Egyptian Informatics Journal 23 (2022) 145–162

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Provably throttling SQLI using an enciphering query and secure matching

Mohammed Abdulridha Hussain [a,b], Zaid Alaa Hussien [c], Zaid Ameen Abduljabbar [a,b,d], Junchao Ma [e,*], Mustafa A. Al Sibahee [e,f], Sarah Abdulridha Hussain [g], Vincent Omollo Nyangaresi [h], Xianlong Jiao [i]

[a] Department of Computer Science, College of Education for Pure Sciences, University of Basrah, Basrah 61004, Iraq
[b] Technical Computer Engineering Department, Al-Kunooze University College, Basrah 61001, Iraq
[c] Information Technology Department, Management Technical College, Southern Technical University, Basrah 61005, Iraq
[d] Huazhong University of Science and Technology, Shenzhen Institute, Shenzhen 430074, China
[e] College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China
[f] Computer Technology Engineering Department, Iraq University College, Basrah 61004, Iraq
[g] National Center for Management Development and Information Technology, Basrah 61004, Iraq
[h] Faculty of Biological & Physical Sciences, Tom Mboya University, Homabay 40300, Kenya
[i] College of Computer Science, Chongqing University, Chongqing 400044, China

ARTICLE INFO

Article history:
Received 7 March 2022
Revised 12 September 2022
Accepted 13 October 2022
Available online 16 November 2022

Keywords:
SQL injection
Cryptography
Searchable encryption
Web application
Internet Security

ABSTRACT

Web applications, which dominate the internet, act as communication media between customers and service providers. Web applications are an internet innovation that provide customer services such as e-banking, e-commerce and e-booking. Developing web applications has become increasingly complicated because of security threats and service issues that involve valuable information. Attack methods such as structured query language (SQL) injection insert malicious code within user input data requests to gain unauthorised access, and then the attacker targets a database to manipulate information. In this paper, we propose a prevention method against SQL injection attacks through cryptography and searchable encryption. The proposed method uses a cryptography technique to encrypt all database information, where each piece of user information is encrypted with a separate key. The rest of the database information is ciphered with secret keys, and a searchable encryption technique is used for other database operations to preserve privacy. The login process compares the ciphered username from the database and user entry to authenticate the user. The proposed method is implemented on the PHP and MySQL databases, which are open-source applications. The results show efficient prevention of SQL injection, and the database remains protected against SQL injection attacks

1. Introduction

The industry is moving toward web applications to support remote services with ubiquity and cost efficiency [1]. Organisation and companies rely on web applications to deliver services to customers while reducing the cost to a minimum because the infrastructure is based on the internet. Services such as e-banking, shopping, e-governance and reservations increase productivity in daily life due to speed and utility flexibility [2].

Web applications are built upon a number of program layers in tier architecture to handle complexity and specific functions. The basic internet technology architecture is client–server, whereas a web application adds a third level (i.e., database tier) to manage information [3]. The database contains records for managing the web application and authenticating users. However, any breach in web application authorisation leads to the disclosure of all database records and information [4]. Access to the database is controlled by access control policies, but other data operations are not controlled or limited. In other words, access policies define how to access but not how to use data [5].

* Corresponding author.
E-mail address: majunchao@sztu.edu.cn (J. Ma).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.eij.2022.10.001

The rapid development of the internet has become a primary concern of businesses, and the growing amount of information online has led to increasing concerns about information security and threats. Moreover, businesses rely on the internet for valuable transactions such as banking, which caused attackers to attempt access to web application databases [6]. One web application attack is known as a Structured Query Language (SQL) injection attack (SQLI), which targets the database for unauthorised access. SQLI attacks are classified as severe and harmful for web applications because SQLI can gain unauthorised access, manipulate database information and may result in denial of service [7].

According to the Open Web Application Security Project (OWASP) [8], SQL injection, which is defined as injecting untrusted data into normal commands or queries to gain access to a database, is the major threat to web application databases [9]. The attacker injects malicious code in the SQL query that targets the back-end web application database [10]. An attacker will cause a security breach to a database, wherein the severity of impact depends on the type of SQL injection and the web application problem [11]. An attacker's intention involves a transaction between a web server and a database, and the way SQL is written and called. The SQLI method uses the normal SQL statement and rules, which is the reason that the server–database communication needs to be validated or the mode of communication needs to be hidden [12,13].

The attacker's code is a normal text format which is interpreted by the web server as an SQL command and executed in the database. Thus, the code injected does not contain any special characters that further complicate detection [14]. Prevention is added in complex analysis processes to the web server. This causes delays and requires page coding for each user input field. The lack of a prevention technique in a web application has severe consequences such as loss in confidentiality, integrity and authentication [15].

In this paper, we present a method to prevent SQLIs. The proposed method is based on the encryption of the database information and searchable encryption. To authenticate the user, the username is encrypted using a symmetric cipher with a key equal to a password. Searchable encryption is used to preserve the privacy of the information while searching records. The results show that the proposed method prevents SQLIs, and the results are comparable with those of the newest methods in terms of time and size.

Fig. 1 illustrates the dominance of PHP in building web applications [16]. Some solutions of SQLI are based on the language level [2,6,17,18,19], whereas PHP is targeted for the above reason. The proposed method is based on a database and not on a specific vendor or language, making the solution highly robust. In other words, the proposal is independent of server compiler language, so it can be deployed on any platform.

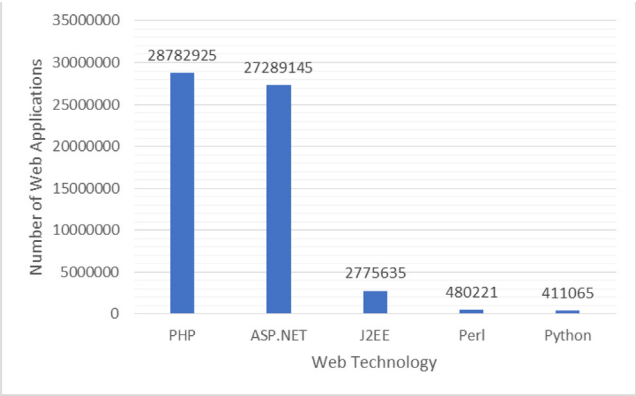

Fig. 1. Usage statistics of web applications [16].

The main contributions of this work are as follows:

- We present a method to prevent SQLI.
- The proposed method can protect web applications such as e-banking from injection attacks.
- The encryption of information resists tautology and comment attacks.
- Searchable encryption is presented to withstand other types of SQLI.
- The proposed method is not specified for a platform or any programming language.
- The results show that the proposal to counter SQLI does not affect the system performance.

The remainder of this paper is structured as follows:

Section 2 provides preliminary and contextual information. Section 3 describes previous works related to the subject. Section 4 explains the proposed method. Section 5 demonstrates the implementation and the experiments carried out to verify the proposal. Section 6 presents the results of the experiment and a discussion. Section 7 explains the security analysis carried out, and the final section presents the conclusion and future works.

2. Preliminaries

2.1. SQL injection attack

SQL injection attacks exploit vulnerabilities in a database layer [20]. Most vulnerabilities are caused by developers' lack of knowledge about SQLI threats or that they ignore such vulnerabilities because a secure web application means higher costs. The following are some of the most popular types of SQLI [21] and [22].

1) Tautology

An attacker exploits the condition format after (WHERE) clustering in the SQL statement. The basic condition for a login query is to check the username and password. Meanwhile, an attacker injects an input that will always be true for the condition. The result of such an attack is to gain access to a web application. Table 1 provides an example of a tautology attack.

2) Commenting

An attacker inserts SQL comments, which are denoted by a double dash: "--." Practically, when a username is known, an attacker comments the password fields to achieve an unauthorised login. Table 2 provides an example of a commenting attack.

3) Piggybacked

The objective of such an attack is to insert a query to be executed with the legal or original query. The second query is not built by the developer, which means the web server cannot handle the result. Hence, this scenario limits the attacker query options without retrieved information. The basic query is updated and inserted, but most of the used queries are deleted. This attack may be executed on any page such as login, search or insert, and the result directly affects the database without raising an alert. Table 3 provides an example of a piggybacked attack.

4) Union

An attacker inserts "UNION" after the original query to retrieve more information about the original queried table. The result of the

Table 1
Example of tautology attack.

Attacker input
Username = a' or 'a'='a Password = a' or 'a'='a *Web server generate query* SELECT * FROM users WHERE username = 'a' or 'a'='a' and password='a' or 'a'='a'; *Database result* Attacker gain access to web application as legal user.

Table 2
Example of commenting attack.

Attacker input
Username = admin' – Password = xx *Web server generate query* SELECT * FROM users WHERE username = 'admin' – and password='xx'; *Database result* Attacker gains access to web application as "admin."

Table 3
Example of piggybacked attack on search page.

Attacker input
search field (id) 5; drop table users; *Web server generate query* SELECT * FROM users WHERE id = 5; drop table users; *Database result* Users table is deleted from database, which prevents access to web application

Table 4
Example of union attack.

Attacker input Note:(admin, moh) is legal username and password pair
Username = admin Password = moh' UNION ALL SELECT * FROM users; *Web server generate query* SELECT * FROM users WHERE username = 'admin' and password = moh' UNION ALL SELECT * FROM users; *Database result* Database displays user information to attacker

attack is to collect information for future use, but the attacker has already gained direct access to the SQL database panel. Table 4 provides an example of a union attack where the attacker has an authorised account and tries to retrieve all other accounts via a union attack query.

5) Alternate encoding

The alternate method is based on transferring a query into a code explainable by a database, such as ASCII. An attacker uses this method to defraud a web server phrase filter, which is used to block the injected query. Table 5 provides an example of an alternate encoding attack where the attacker tries to override the stored procedure keyword filter by changing "shutdown" to hexadecimal encoding and pass the procedure through the search page.

6) Logical incorrect

An attacker's objective is to collect information about a database such as table names and column properties. The technique is based on inserting incorrect symbols into the query to activate a database exception to a user, which will display an error notification such as a table name. Table 6 provides an example of a logical incorrect attack where the attacker tries to insert a symbol in the username field, which results in a database error.

Table 5
Example of alternate encoding attack.

Attacker input Note: char(0x73687574646f776e) is "shutdown"
search field (id) 5; exec(char(0x73687574646f776e)); *Web server generate query* SELECT * FROM users WHERE id = 5; exec(char(0x73687574646f776e)); *Database result* The database turns off (shutdown) when query passes any character filter mechanism.

Table 6
Example of logical incorrect attack.

Attacker input
Username = test' Password = test *Web server generate query* SELECT * FROM users WHERE username = 'test'' and password= 'test'; *Database result* Display error (syntax to use near '' and password = test' at Line 1)

7) Blind

An attacker needs a GET HTTP request to execute a blind attack, which is based on inserting malicious code into the Uniform Resource Locator (URL). The objective of this attack is to gather information about the database by trying or comparing letters by letters. An automatic tool such as SQL-map is proposed because the manual method is rigorous and time-consuming. SQL-map and other tools make a blind attack easy to perform. Table 7 provides an example of a blind attack where the attacker tries to find if any table in database starts with "a" by condition result. In other words, the search page will display if there is a table starting with "a.".

8) Stored procedure

Database application becomes increasingly complicated, which presents new items such as procedures and triggers. The procedure is a code stored at the database end to perform a special task. When an attacker runs a procedure remotely, a stored procedure attack is achieved. Table 8 provides an example of a stored procedure attack where the attacker tries to shut down a database remotely through a stored procedure.

2.2. Searchable encryption

The well-known method to secure information uses cryptography techniques, especially in a public environment such as the cloud. As the name implies, searchable encryption allows searching over encrypted information to preserve privacy [23].

One of the searchable encryption approaches is based on keyword search. The information is encrypted and stored in the database with information keywords in plain text. The database retrieves records according to keywords [24].

2.3. System architecture model

The web application is a model based on the internet environment for communicating model parties and transferring data

Table 7
Example of blind attack.

Attacker input: on the URL
https://localhost/nor/searchbook.php?nam = 5 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="DBname" AND table_name REGEXP '^[a]' LIMIT 0,1)
Web server result
SELECT * FROM users WHERE username = 'test" and password= 'test';
Database result
I If the search page displays a result page, then the database contains a table name starting with "a." When there is no page display, there are no tables name starting with "a"

Table 8
Example of stored procedure attack.

Attacker input
search field (id) 5; exec(SHUTDOWN);
Web server generate query
SELECT * FROM users WHERE id = 5; exec(SHUTDOWN);
Database result
The database turns off (shutdown).

[25]. Fig. 2 shows three theoretical parties, namely, the client, server and database. Practically, a client connects to a network and provides a graphical user interface (GUI) for a user through a web browser. A server handles client request–response using web server applications such as Apache and connects to a database that is mostly on the same server [4].

A web application communicates to the database owing to SQL statements that restrict inserting, updating, deleting and selecting commands. The use of SQL limits GUI pages to add, modify, delete, search and login, in which the first three are handled in the same way [26].

The modification or deletion of records from a database is executed practically in two main scenarios.

Scenario A for modifying records:

1) A user requests the page.
2) The table main column is presented to the user as a drop-down selection.
3) The user selects one selection and submits it to the server.
4) The fields of the selected record are viewed.
5) The required field(s) are modified.
6) The table is updated.

Note: When deleting records, the selected record is deleted after step 3.

Scenario B for modifying records:

1) A user requests the page.
2) A page with an empty text box is viewed.
3) The value of the table main column (e.g., id) is entered and submitted to the server.
4) If the value exists, then view the fields. Otherwise, return an error page.
5) The required field(s) are modified.
6) The table is updated.

Note: When deleting records, if the value indicated in step 4 exists, then the selected record is deleted.

3. Related works

The injection attack is claimed until reaching the first threat on a web application according to OWASP 2021. In recent years, a considerable amount of research has been conducted to detect and prevent SQLI. The following are some proposed methods based on encryption and encoding mechanisms to prevent SQLI.

Raj et al. [17] proposed a coded format to store data. This method is based on the reversal insertion algorithm (RIA). After reversing the input, a special character between each group of two letters is inserted to cancel the (OR) appearance. The proposal evaluates only the login page, and from Algorithm 2, multiple computation and comparison combinations load to the web server, which consumes more time.

D'silva et al. [27] proposed preventing SQLI by using the hashing technique. The method calculates the hash value of the login query with the correct values when a user creates and compares the results of the query hash value during the login phase. The disadvantage of this method is that it can only be applied to the login phase, and compression adds extra processing to the web server.

Anjugam et al. [28] proposed Advanced Encryption Standard (AES) encryption on token SQL queries. The token generates a dynamic client-side table, which is encrypted and sent to the server with the original query. The server generates a lookup table from the user query, decrypts the client dynamic table and then compares both tables. If both tables are equal, then the query is forwarded to the database; otherwise, the query is blocked. However, this duplicates the request size, which is sent to the server, and consumes bandwidth between the client and server. The need for a client to store a dynamic table may be blocked by blocking the client-side script.

Kumar et al. [29] presented a method that is divided between a client and server. The client-side function filters sensitive characters, and the query length is retrieved from the database, whereas the server-side function compares the message authentication

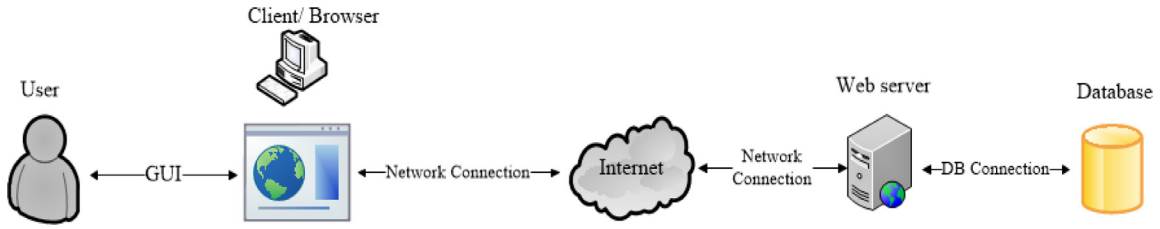

Fig. 2. System architecture model.

code (MAC) value during the user creation and login phases. Retrieving information at the client-side filter increases traffic, and the method protects only the login phase.

Namdev et al. [18] used the hash technique for securing logins. The hash value computes for the username and password and then calculates the EXOR between them. The login procedure depends on the EXOR value to authorise a user. The approach protects only the login from tautology attacks, and the EXOR calculation is evaluated on a web server.

Avireddy et al. [19] proposed a coding-based scheme where each character maps up to four possible random characters. The selected value from the table is based on the next character. The lookup table must be maintained on the client and the server sides, which is the main drawback of the scheme.

Balasundaram et al. [30] proposed a scheme based on AES to verify a user. The database stores the username, password and secret key for each user. The client encrypts the username and password before being sent to the server. The server verifies the username and password after decrypting the received query by using the stored key in the database table. The disadvantage of such a scheme is that users must enter their username, password and secret key for each login. The scheme prevents only tautology attacks.

Zhu et al. [31] proposed modular automation of the SQL statement in which the module maintains a list for a known anomaly pattern. The disadvantage is centred on pretreatment, which consumes considerable time in a web server due to the extraction of the pattern and extra size to store known patterns.

Dalai et al. [32] suggested comparing the string after the 'where' cluster in two cases. The first case is without user input, and the second case has user input after removing the type of correct variable. Their work was evaluated using Java and only handles select clusters. The author evaluated the tautology and removes the alphabet without removing ('a'='a'). The main drawback is that the method does not solve the insert and update clusters.

Xiao et al. [33] detected SQL injection through behaviour and response analysis. The method calculates the normal state and stores the results in the database. When a user requests, a service is executed and compared with the stored result for any unusual value. The first disadvantage is the overwhelming amount of data, and the second disadvantage is that the request is first executed and then blocks the user. Finally, saving the user IP and port in a blacklist does not guarantee blocking the user where the user may run some proxy.

Ghafarian [26] used a hybrid method from static and dynamic analysis to prevent and detect SQLI. The main method is based on extracting table names and conditions from the input query and then reconstructing a SELECT query. If the result returns the symbol from the database, then the query is rejected. The method prevents only tautology attacks.

The proposed method is based on using encryption information in the database to prevent SQLI. The encryption is used only with the insertion of information and keyword search, which can increase the time in contrast to the security offering. Searchable encryption is used to preserve information privacy and save decryption time.

4. Proposed method

According to Section 2, a security breach occurs within user input fields, in other words, HTML text box fields. The proposed method is extended from [34], which is based on encrypting all user input before processing. This method helps defuse all malicious codes. Encryption refers to a symmetric cipher that is being handled at the database level, as shown in Fig. 3.

The related studies state that the majority solution covers only the login phase/page. In contrast, the proposed scheme solves login and other pages through encryption and searchable encryption. Subsection 2.3 shows that the majority of website pages deal with a database to store web application data and information. In such web applications, login, insert and update pages can be protected by encrypting user input. However, in this proposal, the search page can be protected by applying searchable encryption. The role of searchable encryption is to preserve privacy to the database's

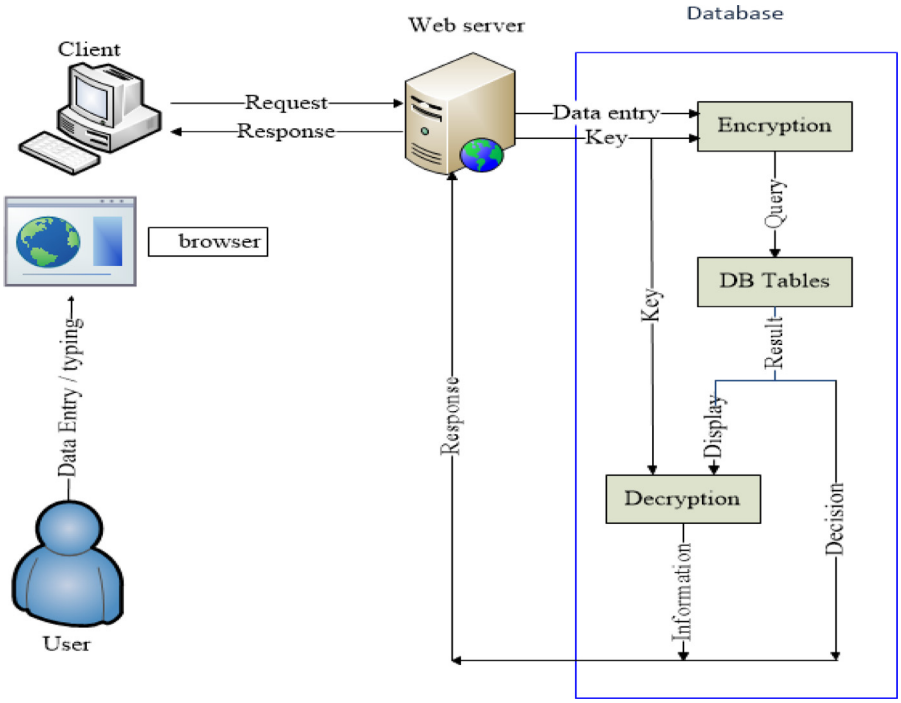

Fig. 3. Proposed method.

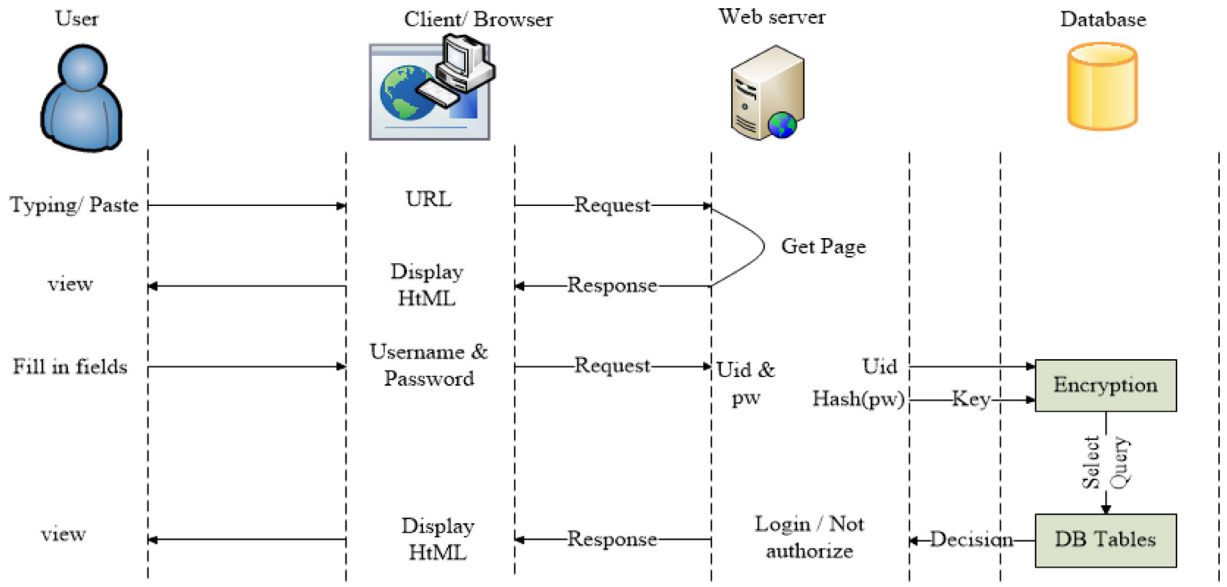

Fig. 4. Login procedure.

Table 9
Login checking code.

```
$userid = $_POST['userid'];
    $password = $_POST['password'];
    @ $db = new mysqli('localhost','root','','libmoh');
    $pw = hash('sha3-512', $password);
    $query = "select * from users where username = AES_ENCRYPT
    ('$userid','$pw')";
    $result = $db->query($query);
    if ($result->num_rows) echo "Authorized Login"; else echo "Not
    Authorized";
```

encrypted information, so the database information is not revealed by data decryption.

The following subsection will explain the proposed scheme on the basis of page type and procedure (each entity function and messages).

4.1. Login procedure

A user calls this procedure every time to access web applications. A user first calls a login form that contains an empty text box for the username and password. The form is displayed as an HTML page. Afterwards, a user fills in the text fields, and the browser submits the data to a web server script such as PHP. The web server extracts the data from the HTTP request, calculates the secret key and submits the data to the database. The key is generated through a hash function (SHA3 [35]) to the password text value. The database engine encrypts the username using the secret key and executes a select query for the cipher username. If found, then the user is an authorised user; otherwise, an error page is displayed, as shown in Fig. 4 and Table 9.

Remark: MD5, SHA1 and SHA2 suffer from collision resistance, which is an easier approach than a brute-force attack. While SHA-1

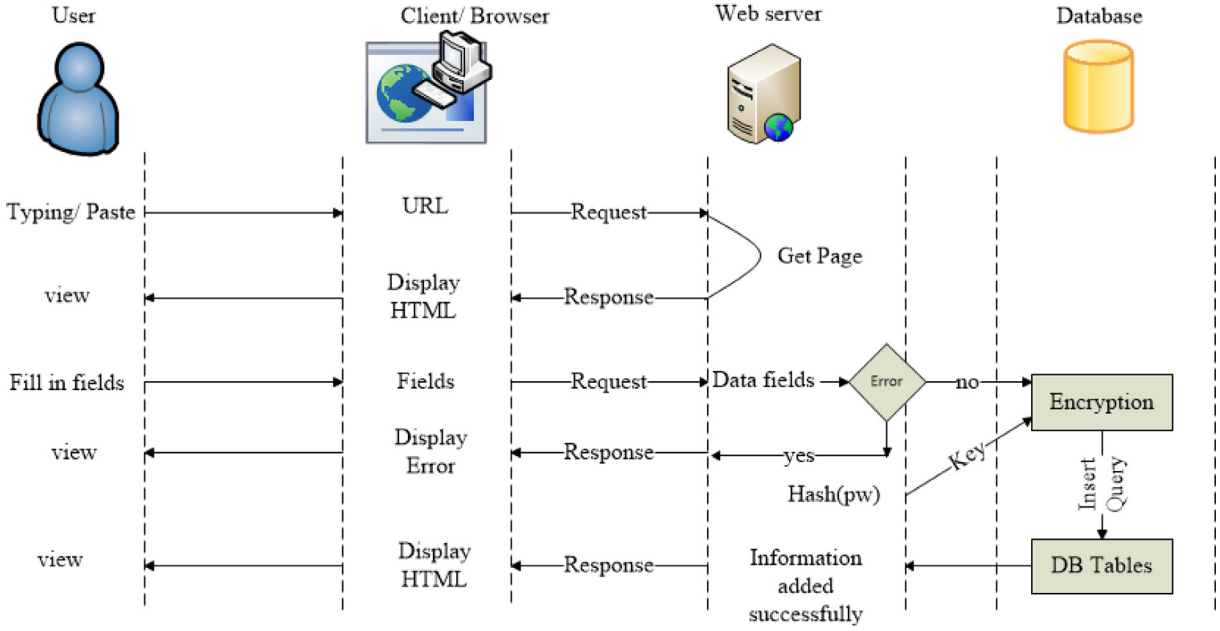

Fig. 5. Insert procedure.

and SHA-2 are mathematically and structurally comparable, SHA-3 is more secure due to its collision and preimage resistance. In light of this, SHA-3 is increasingly replacing other hash functions [36,37].

4.2. Insert procedure

A user executes the insert procedure when entering the user account with the authorised username. The first form is in HTML and contains the necessary information with username and password pairs. In modern forms, fields are required to be completed by users, and any missing information causes the server to reject the information and display an error. In normal scenarios, a web server generates a secret key, which is the hash value (SHA3) from the password and the secret key used to encrypt (e.g., AES [38,39]) the username to store ciphered in the database. As shown in Fig. 5 and Table 10, if the insertion is successful, then the web server returns to the user an HTML page containing the succeeding insert procedure.

Table 10
Insert code.

```
$un=$_POST['n1′];
```

```
$pw=$_POST['n2′];
@ $db = new mysqli('localhost','root','','libmoh');
$pw1 = hash('sha3-512′, $pw);
$query = ''insert into users values(NULL,AES_ENCRYPT('$un','$pw1′))";
$result = $db->query($query);
if ($result) echo $un.'' is Added"; else echo "Error";
```

Remark: The data encryption standard (DES) suffers from brute-force attacks, whereas the advanced encryption standard (AES) is the solution to DES drawbacks [40]. The AES key is the default size, which is 128 bits. The SHA3 output is more than sufficient compared with the key length, but such a size will deceive attackers as a man-in-the-middle attack, and the key driven from SHA3 will not be saved. In other words, there is no waste in database information.

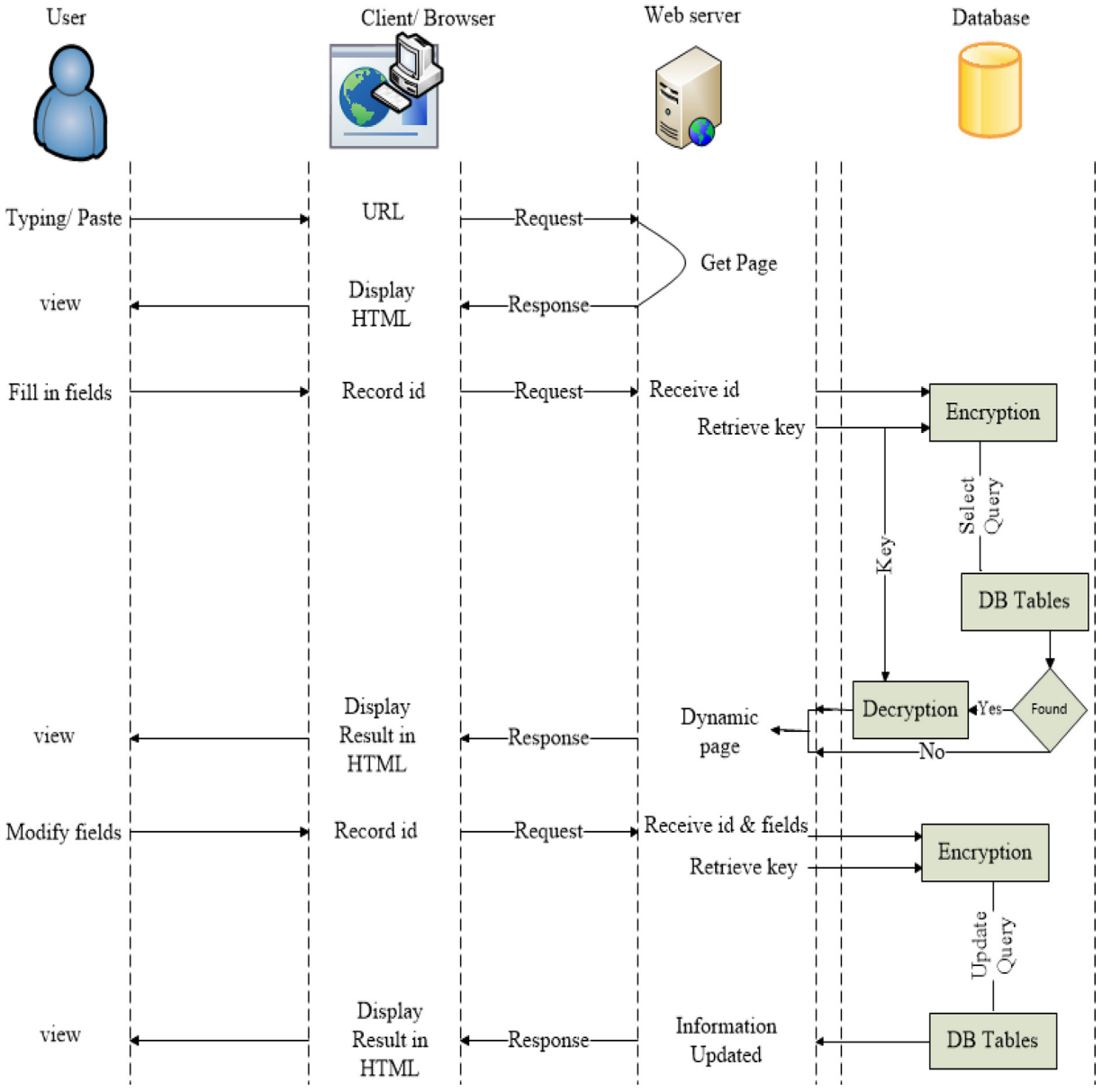

Fig. 6. Update procedure.

Table 11
Update code.

```
$id=$_POST['n1′];
@ $db = new mysqli('localhost','root','','libmoh');
$key = hash('sha3-512′, 'subber');
$query = "select * from books where id = AES_ENCRYPT('$id', '$key')";
$result = $db->query($query);
if ($result)
{
//View the record
echo "<table border = 1>";
echo "<tr><td > id</td><td><input type='text' name='n1′ value = AES_DECRYPT(id, '$key')></td></tr>";
echo "<tr><td > Book Title</td><td><input type='text' name='n2′ value = AES_DECRYPT(title, '$key')></td></tr>";
echo "<tr><td > Author Name</td><td><input type='text' name='n3′ value = AES_DECRYPT(author, '$key')></td></tr>";
echo "<tr><td > Details</td><td><input type='text' name='n4′ value = AES_DECRYPT(details, '$key')></td></tr>"; echo "</table>";
}
else echo "Error: Not Found";
Update the Record
$id=$_POST['n1′]; $ti=$_POST['n2′]; $au=$_POST['n3′]; $de=$_POST['n4′];
@ $db = new mysqli('localhost','root','','libmoh');
$key = hash('sha3-512′, 'subber');
$query = "update books set title = AES_ENCRYPT('$ti', '$key'), author = AES_ENCRYPT('$au', '$key'), details = AES_ENCRYPT('$de', '$key')where id = AES_ENCRYPT('$id',
    '$key')";
$result = $db->query($query);
if ($result) echo "Updated"; else echo "Error";
```

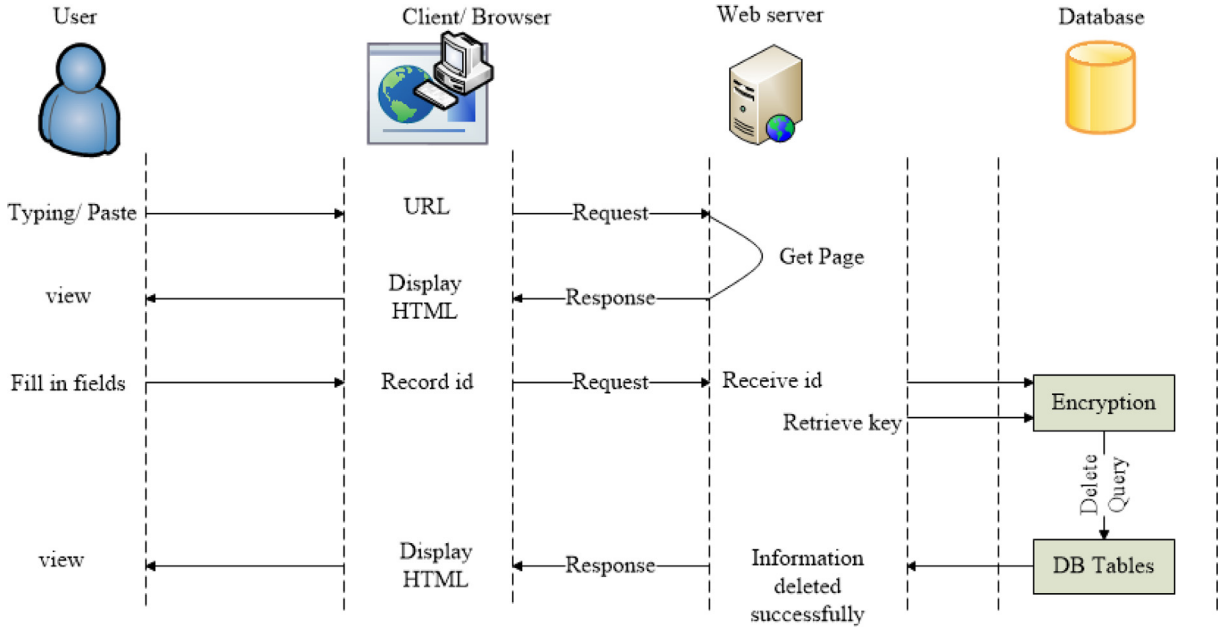

Fig. 7. Delete procedure.

Table 12
Delete code.

```
$id=$_POST['n1′];
@ $db = new mysqli('localhost','root','','libmoh');
$key = hash('sha3-512′, 'subber');
$query = "Delete * from books where id = AES_ENCRYPT('$id', '$key')";
$result = $db->query($query);
if ($result) echo "Deleted"; else echo "Error: Not Found";
```

4.3. Update procedure

In this proposal, the update procedure is based on scenario B in Section 2.3 because scenario A causes decryption of all table main columns in the server to view in step 2.

The user input is encrypted using a secret key stored in the web server and executes a select query in the database to find the requested record. The database decrypts the found record, which is displayed to the user by the web server. After a user submits the required modification, the database encrypts the fields and updates the table record, as shown in Fig. 6 and Table 11.

4.4. Delete procedure

The delete procedure is based on scenario B in Section 2.3 and is demonstrated in Fig. 7 and Table 12. In this procedure, user input is encrypted and a select query is run. The records are deleted if found.

4.5. Search procedure

The search procedure in this proposal is based on searchable encryption because the database information is encrypted. Search-

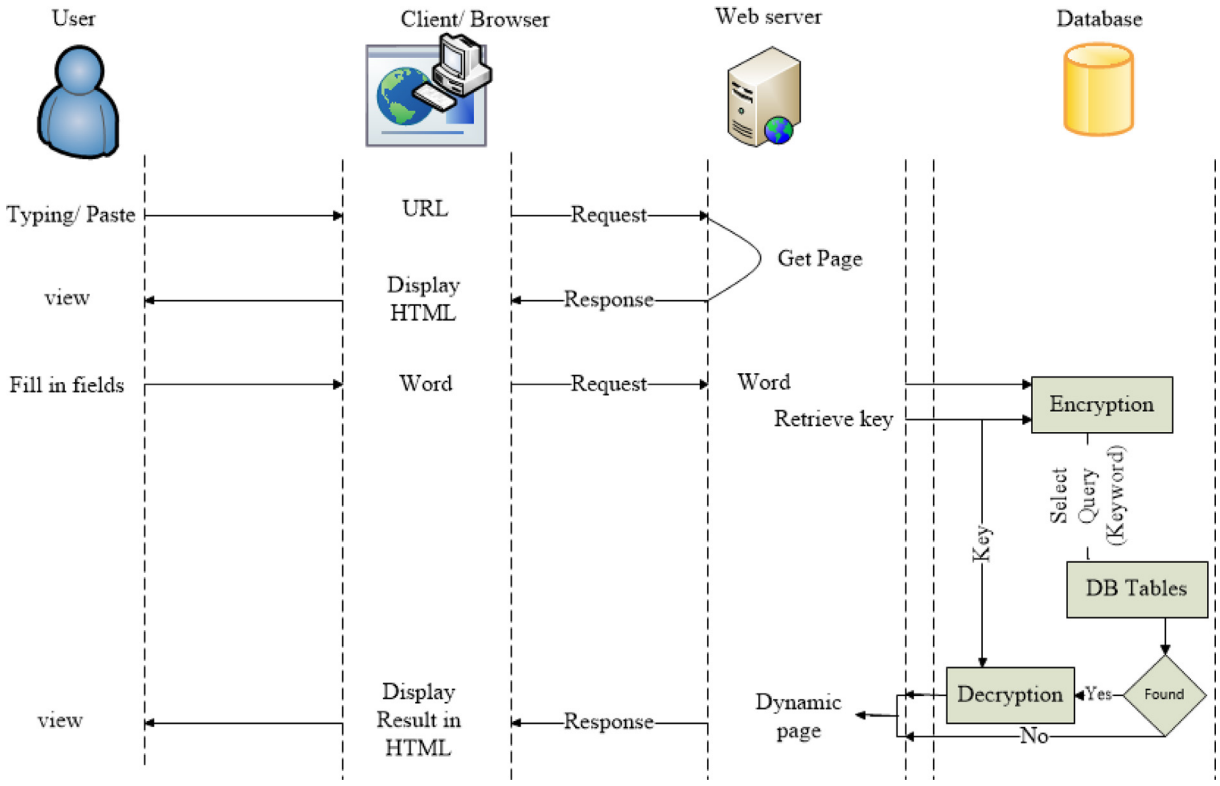

Fig. 8. Search procedure.

Table 13
Search code.

```
$nm=$_GET['nam'];
```

```
@ $db = new mysqli('localhost','root','','libmoh');
$key = hash('sha3-512', 'subber');
$query = "select id, AES_DECRYPT(title, '$key') ti,AES_DECRYPT(author,
    '$key') au, AES_DECRYPT(details, '$key') de from books where kw like
    AES_ENCRYPT('$nm', '$key')";
$result = $db->query($query);
$num_results = $result->num_rows;
for ($i = 1; $i <=$num_results; $i++)
{
$row = $result->fetch_assoc();
$id=$row['id']; $ti=$row['ti'];$au=$row['au']; $de=$row['de'];
echo "<table border = 1>"; echo "<tr><td > id</td><td>$id</td></tr>";
echo "<tr><td > Book Title</td><td>$ti</td></tr>";
echo "<tr><td > Author Name</td><td>$au</td></tr>";
echo "<tr><td > Details</td><td>$de</td></tr>"; echo "</table>";
}
```

able encryption based on keywords is used, as mentioned in subsection 2.2, but here the keywords are encrypted and stored in the database. The procedure starts by encrypting the user keyword and executing the selected query in the database. If the keyword is found, then it will display the related records to the user after decryption, as shown in Fig. 8 and Table 13.

5. Implementation and experimental tests

The attack occurs at the application layer where the browser defines the protocol rules [41]. The browser handles normal client and attack requests in the same way regardless of the application layer protocol carrying the request. Practically, HTTP or HTTPS will handle browser requests, as in [6,17,32,33]. HTTP is used in the experiment. The HTTP helps the network analyser program read messages that are transferred in plain text.

5.1. Implementation

The proposal is implemented using XAMP 7.2.11 [42] as a localhost server, which contains Apache 2.4.35, PHP 7.2.11 and MySQL 5.0.12. The web application consists of creating user accounts, logging in, logging out and search functions. The ideal implementation with previous proposals is implemented for comparison purposes. These are listed in Table 14.

The main database tables are users, table1 and books, which contain book information from the dataset in [43]. The dataset information is split into title, author, details and keywords, where the detailed column contains other uncategorised data. 'table1′ is meant to test piggybacked drop table attacks. If successful, table1 is removed. The 'users' table is meant for storing authorised user information for login purposes. The table 'books' is the same as all previous proposals except the proposed method, and the user table is different for all proposals, as shown in Table 14.

The created user account function is implemented by using two pages. The first is the HTML page, and the second is the PHP page to perform the insert query to the 'users' table. The login function contains a single PHP page, which executes select queries to the 'users' table. If successful, then the session is created. The logout is a single PHP page that ends the session and empties the session variable. Finally, the search function is based on two pages, HTML and PHP, which perform the selected query to the books table and display the results. Table 15 shows the list of functions with related files for each function.

5.2. Experimental tests

SQL attacks can be categorised depending on the level of injection, and the experiment is chosen according to each category. The first category contains attacks that gather database information by injecting malicious codes in URLs or by incorrect requests. This cat-

Table 14
Implementation and table columns of proposed method.

Proposal	DB name	Tables and columns
Normal (Ideal)	libnormal	users (id, username, password) table1(id, name) books (id, title, author, details)
Raj [17]	libraj	users (id, username, password, keyvalue, userlen) table1(id, name) books (id, title, author, details)
Dsilva [27]	libdsilva	users (id, username, password, hashdigest) table1(id, name) books (id, title, author, details)
Namdev (ENCP)[18]	libencp	users(id, username, password, hashusername, hashpassword, hashexor) table1(id, name) books (id, title, author, details)
Kumar (MAC) [29]	libmac	users (id, username, password, qlength, smac) table1(id, name) books (id, title, author, details)
Our proposal	libmoh	users (id, username) (moh) table1(id, name) books (id, title, author, details, keywords)

Table 15
Functions and files.

Functions	File name	Called file name
Create user account	inuser.html	adduser.php
Login	authmain.php	authmain.php
Logout	logout.php	authmain.php
Search	search.html	search.php

egory includes blind and logical incorrect attacks, which are based on trial and error and can be experimented with easily using the sqlmap tool. All pages use the HTTP POST request in the Moh proposal to withstand the first category attacks, as shown in Fig. 9.

The second category includes the Union attack, which needs console access for the database because the victim site did not prepare to handle the Union result; in other words, there is no experimental test. The Moh proposal protected against Union attack; this is proven in the security analysis section if the site is prepared to handle all client queries. The third category is targeted at the database procedures, which include stored procedures and alternate encoding attacks. The proof is in the security analysis section.

The fourth category is targeted at the site input fields, which include tautology, commenting and piggybacked attacks. The experiment uses a localhost server, which runs Apache to handle HTTP messages, a PHP compiler and MySQL database. Two situations, without and with attack, are used to evaluate browsing web applications. Each function of the proposed method or solution is evaluated on the basis of the following experimental procedure.

1) Create user account
   a) Run the browser application on the client.
   b) Call inuser.html using a browser address bar.
   c) Enter the username and password.
   d) The server calls adduser.php and sends insert query to the database.
   e) The server displays the 'user added successfully' message.
2) Login
   a) Run the browser application on the client.
   b) Call authmain.php using the browser address bar.
   c) Enter the username and password for login.
   d) The server calls authmain.php and sends the select query to the database.
   e) If the user is authorised, then the server sets a session variable and displays login successfully.
   f) If the user is not authorised, then the server displays an error message.
3) Logout
   a) The user selects the logout link when the user successfully logs in.
   b) The server calls logout.php and unsets the session variable.
   c) The server displays the 'successfully logged out' message.
4) Search
   a) Run the browser application on the client.
   b) Call search.html using browser address bar.
   c) Enter search text or keyword.
   d) The server calls search.php and sends a select query to the database.
   e) The server displays the results to the user.

The attack scenario injects the malicious code into the login and search functions. The attacker uses the tautology and commenting type of SQL injection on the login and a piggybacked attack on the search function. Table 16 illustrates the attacks.

6. Results and discussions

The data collected from the above experiments are categorised depending on functions without and with attacks. These functions are mentioned in Section 5. The data are captured for each procedure using Wireshark [44], where the information is extracted on the basis of HTTP messages. HTTP messages are requests and

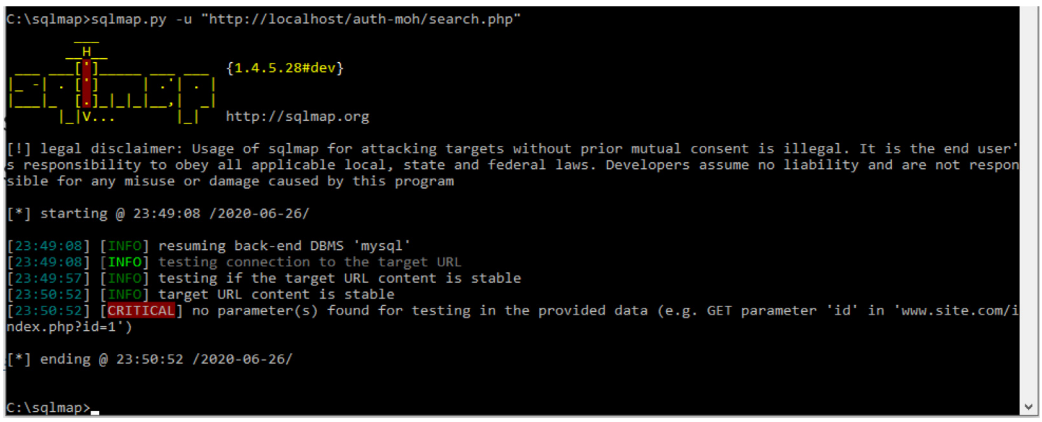

Fig. 9. Experiment SQLmap on Moh proposal.

Table 16
Text input in attacks.

Attack	on function	Input text
Tautology	Login	username: a' or 'a'='a password: a' or 'a'='a
Commenting	Login	username: admin' – password: x
Piggy-backed	Search	London'; drop table table1

responses that have basic metrics of time and size. Round trip time (RTT) is the time difference between the arrived response and sent request. RTT includes the request time to reach the server, web server calculation time, database access and retrieval time, and time spent from the server to the client. Size indicates the amount of data transfer between the client and server.

Table 17 shows the web server pages, tables and database sizes. Some file sizes are fixed because the current proposal does not affect page codes, such as inuser.html, logout.php and search.html. The size changes in adduser.php can be ignored because the page is called once for each user. The authmain.php page is called frequently for each login where Kumar's file size is the largest and all others are within the normal state. Moh has the largest file size for search.php caused by the keyword comparison code. The book table size is greater because of the encrypted keywords for each record. The number of columns for each proposal will affect the user table size, where the Namdev table is the largest because of the number of details for each user.

The proposal results were compared with other approaches and methods that prevent SQLI attacks in web applications. The results are shown in the next subsections. The following results are divided on the basis of the function, and a result value of (0) means the attack is blocked.

6.1. Create account

The created account function is tested against RTT, HTTP request and HTTP response size to compare different proposals. This function is not tested against attacks that are tested in depth with a search function.

Fig. 10 and Table 18 demonstrate the RTT for the creation account function, which includes the server processing time. The Normal is the shortest RTT, which is evident because no encryption or encoding process occurs. Namdev is the longest RTT because of the number of processes where the hash function is calculated twice and the EXOR operation for 160 bits. The Moh proposal is the second longest, but this function is generated or called once for each user.

The call web page sizes for request and response are the same for all proposals because the content is equal. This is essentially the HTML page part, as shown in Fig. 11 and Table 19. The created account request has the same value because the information carried is the username and password. Finally, the response size for

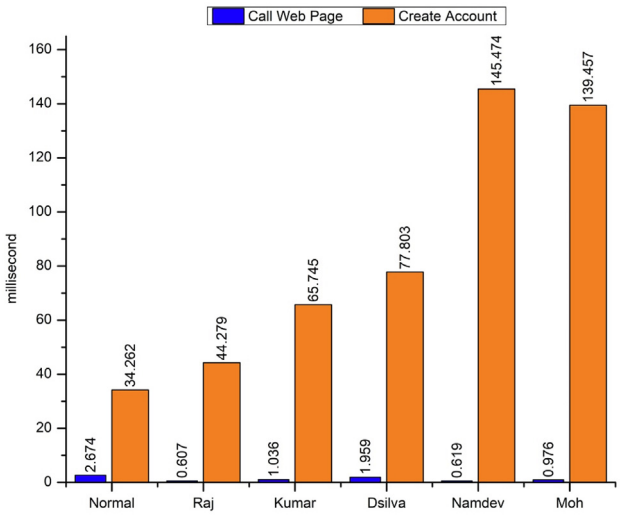

Fig. 10. RTT created account.

creating an account is based on the server HTML response after successfully creating the user.

6.2. Login and logout

The logout function occurs when a user successfully logs in authorised or unauthorised user scenarios. If no logout occurs, then the attacker is blocked.

From Figs. 12, 13, 14 Tables 20, 21 and 22, the call web page is calculated from the average where all proposals take almost the same RTT, request and response size. The logout without attack is equal for all proposals in all metrics. Kumar login is the longest time and largest size because the number of operations is taken into account, whereas the call page is large in response size because of predefined conditions and values. Moh has the second largest RTT value because of the encryption operation. All proposals block tautology attacks with different RTTs, which is based on the proposal complexity.

Notably, Namdev and Dsilva do not block or prevent commenting attacks, which is prevented by other proposals, and the Moh proposal takes a shorter RTT value. The response size is within average means, RTT expresses the server decision time, which varies, and Kumar takes the longest time to evaluate.

6.3. Search

The request size from Fig. 16 and Table 24 and the response size from Fig. 17 and Table 25 explain the search function for all tested proposals. All tested proposals have the same size in scenarios without and with attacks. However, the Moh proposal for response sizes under attack is different from the other scenarios because the

Table 17
Size of files, table and database (Bytes).

	Normal	Namdev	Raj	Dsilva	Kumar	Moh
inuser.html	381	381	381	381	381	381
adduser.php	684	826	900	756	774	717
authmain.php	1558	1725	1892	1734	3723	1589
search.html	295	295	295	295	295	295
search.php	1185	1183	1182	1185	1182	1223
table1.sql	995	993	992	995	992	992
users.sql	1518	2204	1698	1746	1788	1463
books.sql	317,543	317,541	317,540	317,543	317,540	545,602
database size	320,056	320,738	320,230	320,284	320,320	548,057

Table 18
RTT created account (milliseconds).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	2.674	0.607	1.036	1.959	0.619	0.976
Create Account	34.262	44.279	65.745	77.803	145.474	139.457

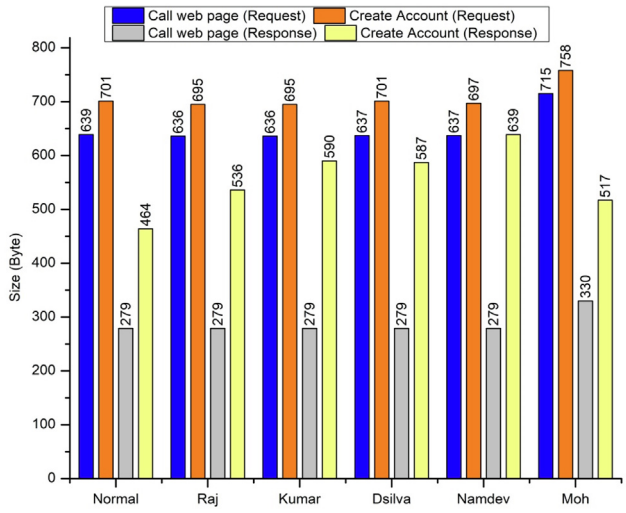

Fig. 11. Created account (request–response size).

attacker is blocked from dropping the table. All input for Moh is encrypted to generate the keyword that will defuse a piggybacked attack.

Moh takes the longest time in RTT (Fig. 15 and Table 23) but prevents piggybacked attacks during the time a decision is made. On the one hand, this time period is reasonable compared with the benefit of preventing a piggybacked attack. On the other hand, the time delay in Moh is not very long in contrast to other tested proposals and even the normal technique..

6.4. Discussions

HTML pages have fixed averages for RTT, HTTP request and HTTP response size. The only difference is in PHP files or the server-side script because that response is based on processing user input to make decisions or to generate dynamic HTML pages.

The results indicate the robustness of the proposal (Moh). Over time, the proposal prevents tautology, commenting and piggybacked attacks. No other proposals prevent or detect the attacker related to the search function, such as a piggybacked attack. Kumar's proposal suffers from a long processing time to prevent tautology and commenting. Raj's proposal calls for a shorter time to prevent login attacks because no cryptography is proposed. Namdev and Dsilva prevent only tautology attacks and even the use of cryptography.

7. Security analyses

Theorem 1. *The Moh proposal is protected from the SQL Tautology Attack.*

Proof. According to Section 2, tautology attacks a login page where the injection code places the username and password fields. The Moh proposal method encrypts the username and password from the client before being entered into the database. Table 26 shows the query result as an example of a tautology attack on the Moh proposal.

Table 26 shows a proof that the Moh proposal is protected from SQL tautology attacks and shows even how hard an attacker will try to change the input combination. The proposal disables the 'or' effect by encrypting input text, which results in different text named cipher text.

Theorem 2. *If an attacker knows the username, then the attacker can inject an SQL commenting attack, but the Moh proposal is immune to such attacks.*

Proof. As mentioned in Section 2, to accomplish a commenting attack, an attacker must know the username in advance. An attacker will disable the password condition by commenting on the fields after the known username. Table 27 shows an example of a commenting attack if the known username is 'admin'.

Table 27 proves that Moh proposal is immune to SQL commenting attacks even if an attacker knows the username in advance. However, the proposal encrypts the input that will cancel the commenting '--' effect on the following fields.

Theorem 3. *Moh proposal prevents SQL Piggybacked attacks.*

Proof. An attacker injects the piggybacked malicious code as a second query, which can be made through the search page of any web application. The Moh proposal prevents such attacks because the proposal encrypts all client inputs and uses a searchable encryption method to search the encrypted database. Table 28 demonstrates an example of such an attack.

Table 28 is a proof of preventing a piggybacked attack where the keyword is encrypted rather than using plain text format.

Table 19
Created account (request-response size) (Bytes).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page (Request)	639	636	636	637	637	715
Create Account (Request)	701	695	695	701	697	758
Call web page (Response)	279	279	279	279	279	330
Create Account (Response)	464	536	590	587	639	517

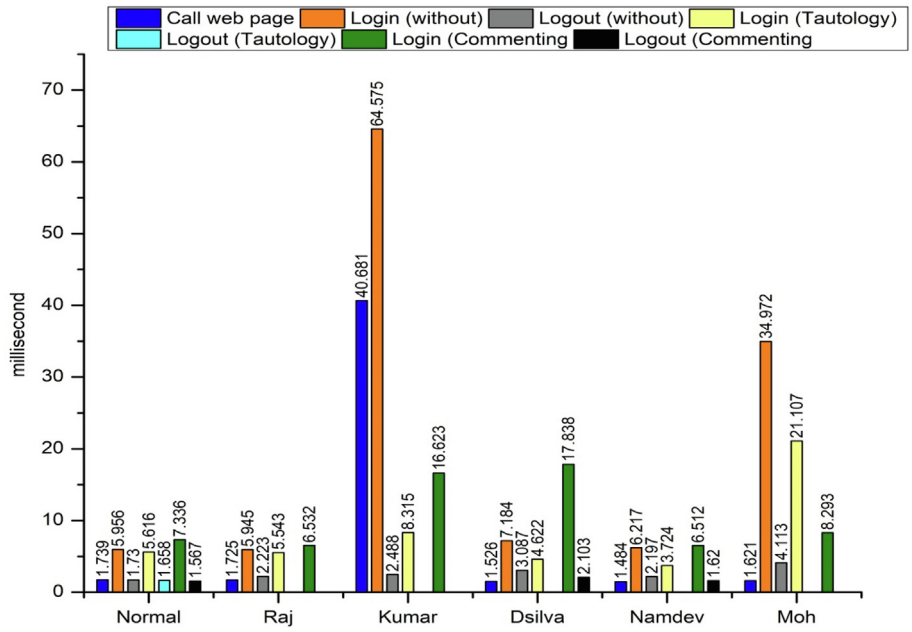

Fig. 12. RTT login–logout.

Table 20
RTT login-logout (milliseconds).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	1.739	1.725	40.681	1.526	1.484	1.621
Login (without)	5.956	5.945	64.575	7.184	6.217	34.972
Logout (without)	1.73	2.223	2.488	3.087	2.197	4.113
Login (Tautology)	5.616	5.543	8.315	4.622	3.724	21.107
Logout (Tautology)	1.658	0	0	0	0	0
Login (Commenting)	7.336	6.532	16.623	17.838	6.512	8.293
Logout (Commenting)	1.567	0	0	2.103	1.62	0

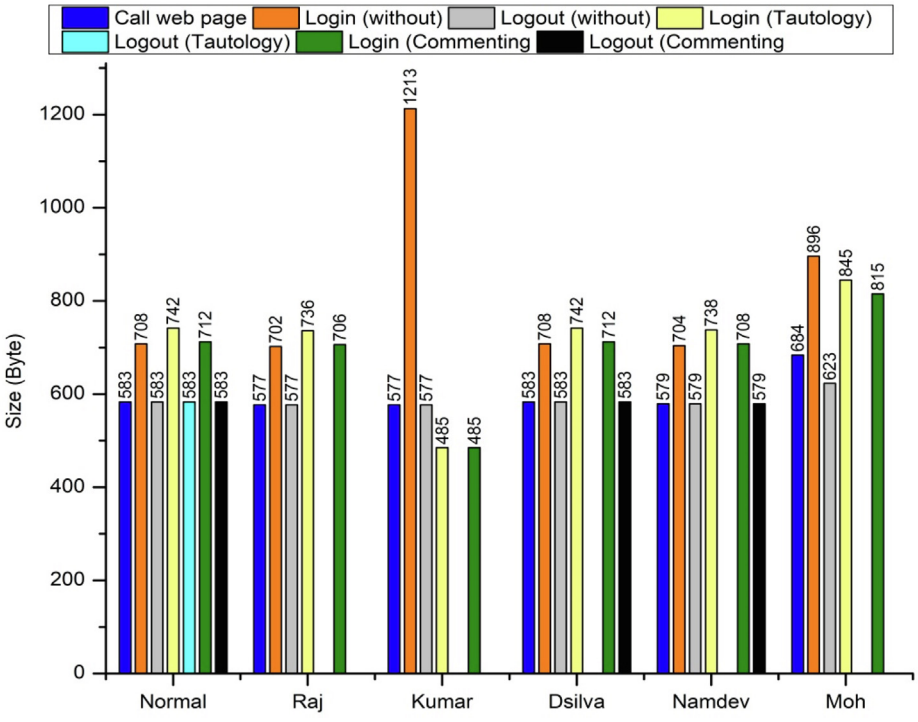

Fig. 13. Request size login–logout.

Table 21
Request size login-logout (Bytes).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	583	577	577	583	579	684
Login (without)	708	702	1213	708	704	896
Logout (without)	583	577	577	583	579	623
Login (Tautology)	742	736	485	742	738	845
Logout (Tautology)	583	0	0	0	0	0
Login (Commenting)	712	706	485	712	708	815
Logout (Commenting)	583	0	0	583	579	0

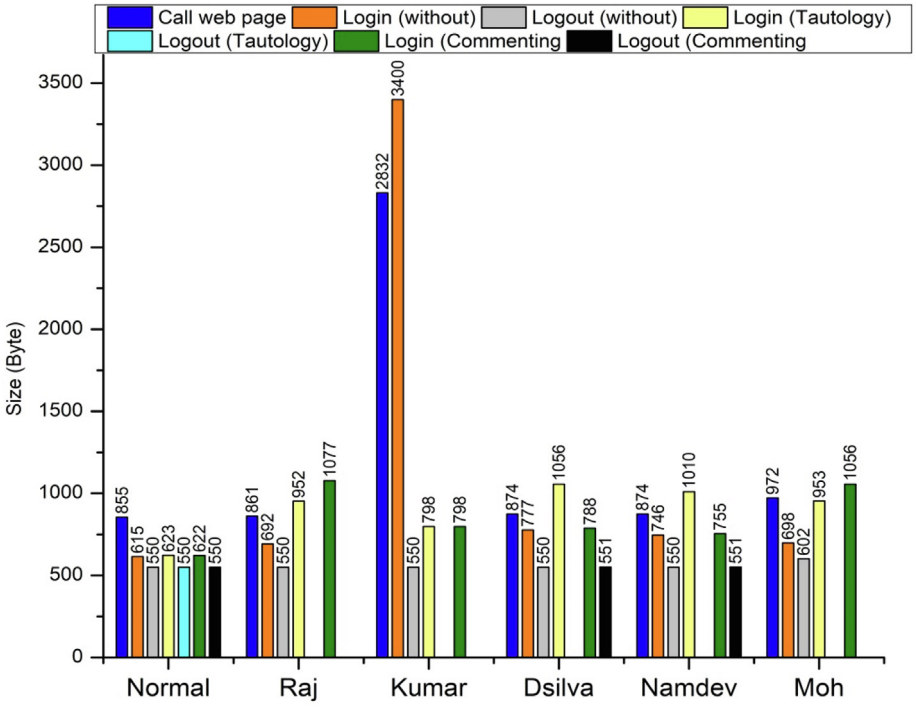

Fig. 14. Response size login–logout.

Table 22
Response size login-logout (Bytes).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	855	861	2832	874	874	972
Login (without)	615	692	3400	777	746	698
Logout (without)	550	550	550	550	550	602
Login (Tautology)	623	952	798	1056	1010	953
Logout (Tautology)	550	0	0	0	0	0
Login (Commenting)	622	1077	798	788	755	1056
Logout (Commenting)	550	0	0	551	551	0

Table 23
RTT search (milliseconds).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	0.784	0.762	0.793	0.741	0.63	0.763
Search (without-Attack)	12.624	18.201	9.428	8.671	20.253	33.993
Search (Piggybacked)	10.388	17.29	8.387	9.381	7.222	0

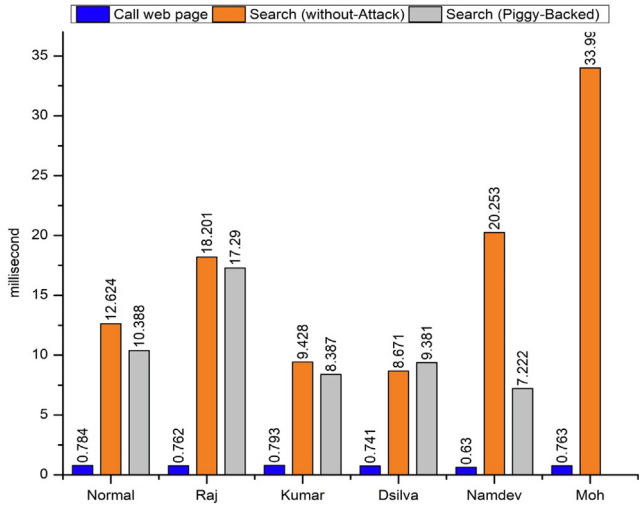

Fig. 15. RTT search.

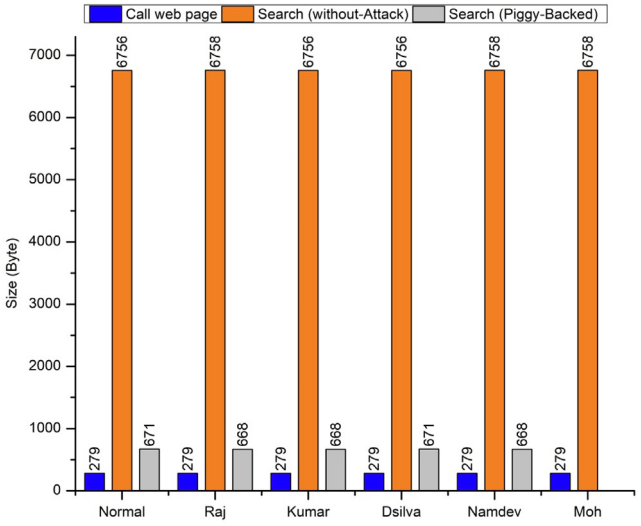

Fig. 17. Response size search.

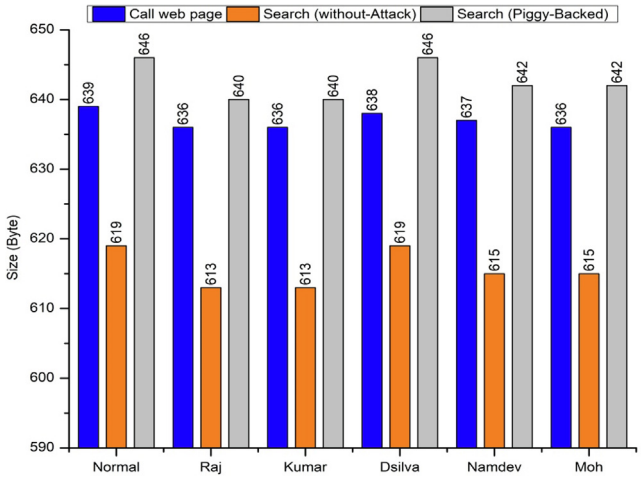

Fig. 16. Request size search.

Theorem 4. *The Moh proposal is protected against SQL piggybacked attacks even if the attacker knows the encryption key.*

Proof. Knowing the secret key will enable an attacker to decrypt the books table only if the attacker gains access to the database. The searchable encryption technique will limit the search to encrypted keywords. A malicious code from an attacker does not exist in the keywords.

Theorem 5. *If an attacker knows the encryption key, then the attacker does not threaten users' table privacy.*

Proof. The encryption key is related to the books table or other database table information, whereas the user table is encrypted using a different key. Each user in the user table is encrypted using the hash digits from the password as an encryption key. Each user uses a different key, which proves that disclosing the secret key does not threaten the user table.

Theorem 6. *The Moh proposal is protected from the SQL Union Attack.*

Proof. Table 29 shows the query result as an example of a Union attack on the Moh proposal, which shows proof that encrypting input text will disable the attack purpose or target.

Theorem 7. *The Moh proposal is protected from the SQL alternate encoding attack.*

Proof. Table 30 shows the query result as an example of an alternate encoding attack on the Moh proposal. This is proof that encrypting input text will disable the attack purpose or target.

Theorem 8. *The Moh proposal is protected from the SQL Stored Procedure Attack.*

Proof. Table 31 shows the query result as an example of a Stored Procedure attack on the Moh proposal. This is proof that encrypting input text will disable the attack purpose or target.

8. Conclusion and future works

An SQL injection attack is a web application safety threat whose technique is based on injecting malicious code into user input fields. An attacker's objective is to target database information that can be affected according to the injection type. According to SQL injection types, an attacker may gain unauthorised access or manipulate database information.

The proposed method encrypts all user input before contact with the database and uses searchable encryption on the basis of encrypted keywords. Symmetric encryption is applied, which

Table 24
Request size search (Bytes).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	639	636	636	638	637	636
Search (without-Attack)	619	613	613	619	615	615
Search (Piggybacked)	646	640	640	646	642	642

Table 25
Response size search (Bytes).

	Normal	Raj	Kumar	Dsilva	Namdev	Moh
Call web page	279	279	279	279	279	279
Search (without-Attack)	6756	6758	6756	6756	6758	6758
Search (Piggybacked)	671	668	668	671	668	0

Table 26
Example of tautology attack on Moh proposal.

Attacker input
Username = a' or 'a'='a Password = a' or 'a'='a *Web server generate query* $pw = hash('sha3-512', 'a' or 'a'='a'); SELECT * FROM users WHERE username = AES_ENCRYPT('a' or 'a'='a','$pw') SELECT * FROM users WHERE username = AES_ENCRYPT('a' or 'a'='a',' 5e6b1a402c695983e878fb17ab80abb096a07f5b50491cc9a3eb8295eafd91984 580cf2baf15f8bca180886e10af17f5d55478129631251e3fcda46ab26322ca') SELECT * FROM users WHERE username= ***(***4641d08ce2cda412c1b8bc6e60ff3479***)HEX*** *Database result* There is no match for the above username which display to the attacker 'Could not log you in' message

Table 27
Example of commenting attack on Moh proposal.

Attacker input
Username = admin' – Password = xx *Web server generate query* $pw = hash('sha3-512', 'xx'); SELECT * FROM users WHERE username = AES_ENCRYPT('admin' – ','$pw'); SELECT * FROM users WHERE username = AES_ENCRYPT('admin' – ',' ab585b96d532a05ebbff8653c430099964421248ac320dde44cab79388a78d1d2 230630e682cbe9e6ee3e2252cabbc8524fcb3b8f5a8efe42de278e2d33da3b0'); SELECT * FROM users WHERE username= (0x4baa698de885afab73552073db7bb408)$_{HEX}$ *Database result* There is no match for the above username which display to the attacker 'Could not log you in' message

Table 28
Example of piggybacked attack on Moh proposal.

Attacker input
search field (any keyword) London'; drop table users; *Web server generate query* $key = hash('sha3-512', 'subber'); SELECT * FROM books WHERE keyword = AES_ENCRYPT('London'; drop table table1;','$key'); SELECT * FROM books WHERE keyword= (0x8bce6926813c7d985eb978d44d731bea1352f26465785a04...)$_{HEX}$ *Database result* The result keyword does not match any encrypted keyword, there are no result and the table 1 not drop

Table 29
Example of union attack on Moh proposal.

Attacker input
Username = admin Password = moh' UNION ALL SELECT * FROM users; *Web server generate query* $pw = hash('sha3-512', moh' UNION ALL SELECT * FROM users;); SELECT * FROM users WHERE username = AES_ENCRYPT('admin','$pw') SELECT * FROM users WHERE username = AES_ENCRYPT('admin',' 0c94d2d819b1c0deba625dc37133ec3e8d547baa8fe911e6188e88c7dab163876 4e447b8883b7f72f316ec176806a3f7b1610dc0687fa64e39a81024dc08aeac') SELECT * FROM users WHERE username= ***(***d4a8242e822289b27dc1003f2b627ce9***)HEX*** *Database result* There is no match for the above username which display to the attacker 'Could not log you in' message

Table 30
Example of an alternate encoding attack on the Moh proposal.

Attacker input
search field (id) London'; exec(char(0x73687574646f776e));
Web server generate query
SELECT * FROM books WHERE keyword = AES_ENCRYPT('London'; exec(char(0x73687574646f776e));','$key');
SELECT * FROM books WHERE keyword= $(69eaa388e5a8a9392e5b893c97ee4198ee12aad0d0c934c9...)_{HEX}$
Database result
The result keyword does not match any encrypted keyword, there are no result and procedure does not execute even if its encoded.

Table 31
Example of stored procedure attack on Moh proposal.

Attacker input
search field (id) London'; exec(SHUTDOWN);
Web server generate query
SELECT * FROM books WHERE keyword = AES_ENCRYPT('London'; exec(SHUTDOWN);','$key');
SELECT * FROM books WHERE keyword= $(bd79ef151b0beee216f4b2b4640044e5731b0d01aac0af6b...)_{HEX}$
Database result
The result keyword does not match any encrypted keyword, there are no results and procedure does not execute.

causes a slight delay in processing, as when creating an account. The RTT is 139 msec, while in the normal state it is 34 msec. This method prevents SQLIs, and the results are comparable with previous methods where the encryption process takes a long time but can withstand attacks. The process time delay is reasonable because any attack is prevented. The RTT metric is shown in Tables 21 and 23. The Moh proposal measurements are within the average except a login without attack is 35 msec, login with tautology is 21 msec, and search without attack is 34 msec. Kumar's proposal has the longest login without attack at 65 msec, and Namdev's proposal has the second longest search at 20 msec. The request and response size as expressed in Tables 21, 22, 24 and 25 are within the average value of other proposals and normal states except for Kumar's proposal, which requires greater size in the login request without attack at 1213 bytes with a response of 3400 bytes.

The proposal has no limitation for platform or language and is applicable to any web application environment such as e-banking. The results with security analysis prove that the proposal is practical and secure. The future work of study will involve applying the idea to cloud architecture and determining whether it can survive SQL injection assaults in various topologies.

Funding Statement

This work is supported by Natural Science Foundation of Top Talent of SZTU (Grant No. 20211061010016).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Rani, G. S. Sarika, and P. Rupa. "A Study of Prevention and Detection Analysis of SQL Injection Attack". AIP Conference Proceedings 2358, 2021.
[2] Sadalkar K, Mohandas R, Pais A. Model Based Hybrid Approach to Prevent SQL Injection Attacks in PHP. Lecture Notes in Computer Science, vol 7011. Berlin, Heidelberg: Springer; 2011. p. 3–15.
[3] Z. Hlaing and M. Khaing, "A Detection and Prevention Technique on SQL Injection Attacks". In Proceedings of the IEEE Conference on Computer Applications(ICCA), IEEE, Myanmar, 2020.
[4] K. Mohanram and T. Mirnalinee, "Secured Data Storage and Retrieval Techniques for Effective Handling of Transport Data," in Proceedings of the 2nd International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM), pp. 239-243, India, IEEE, 2017.
[5] N. Johnson, J. Near, and D. Song, "Towards Practical Differential Privacy for SQL Queries" in Proceedings of the VLDB Endowment, Vol. 11, No. 5, pp. 526-539, ACM, 2018.
[6] B. Nagpal, N. Chauhan, and N. Singh, "SECSIX: security engine for CSRF, SQL injection and XSS attacks," International Journal of System Assurance Engineering and Management, vol. 8, no. 2, pp. 631-644, Springer, Nov. 2017.
[7] Z. Marashdeh, K. Suwais and M. Alia. "A Survey on SQL Injection Attack: Detection and Challenges". In Proceedings of the International Conference on Information Technology (ICIT), IEEE, Jordan, 2021.
[8] OWASP. Category: OWASP top ten project, https://owasp.org/www-project-top-ten/ (accessed: 2020-02-20).
[9] C. Ping, "A second-order SQL injection detection method," in Proceedings of IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), pp. 1792-1796, Chengdu, IEEE, 2017.
[10] A. Maraj, E. Rogova, G. Jakupi, and X. Grajqevci, "Testing techniques and analysis of SQL injection attacks," in Proceedings of the 2nd International Conference on Knowledge Engineering and Applications (ICKEA), pp. 55-59, London, IEEE, 2017.
[11] A. Patil, A. Laturkar, S. Athawale, R. Takale, and P. Tathawade, "A multilevel system to mitigate DDOS, brute force and SQL injection attack for cloud security," in Proceedings of the International Conference on Information, Communication, Instrumentation and Control (ICICIC), pp. 1-7, Indore, IEEE, 2017.
[12] B. Thombare and R. Soni. "Prevention of SQL Injection Attack by Using Black Box Testing". In Proceedings of the 23rd International Conference on Distributed Computing and Networking. 2022.
[13] F. Joe and V. Selvarajah. "A Study of SQL Injection Hacking Techniques". In Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC). 2021.
[14] A. Ramesh. A. Bhowmick, and A. Lal, "An Authentication Mechanism to Prevent SQL Injection by Syntactic Analysis", in Proceedings of the International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15),pp. 1-6, India, IEEE, 2015.
[15] N. Singh, M. Dayal, R. Raw, and S. Kumar, "SQL Injection: Types, Methodology, Attack Queries and Prevention," in Proceedings of the 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2872-2876, India, IEEE, 2016.
[16] A. Marashdih and Z. Zaaba, "Cross Site Scripting Removing Approaches in Web Application," in Proceedings of the 4th Information Systems International Conference (ISICO), pp. 647-655, Indonesia, Elsevier, 2017.
[17] S. Raj and E. Sherly, " An SQL Injection Defensive Mechanism Using Reverse Insertion Technique," Smart and Innovative Trends in Next Generation Computing Technologies (NGCT), Communications in Computer and Information Science, vol 828, pp. 335-346, Springer, Singapore, 2017.
[18] Namdev M, Hasan F, Shrivastav G. A Novel Approach for SQL Injection Prevention Using Hashing & Encryption (SQL-ENCP). Int J Comput Sci Informat Technol (IJCSIT) 2012;3(5):4981–7.
[19] S. Avireddy, V. Perumal, N. Gowraj, R. Kannan, P. Thinakaran, S. Ganapathi, J. Gunasekaran, and S. Prabhu, "Random4: An Application Specific Randomized Encryption Algorithm to Prevent SQL Injection," in Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1327-1333, IEEE, Liverpool, 2012.
[20] Y. Swarup, A. Kumar, A. Tyagi, and V. Kumar. "Prevention of SQL Injection Attacks using Query Hashing Technique". In Proceedings of the 2nd

International Conference on Range Technology (ICORT), IEEE, Balasore, India, 2021.
[21] N. Palsetia, G. Deepa, F. Khan, P. Thilagam, and A. Pais, "Securing Native XML Database-Driven Web Applications from XQuery Injection Vulnerabilities," Journal of Systems & Software, vol. 122, pp. 93-109, Elsevier, 2016.
[22] M. Hussain, S. Abbdal refish, M. Khalefa, S. Hussain, Z. Hussien, Z. Abduljabbar, and M. Al sibahee. "Web application database protection from SQLIA using permutation encoding". In Proceedings of the 4th International Conference on Information Science and Systems. ACM. Edinburgh Napier University, UK. 2021.
[23] J. Blömer and N. Löken, " Cloud Architectures for Searchable Encryption, " in Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES), pp. 25:1-25:10, Hamburg, Germany, ACM, 2018.
[24] N. Pramanick and S. Ali, "A comparative survey of searchable encryption schemes," in Proceedings of the 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1-5, IEEE, Delhi, 2017.
[25] S. Mitra, S. Roy, M. Sarkar, S. Bhowmik, and S. Maur. "Prevention of SQL Injection and Security Enhancement in Cyber Networks". In Proceedings of the 5th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), IEEE, Kolkata, India, 2021.
[26] A. Ghafarian, "A hybrid method for detection and prevention of SQL injection attacks," in Proceedings of the Computing Conference (SAI), pp. 833-838, IEEE, London, 2017.
[27] K. D'silva, J. Vanajakshi, K. Manjunath, and S. Prabhu, "An effective method for preventing SQL injection attack and session hijacking," in Proceedings of the 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), pp. 697-701, IEEE, Bangalore, 2017.
[28] Anjugam S, Murugan A. Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization. Int J Adv Res Comput Sci Softw Eng 2014;4(4):173–7.
[29] D. Kumar and M. Chatterjee, " MAC based solution for SQL injection ," Journal of Computer Virology and Hacking Techniques, vol. 11, no. 1, pp. 1-7, Springer, 2015.
[30] Balasundaram I, Ramaraj E. An Authentication Mechanism to prevent SQL Injection Attacks. Int J Comput Appl 2011;19(1):30–3.
[31] Zhu Y, Zhang G, Lai Z, Niu B, Shen Y. A Two-Tiered Defence of Techniques to Prevent SQL Injection Attacks. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Advances in Intelligent Systems and Computing, vol 612. Cham: Springer; 2017. p. 286–95.
[32] A. Dalai and S. Jena, " Neutralizing SQL Injection Attack Using Server Side Code Modification in Web Applications, " Security and Communication Networks, vol. 2017, pp. 12, Hindawi,2017.
[33] Z. Xiao, Z. Zhou, W. Yang and C. Deng, "An approach for SQL injection detection based on behavior and response analysis," in Proceedings of the 9th International Conference on Communication Software and Networks (ICCSN), pp. 1437-1442, IEEE, Guangzhou, 2017.
[34] Selvamani K, Kannan A. A Novel Approach for Prevention of SQL Injection Attacks Using Cryptography and Access Control Policies,". Advances in Power Electronics and Instrumentation Engineering PEIE, Communication in Computer and Information Science, vol 148. Berlin, Heidelberg: Springer; 2011. p. 26–33.
[35] D. Kundi, A. Khalid, A. Aziz, C. Wang, M. O'Neill and W. Liu, "Resource-Shared Crypto-Coprocessor of AES Enc/Dec With SHA-3," in IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 1-14, 2020.
[36] W.Yu, C.Jianhua, and H. Debiao. "A new collision attack on MD5". In Proceedings of the International Conference on Networks Security, Wireless Communications and Trusted Computing. 2009.
[37] T. Zhou, Y. Zhu, N. Jing, T. Nan, W. Li, and B. Peng. "Reliable SoC Design and Implementation of SHA-3-HMAC Algorithm with Attack Protection". In Proceedings of the IEEE International Conference on Smart Cloud (SmartCloud). 2020.
[38] T. Singha, R. Palathinkal and S. Ahamed, "Implementation of AES Using Composite Field Arithmetic for IoT Applications," 2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP), India, pp. 115-121, 2020.
[39] Wang H, Xia Z, Fei J, Xiao F. An AES-Based Secure Image Retrieval Scheme Using Random Mapping and BOW in Cloud Computing. IEEE Access 2020;8:61138–47.
[40] M. Pranav and A. Rajab. "DES security Enhancement with Dynamic Permutation". In Proceedings of the International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). 2015.
[41] M. Hussain, Z. Hussien, Z. Abduljabbar, S. Hussain, and M. Al Sibahee. "Boost Secure Sockets Layer against Man-in-the-Middle Sniffing Attack via SCPK." In Proceedings of the 2018 International Conference on Advanced Science and Engineering (ICOASE), IEEE, Kurdistan Region, Iraq. 2018.
[42] XAMPP, https://www.apachefriends.org/download.html (accessed: 2020-02-20).
[43] Dataset from Gutenberg, http://www.gutenberg.org/ebooks/offline_catalogs.html (accessed: 2020-02-20).
[44] Wireshark, https://www.wireshark.org (accessed: 2020-02-20).