33] Lori Mac Vittie,*SQL Injection Evasion Detection*, White Paper, Technical Marketing Manager

[34] Deevi Radha Rani, B.Siva Kumar, L.Taraka Rama Rao, V.T.Sai Jagadish, M.Pradeep,*Web Security by Preventing SQL Injection Using Encryption in Stored Procedures*, Deevi Radha Rani et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2) , 2012,3689-3692

[35] Nikita Patel, Fahim Mohammed, Santosh Soni, *SQL Injection Attacks: Techniques and Protection Mechanisms*, International Journal on Computer Science and Engineering (IJCSE)

[36] Dr K.V.N.Sunitha, Mrs.M. Sridevi *Automated Detection System for SQL Injection Attack*, International Journal of Computer Science and Security (IJCSS), Volume (4): Issue (4)

[37] Perumalsamy Ramasamy,*Sql Injection Attack Detection AND PREVENTION*, International Journal of Engineering Science and Technology (IJEST).

[38] Philipp Grabher, Johann Groschadl, and Dan Page,*Light-Weight Instruction Set Extensions for Bit-Sliced Cryptography*

[39] Tomoiag Radu Daniel, Stratulat Mircea, *AES Algorithm Adapted on GPU Using CUDA for Small Data and Large Data Volume Encryption*, International Journal of Applied Mathematics And Informatics.

[40] Peter M. Maurer, William J. Schilp, *Software Bit-Slicing: A Technique for Improving Simulation Performance*, National Science Foundation under grant number MIP-9403414.