

12-2018

# SQL Injection Prevention Technique Using Cryptography

Jayakrishnan S. Karunanithi  
jskarunanithi@stcloudstate.edu

Follow this and additional works at: [https://repository.stcloudstate.edu/msia\\_etds](https://repository.stcloudstate.edu/msia_etds)

---

## Recommended Citation

Karunanithi, Jayakrishnan S., "SQL Injection Prevention Technique Using Cryptography" (2018). *Culminating Projects in Information Assurance*. 74.  
[https://repository.stcloudstate.edu/msia\\_etds/74](https://repository.stcloudstate.edu/msia_etds/74)

This Starred Paper is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact [rswexelbaum@stcloudstate.edu](mailto:rswexelbaum@stcloudstate.edu).

# **SQL Injection Prevention Technique Using Cryptography**

by

Jayakrishnan S. Karunanithi

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in

Information Assurance

December, 2018

Starred Paper Committee:  
Susantha Herath, Chairperson  
Lynn Collen  
Balasubramanian Kasi

### **Abstract**

In our day-to-day life, web applications play an important role such as shopping, making financial transactions, social networking, etc. Most of the business prefer online services instead of in-person services because it is easier for both customers and organizations. Making a web application available to everyone makes it more vulnerable. One of those vulnerabilities is SQL (Structured Query Language) injection. SQL injection is a technique where attackers inject malicious code through user inputs or URLs and gain access to the database. Through this attack, hackers can destroy or change the data present in the database. This paper focuses on how to prevent the SQL injection attacks using five cryptographic algorithms (AES, Triple DES, RSA, Blowfish, and Twofish). Finally, the research evaluates which cryptographic algorithm is most appropriate to prevent SQLIA in web applications.

Keywords: Web applications, Cryptography, and SQL injection.

## Table of Contents

	Page
List of Tables .....	5
List of Figures .....	6
Chapter	
I. Introduction .....	9
Introduction .....	9
Problem Statement .....	10
Nature and Significance of the Problem .....	10
Objective of the Study .....	11
Research Questions/Hypotheses .....	11
Limitations of the Study .....	12
Definition of Terms .....	12
Summary .....	13
II. Background and Review of Literature .....	14
Introduction .....	14
Background Related to the Problem .....	14
Literature Related to the Problem .....	14
Literature Related to the Methodology .....	16
Summary .....	16
III. Methodology .....	17
Introduction .....	17

Chapter	Page
Design of the Study .....	17
Data Collection .....	43
Data Analysis .....	70
Summary .....	71
IV. Data Presentation and Analysis .....	73
Introduction .....	73
Data Presentation .....	73
Data Analysis .....	79
Summary .....	80
V. Results, Conclusion, and Recommendations .....	81
Introduction .....	81
Results .....	81
Study Questions .....	82
Conclusion .....	83
Future Work .....	83
References .....	86

### List of Tables

Table	Page
1. AES-Key Size with the Number of Rounds .....	28
2. Data Stored after Encryption .....	43
3. SQLIA vs. Cryptographic Algorithms .....	44
4. SQLIA vs. Cryptographic Algorithms .....	45
5. Application Response–Encryption only Password Field .....	74
6. Application Response–Encrypting both Username and Password .....	76
7. Time Taken by each Algorithm for Encryption and Decryption .....	77

## List of Figures

Figure	Page
1. Properties of database security .....	9
2. B2C E-commerce sales growth .....	10
3. Classification of SQL injection .....	17
4. SQL code to authenticate the user .....	18
5. Login screen .....	19
6. Bypassed the authentication .....	19
7. Various user inputs .....	20
8. Attempting the Union query attack .....	21
9. Piggy-backed query attack .....	22
10. Stored procedure to get the employee address .....	23
11. Timing attack structure .....	24
12. Alternate encoding attack .....	25
13. Encryption .....	26
14. Symmetric encryption .....	26
15. Asymmetric encryption .....	27
16. Hashing .....	28
17. AES–block diagram .....	29
18. AES add round key .....	30
19. AES substitution bytes .....	30
20. AES–Shift rows .....	31

Figure	Page
21. AES–mix column .....	31
22. Key expansion .....	32
23. DES block diagram .....	33
24. DES-expansion function .....	34
25. Triple DES .....	35
26. RSA process .....	37
27. Blowfish process .....	38
28. Blowfish function .....	39
29. Twofish process .....	41
30. Using algorithms in PHP .....	46
31. Database with encrypted password .....	47
32. Tautology attack .....	48
33. Successful tautology attack .....	48
34. Illegal/logical incorrect queries attack .....	49
35. Successful illegal/logical incorrect queries attack .....	50
36. Successful union query attack .....	51
37. Piggy-backed queries attack .....	52
38. Failure piggy-backed queries .....	53
39. BLIND injection attack .....	55
40. Successful BLIND injection attack .....	56
41. Timing attack .....	57

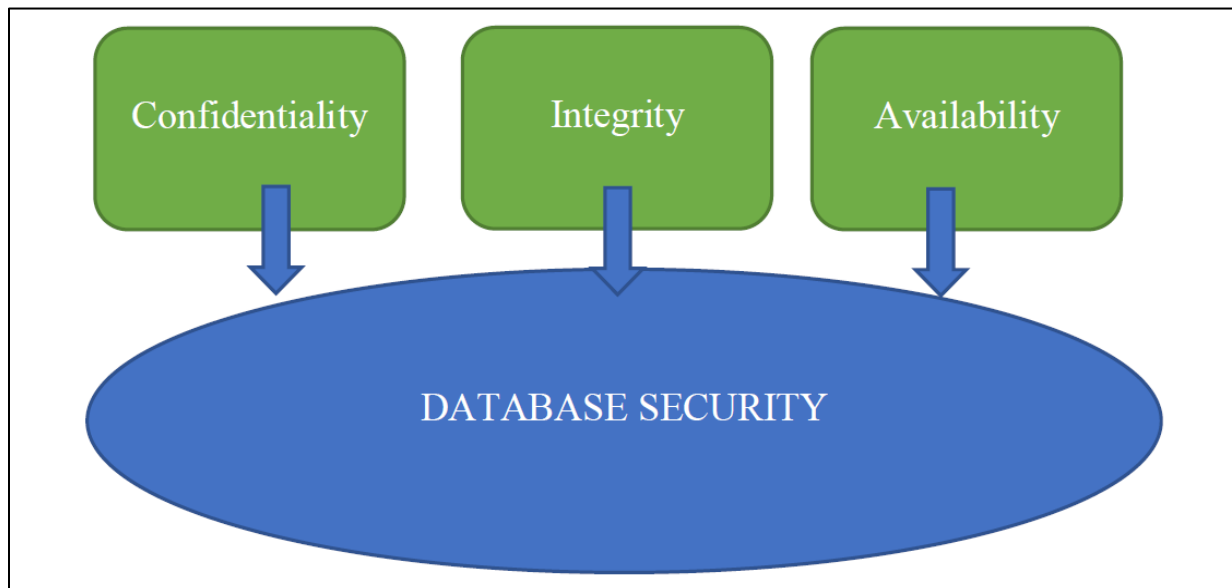


Figure	Page
42. Successful timing attack .....	58
43. Alternative encoding attack .....	59
44. Failure of alternate encoding attack .....	59
45. Database–encrypted both username and password .....	60
46. Tautology attack after encrypting both the fields .....	61
47. Failed tautology attack after encrypting both the fields .....	61
48. Successful illegal/logical incorrect queries attach–encrypting all fields .....	63
49. Timing attack after encrypting both input fields .....	67
50. Unsuccessful timing attack after encrypting both input fields .....	68
51. Speed test web page .....	70
52. Configuration .....	71
53. Graphical representation: Algorithms vs. time .....	78
54. Moving all SQL errors into a table .....	84
55. Moving all error messages and warnings from PHP into a file .....	85

## Chapter I: Introduction

### Introduction

SQL injection has been a serious problem faced by all the websites. As per web application attack statistics, about 24.9% of attacks were SQL injection ("Web Application Attack Statistics", n.d., 2017). Cryptography is one of the dominant techniques to prevent SQL injection attacks. All the confidential data are encrypted and stored in the database; even if the hacker gains access to the database, he/she cannot be able to decrypt the data without the knowledge of algorithm and key used to encrypt the data. Providing security to the database is the essential part of running an online business. Database security requires permitting or prohibiting user action on the database and the objects on it (Deepika, 2015, p. 621).



*Figure 1:* Properties of database security (Deepika, 2015).

The above figure is the representation of the properties (Confidentiality, integrity, and availability) in the database security.

## Problem Statement

Several methods have been proposed to stop SQLIA. Perhaps the most significant of these is ‘Cryptographic Technology.’ In this research, various cryptographic algorithms are compared, and the most appropriate algorithm for the web application is determined.

SQLIA can be prevented using many techniques, but cryptography might be the best and easiest method to prevent web applications. Using cryptography will reduce the effort involved in validating and filtering the user inputs. Encoding the user inputs makes all malicious codes into meaningless data. This method might help to prevent the web application from SQLIA.

## Nature and Significance of the Problem

B2C e-commerce makes both the businesses and consumers more comfortable, as they can sell or buy any products through a website. The availability of these websites makes it more vulnerable. SQL injection is one of the most dangerous attacks on web applications. This paper focuses on how to prevent web applications from SQLIA using a cryptography technique.

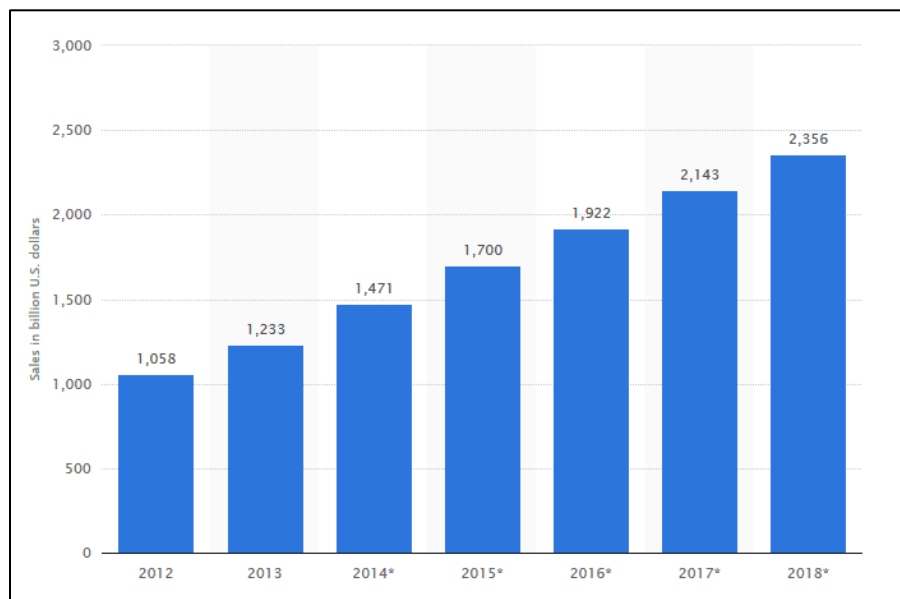


Figure 2: B2C E-commerce sales growth ("E-commerce statistics", n.d.).

The above figure shows the rapid increase in business to consumer E-commerce. It was expected to have a sale of \$2,356 billion in 2018 through B2C e-commerce.

### **Objective of the Study**

The primary aim of this study to implement various cryptographic algorithms in a web application and observe how they react to SQLIA. The study is conducted by encrypting only the password field and by encrypting all the user input fields. Implementing the cryptographic algorithms can slow down the response speed of the application. A speed test conducted on the cryptographic algorithms taken in this study to come out with the best algorithm for the web application.

SQLIA can be conducted not only through the user inputs but also through URL. This study focuses on the SQLIA conducted through the user input fields. SQLIA is becoming more vulnerable because this can exploit the database which contains all confidential information of the customers. A successful SQLIA can cause a massive loss to the business.

### **Research Questions/Hypotheses**

Cryptographic techniques may prevent most of the primary SQLIA on web applications. The response speed of the web application may reduce due to the implementation of the cryptographic algorithms in web applications. Hashing is also used in the web application to save the passwords. The study questions are listed below:

- 1) What is SQLIA?
- 2) What is the primary classification of SQLIA?
- 3) Why is web application security an important issue?
- 4) What background information necessary to understand web application security?

- 5) What are the effects of SQLIA?
- 6) What is being currently done to secure web application?
- 7) Based on the research, what are the thoughts about web application security from SQLIA and why?
- 8) Based on the research, what should be done to secure web application from SQLIA?

### **Limitations of the Study**

This study has five cryptographic algorithms with various strengths. Also, this study focused only on seven basic types of SQLIA, which are mostly executed by the attackers to gain knowledge about the database. The web application can become vulnerable because of many kinds of attacks. SQL injection is the most powerful among them, so this study focusses only on SQLIA.

One of the most used algorithms was ‘Blowfish’ because of its simplest structure. AES’ is used for its design and stability ( “*Blowfish*”, n.d.). ‘Triple DES’ was considered because it is the only algorithm which has three encryptions and decryptions. ‘Twofish’ has a larger ciphertext size while ‘RSA’ is the best asymmetric algorithm with maximum key size.

### **Definition of Terms**

Cryptography is the conversion of plaintext into ciphertext or vice versa. A cryptographic algorithm has three sections: encryption, decryption and key generation. The data obtained from the user is encrypted. Also, key generation will process the key and provide it to the encryption process to get the ciphertext. Similarly, during the decryption process, the prepared key is used to get the plain text.

Injection of malicious code through user inputs or on any other fields to make changes to the application is called a SQL injection attack.

**Summary**

This chapter discussed the importance of web applications and how the SQL injection is attacking them. It is more curious to know whether cryptography technique prevents all the SQLIA. The next chapter explains about the various researchers that happened on database security, encryption techniques and different methods used to avoid applications from attackers.

## **Chapter II: Background and Review of Literature**

### **Introduction**

This chapter discusses the various articles related to the database security using the cryptographic technique. Those articles are more helpful to make some crucial decisions in researching database security. The background information related to the problem and the literature review are explained.

### **Background Related to the Problem**

SQLIA have various classifications. Attackers can make use of any of those to intrude into the application. Similarly, the cryptographic technique consists of multiple algorithms. Each algorithm has its advantages and disadvantages. Sorting the most significant algorithms is the essential part of researching data security using encryption.

Web applications are overgrowing. As per a survey, \$2,356 billion would be the business amount from e-commerce web applications in 2018. Most of the businesses rely on the web application, exploiting them can cause a massive loss to the customers as well as to the company. SQLIA is the most common attack on the web application, around 28 percentage of the attacks on the web application are due to the SQLIA. The best tool is needed to prevent the SQLIA. This research is on the SQLIA on the database.

### **Literature Related to the Problem**

Mushtaq et al. (2017) compared the performance of DES, 3DES, AES, Blowfish, HiSea based on the memory, encryption & decryption time, and design. The author came up with the result saying that AES, Blowfish, and HiSea are the best algorithms. The author failed to consider asymmetric algorithms into the account.

Nadeem and Javed (n.d.) compared the speed of DES, 3DES, AES, and Blowfish and concluded that Blowfish is the fastest algorithm. The author added that an algorithm which is more complex and with more number of rounds is secure. Whenever the complexity of the algorithm increases, the performance of the application would be affected.

Mewada, Sharma, and Gautam (2016) conducted a performance analysis of the encryption algorithm. Panda (2016) recorded the encryption, decryption, and throughput time taken by AES, DES, Blowfish, and RSA with different file types such as Binary, text and image file. After comparing the results, the author said that AES has better performances than other algorithms (Mewada et al., 2016). The author also made a detailed analysis of symmetric key algorithms (DES, T-DES, Blowfish, IDEA, TEA, CAST, and AES) and declared that AES is the best algorithm by security, flexibility, performance, and memory utilization. AES is one of the most important algorithms, which has the best design.

Kumar, Prabhu, Durga, and Jeevitha (2016) surveyed the various asymmetric algorithm (RSA, Diffie-Hellman, DSA, OAEP, and Elgamal) and said that RSA is providing much overhead in encryption. Gaithuru, Bakhitari, Salleh, and Muteb (2015) conveyed that, RSA continues to have high implementation because of its longer key size, which makes it more secure. While implementing various algorithms to examine the performances, RSA comes under the asymmetric key algorithms. Mushtaque, Dhiman, Hussain, & Maheshwari (2014) compared the size of the ciphertext generated by various algorithms. The author ended up saying that Twofish and Blowfish have a larger ciphertext comparing with other algorithms, which makes them more secure. Ciphertext should be more complicated so that attackers will not get any idea on how to decrypt them.



## **Literature Related to the Methodology**

Basharat, Azam and Muzaffer (2012) said that encryption could provide confidentiality but no assurance of integrity unless the hash or digital signature is used. The author also states that encryption also reduces the performance of web applications. It is always acceptable with the reduction in response speed of the web-application when it becomes more secure.

Rani, Kumar, Rao, Jagadish, and Pradeep (2012) used a methodology in which he uses the first three letters of username and password in a random order to generate a key (6). This key is used to encrypt using the AES algorithm. If the key is known, then there are more chances that attacker can attempt a brute force attack to crack the database.

Sood and Singh (2017) compared Blowfish and AES along with MD5 hash functions to secure the database. According to their results, AES + MD5 gives more security than Blowfish. S, Kumar and Rahman (2015) used a methodology in which he does password verification using MD5 hashing and AES algorithm to encrypt all the data in the database. Whenever the encryption combined with the hashing algorithm, the security doubled. Singh and Kaur compared various papers on database security through encryption. Each algorithm has its advantage and disadvantages (Singh & Kaur, n.d.). More research is needed to find a perfect solution.

## **Summary**

This chapter helped to understand the various researchers conducted on the database security using encryption techniques. It's time to move to the methodology which can be used to prevent SQLIA.

## Chapter III: Methodology

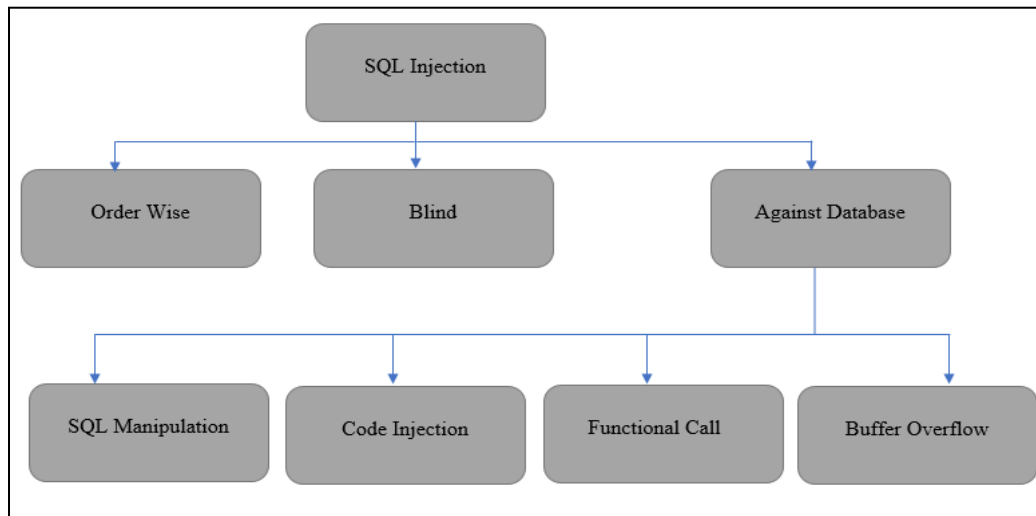
### Introduction

This chapter describes the SQLIA and cryptographic techniques used in this study. Creating a real-time web application and conducting the experiments on them might provide a better outcome. The chapter also covers the design of the study and the data collection process.

### Design of the Study

Design involves the following steps:

1. Create a web application.
2. Conduct all the major SQLIA on the web application.
3. Implement the cryptographic algorithms on the web application.
4. Conduct SQLIA after implementation.
5. Record the response of the web application.



*Figure 3: Classification of SQL injection.*

The above figure shows the primary classification of SQLIA. SQL manipulation, code injection, function call, and buffer overflow come under the attacks against the database.

**Tautologies.** In a Tautology attack, hackers provide malicious code through the user input and bypass the authentication or extract data from the database. The attacker modifies the SQL statement such a way that WHERE clause is always TRUE, this results in avoiding the condition in the SQL statement (Sadeghia, Zamani, & Abdullah, 2013, p. 270). The standard method used to bypass 'WHERE' condition is by providing 'OR' '1 = 1', which inserts another condition by providing 'OR' followed by '1=1' which is always TRUE, so the result of the whole SQL statement becomes TRUE (Sadeghia et al., 2013). Attackers also provide "--" inline comment in user inputs, which ignores the rest of the SQL statement (Mukherjee, Bora, Sen, & Pradhan, 2015).

```

<?php
    if (isset($_POST['login']))
    {
        //echo "<script type='text/javascript'> alert('Success') </script>";
        $username = $_POST['username'];
        $password = $_POST['password'];

        $query = "select * from login where username = '$username' and password = '$password'";
        $query_run = mysqli_query($conn,$query);

        if(mysqli_num_rows($query_run) > 0)
        {
            $_SESSION['username'] = $username;
            header('location:home.php');
        }
        else
        {
            echo "<script type = 'text/javascript'> alert('Invalid Credentials') </script>";
        }
    }
?>

```

Figure 4: SQL code to authenticate the user.

The above figure shows the PHP script which has a SQL query to authenticate the user by validating the username and password. The values obtained from the user is pushed into the SQL query to find their records in the database.

If the attacker provides ' or '1'='1 in both the username and password field, he/she might be able to bypass the authentication. The actual SQL code becomes as provided below.

"select \* from login where username = "or '1'='1' and password = "or '1'='1'"; The query makes the SQL statement TRUE and allows the hacker to enter the website.

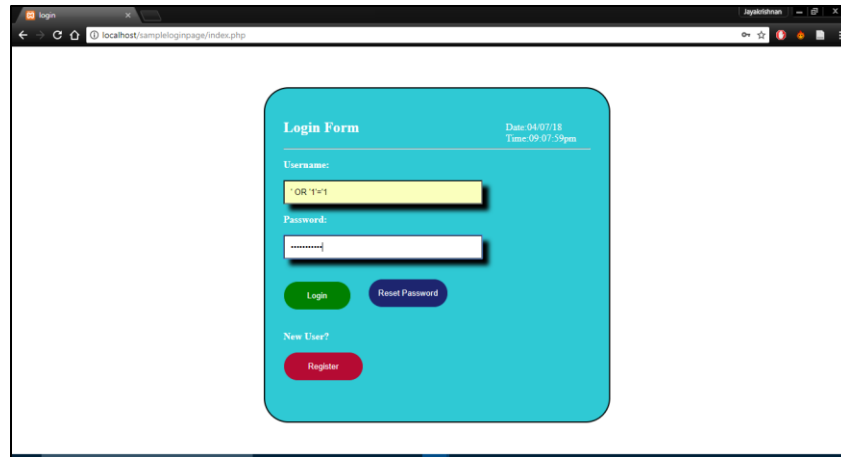


Figure 5: Login screen.

The above figure shows the login page of the web application which accepts username and password from the users. The user provided ' or '1'='1' in both the username and password fields.

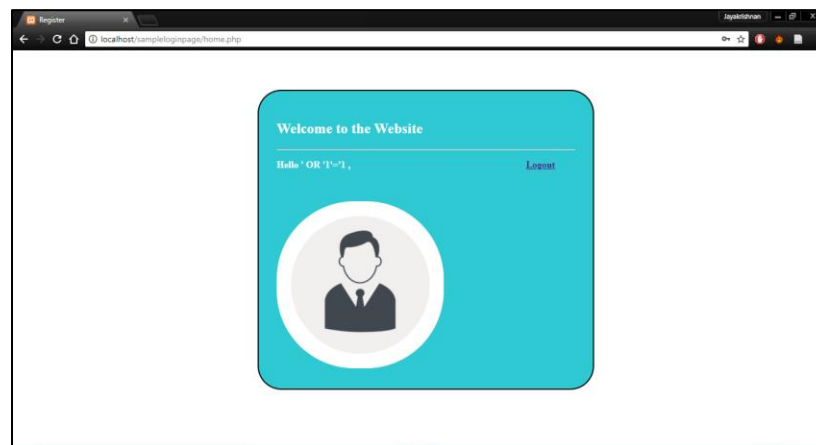


Figure 6: Bypassed the authentication.

The above figure shows that welcome page once the user logged into the application successfully. The images show that by providing ' or '1'='1' as username and password, the attack can successfully login into the first account in the database.

Similarly, the below user inputs are also able to bypass the authentication.

Figure 7: Various user inputs.

The above image shows the malicious code injected into the database. The user inputs provided in the above image will modify the SQL query as shown below.

"select \* from login where username = ' or 1=1-- -' and password = 'test'";

"select \* from login where username = ' or 1=1#' and password = ' or 1=1#'";

Since both '--' and '#' are comments in SQL language, they made the statement to ignore the password.

**Illegal/Logically incorrect queries.** This attack comes under the SQL manipulation attack in which the attacker injects the logically incorrect commands and makes use of the error generated by the database (Sadeghia et al., 2013). Conducting this attack multiple time can lead to the exploitation of the database structure and various other important information about the database. Below are the examples of logically incorrect query attacks:

"select \* from login where username = ' HAVING 1='1'; --' and password = 'test'";

"select \* from login where username = " GROUP BY User\_ID HAVING 1='1'; --" and password = 'test'";

Error: Column 'User\_ID.Login' is invalid in the select list because it is not either an aggregate function or group by clause (Kumar & Pateriya, 2012).

The above error messages help the attacker to identify the column name (User\_ID) and table name (Login). With these details, attackers can continue their attack and gather all the information available in the database.

**Union queries.** The Union query attack conducted on the web application by injecting the UNION keyword followed by a select query statement. The result provided by the database contains the result of actual and injected query (Jie Wang, Whitley, & Parish, n.d.). Once the attacker gains knowledge about the table and column names in the database using Illegal / Logically incorrect queries, he/she can attempt the Union query attack to get the details inside a table.

Figure 8: Attempting the Union query attack.

The above figure shows the user input used to conduct a Union query attack. The 'UNION SELECT \* FROM LOGIN --' and '12345' are used as the username and password respectively.

The user inputs provided in the above image will modify the SQL query as shown below:

"select \* from login where username = " UNION SELECT \* FROM LOGIN --' and password = '12345'";

This query could provide the details present in the Login table.

**Piggy-backed queries.** In Piggy-backed query attack, the original code was made to carry some malicious code with the help of the SQL statement separator ';' (Anitha, Lakshmi, Revathi, & Selvi, 2014). This attack allows the attacker to make any change to the database. They can INSERT, UPDATE, or DELETE any table or row into the database. To execute a successful Piggy-Backed query attack, the attacker should know the structure of the database.

Below are some of the examples of Piggy-Backed attacks:

The image shows a web-based login form with a light blue background. At the top, it says 'Login Form' in bold. To the right, it displays the date and time: 'Date:04/08/18' and 'Time:01:12:38am'. Below this, there are two input fields. The first is labeled 'Username:' and contains the text ' drop table login --'. The second is labeled 'Password:' and contains the text '12345'. At the bottom of the form, there are two buttons: a green 'Login' button and a blue 'Reset Password' button.

Figure 9: Piggy-backed query attack .

The above figure shows the user input used to conduct Piggy-backed query attack. The " drop table login --' and '12345' are used as the username and password respectively. The user inputs provided in the above image will modify the SQL query as shown below:

"select \* from login where username = 'drop table login --' and password = '12345'";

This query will lead the deletion of the login table.

**Stored procedure.** A set of SQL statements that are designed to do a specific task is a stored procedure. There are user-defined Stored Procedures and built-in Stored procedures, which allows it to interact with the operating system. There are different kinds of database types; without knowing the database type, it is tough to conduct a Stored Procedure injection attack. So, the attacker makes Illegal/Logically incorrect query attacks to see the database type. Once the attacker knows the database type used, they execute various built-in Stored Procedures by injecting malicious code ("SQL Injection", n.d.). Below is a sample of Stored Procedure injection attacks:

```
Alter procedure Employee_Address (@Employee_Name NVARCHAR(100))
AS
Begin
    Declare @result NVARCHAR(MAX);
    Set @ result = N'Select * from Emp_Address where Emp_Name = ''' + @Employee_Name + ''';
    EXECUTE (@result)
End
```

*Figure 10:* Stored procedure to get the employee address.

The above image shows the stored procedure used in the web application to fetch the employee address by obtaining the employee name. If the attacker provides "John'; SHUTDOWN; --" the SQL statement will become,

Select \* from Emp\_Address where Emp\_Name = 'John'; SHUTDOWN; -- ';

**Inference.** The purpose of this attack is to identify injectable parameters, identify schema, and extract data. The attacker injects some malicious code and waits for the response from the web application. Based on the answers, he/she can continue to attack and change the behavior of the application. Mostly the attacker injects true or false SQL statements. If the SQL



statement becomes true, there will be no change in application behavior else they behave differently ("SQL Injection", n.d.). There are two types of Inference attacks namely Blind injection and Timing attack.

**Blind injection.** In Blind injection attack, the attacker injects true or false SQL statements through user inputs or URLs. If the SQL statement becomes true, there will be no change in application behavior else they behave differently ("SQL Injection", n.d.).

Example of a blind injection attack (Mukherjee et al., 2015). The attacker can ask the web application whether the username of the database starts with the letter 'a'. If the answer is correct, the web-application shows the page or print out an error. By asking this question for 26 times, the attacker can get the first letter of the username of the database. He/she needs to continue this attack repeatedly to get the complete username of the database.

**Timing attack.** In timing attack, the attacker injects malicious code along with the time delay statement. If the time delay happened then the injected query was executed successfully (Jie Wang et al., n.d.).

```

IF
    <SQL injection statement>
Then
    <Time delay statement>

```

Figure 11: Timing attack structure.

The above figure shows the structure of the timing attack. For a successful SQL injection, there will be a time delay. The delay command is different for each database management system. Microsoft SQL server has 'WAITFOR,' while Oracle and MySQL have 'SLEEP' (Sadeghia et al., 2013).

**Alternate encoding.** An alternate encoding attack is used to bypass the scanning and detection technique used in web applications. Below is an example of this kind of attacks (Roy & Sairam, 2011).

*Figure 12:* Alternate encoding attack.

The above figure shows the user input used to conduct 'Alternate encoding attack'. The '`; exec(char(Ox73687574646j776e)) -- -`' and 'test' are used as the username and password respectively. The user inputs provided in the above image will modify the SQL query as shown below:

```
"select * from login where username = ' ; exec(char(Ox73687574646j776e)) -- -' and
password = 'test'";
```

The command `exec(char(Ox73687574646j776e))` converts the hexadecimal encoding to actual character and execute it. The encoded string (Ox73687574646j776e) will be converted into shutdown and gets executed.

**Cryptographic technique.** The various cryptographic techniques involve encryption a plain text into the ciphertext and decrypting the cipher into plain text. The ciphertext can be transmitted across an unsecured channel.

**Encryption.** Encryption is converting a plaintext or data into ciphertext using a key. Only authorized persons can decrypt the ciphertext into readable plaintext. One of the best methods to transmit data across the network is encryption. Symmetric and asymmetric encryption are the two types of encryption namely.

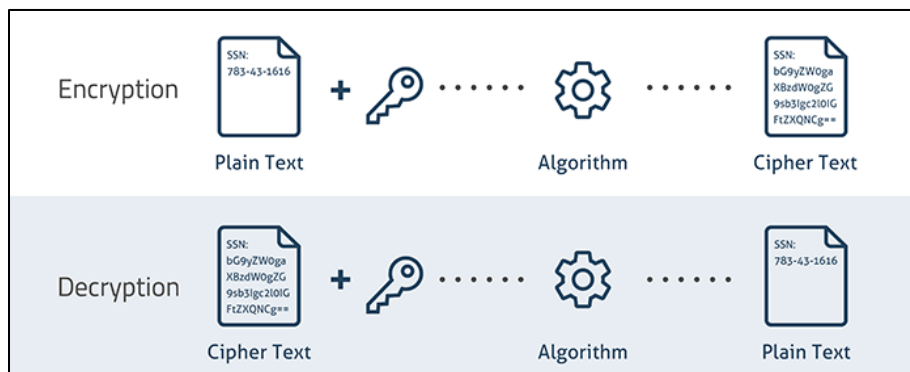


Figure 13: Encryption ("Encryption", n.d.)

The above figure shows the structure of encryption and decryption.

**Symmetric encryption.** A shared key used for both encryption and decryption. A plaintext converted into ciphertext with the help of a key and encryption algorithm. This ciphertext sent to the receiving end. The same key with the decryption algorithm is used to get the plaintext from the ciphertext.

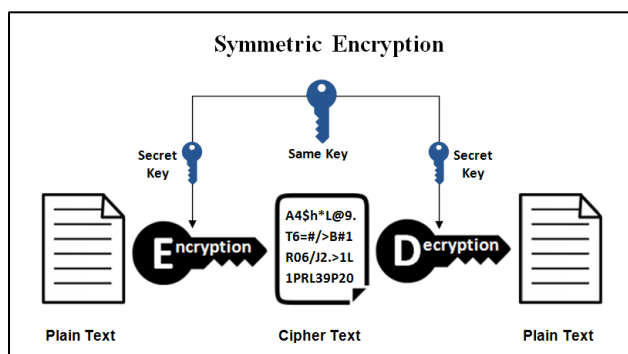


Figure 14: Symmetric encryption ("Asymmetric vs Symmetric encryption differences", n.d.).

The above figure shows the symmetric encryption process which accepts the same key for encryption and decryption. A secure channel is used to send the key.

Examples of Symmetric Algorithms: AES, DES, Blowfish, Twofish, 3DES, and RC4

*Asymmetric encryption.* Two different keys for both encryption and decryption.

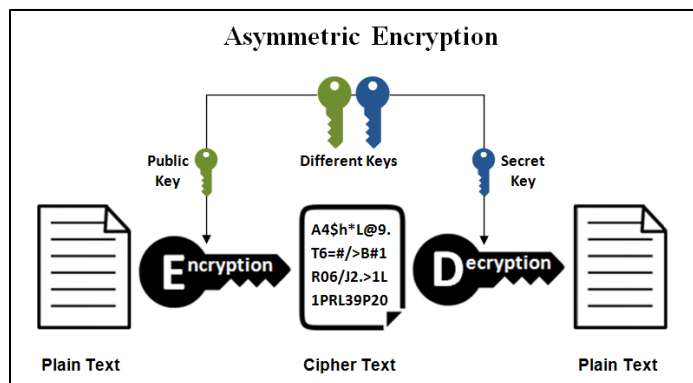
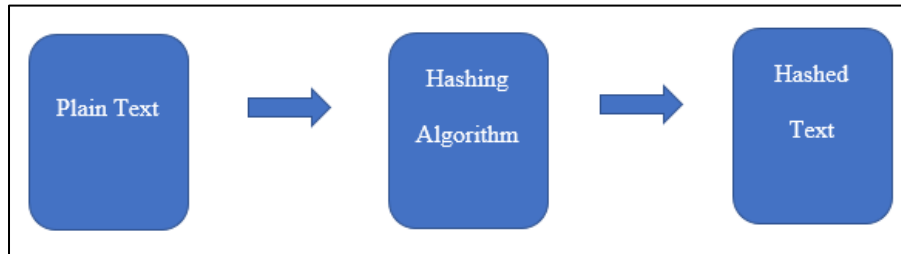


Figure 15: Asymmetric encryption ("Asymmetric Vs Symmetric encryption differences", n.d.)

The above figure shows the structure of the asymmetric encryption. A plaintext is converted into ciphertext with the help of a public key (known to everyone) and an encryption algorithm and sent to the receiving end. The secret key and a decryption algorithm are used to convert ciphertext into plaintext.

Examples of Asymmetric algorithms: Diffie-Hellman Key Exchange, RSA, SHA-3, and SHA-224

**Hashing.** Conversion of a string of characters into a fixed length value. Hashing is a one-way function.



*Figure 16: Hashing.*

The above figure a hashing model. A plaintext is sent to a hashing function to obtain a message digest (hashed text).

Examples of hashing algorithm: MD5, and SHA-1.

**Cryptographic Algorithms.** The cryptographic algorithms are listed below:

1. AES
2. Triple-DES
3. Blowfish
4. Twofish
5. RSA

Table 1

*AES-Key Size with the Number of Rounds*

Key Size (Bit)	No. of Rounds
128	10
192	12
256	14

The above table shows the comparison between the 'key size' and number of 'rounds' in the AES algorithm. The number of rounds will increase with key size.

**AES.** Advanced encryption standard (AES) is a symmetric key algorithm. The plaintext is 128 bits, and the key is also 128 bits. The number of rounds varies with key size as shown in the above table.

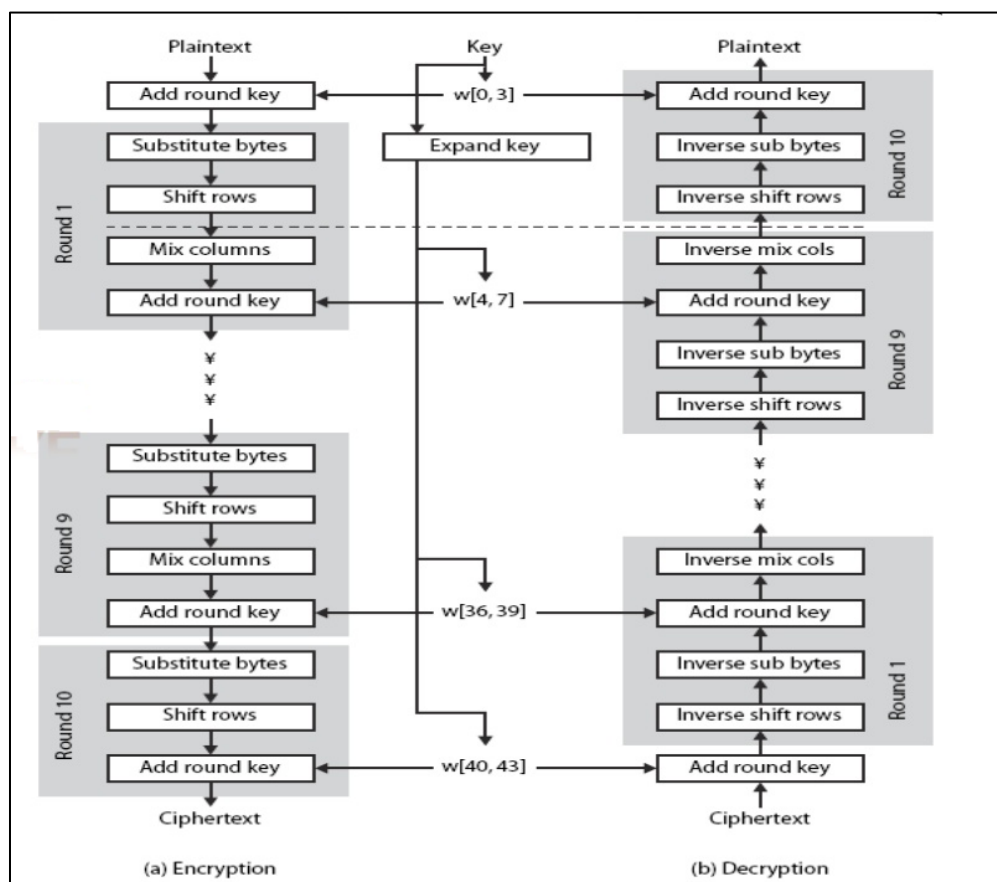


Figure 17: AES–block diagram (Stallings, n.d.).

The above figure is the block diagram of the AES algorithm. The encryption and decryption processes are shown on the left and right respectively. Below given are the steps involved in AES encryption:

1. Add round Key: Add 128-bit Plaintext is with 128-bit Key.

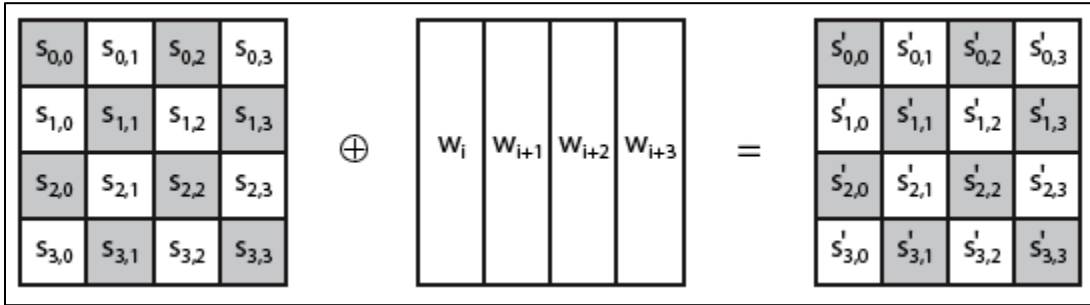


Figure 18: AES add round key (Stallings, n.d.).

The above figure shows the process of adding plain text ( $S_{a,b}$ ) and key ( $W_i$ ) to obtain the output ( $S'_{a,b}$ ).

2. Substitute Bytes: The matrix derived from the previous step is substituted by values in the S-box as shown below.

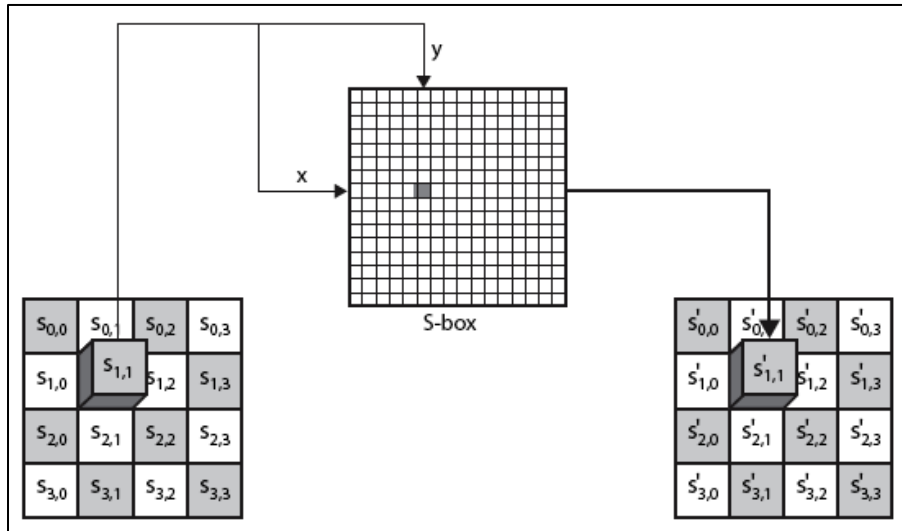


Figure 19: AES substitution bytes (Stallings, n.d.).

The above figure shows the substitution bytes. A block from  $S_{a,b}$  and  $S'_{a,b}$  are taken and matched with the S-box to obtain the substitution bytes.

3. Shift Rows: The first row of the received matrix remains unchanged. Second, third, and fourth rows do one byte, two-byte, three-byte circular shift to the left respectively.

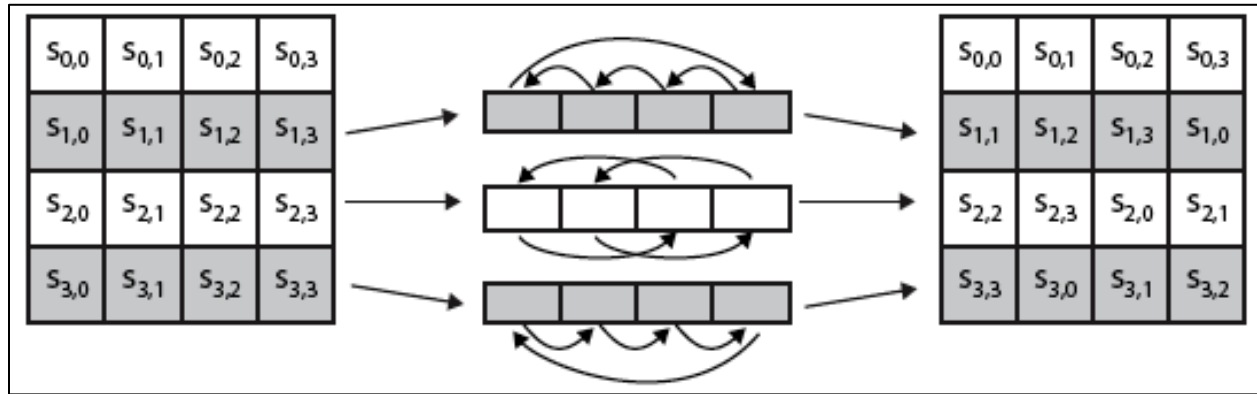


Figure 20: AES – Shift rows (Stallings)

The above figure shows the shift rows. The values are moved to different boxes as shown to form a new matrix  $S_{a,b}$ .

4. Mix column: Multiply the Matrix obtained from the previous step by a standard Matrix (shown below).

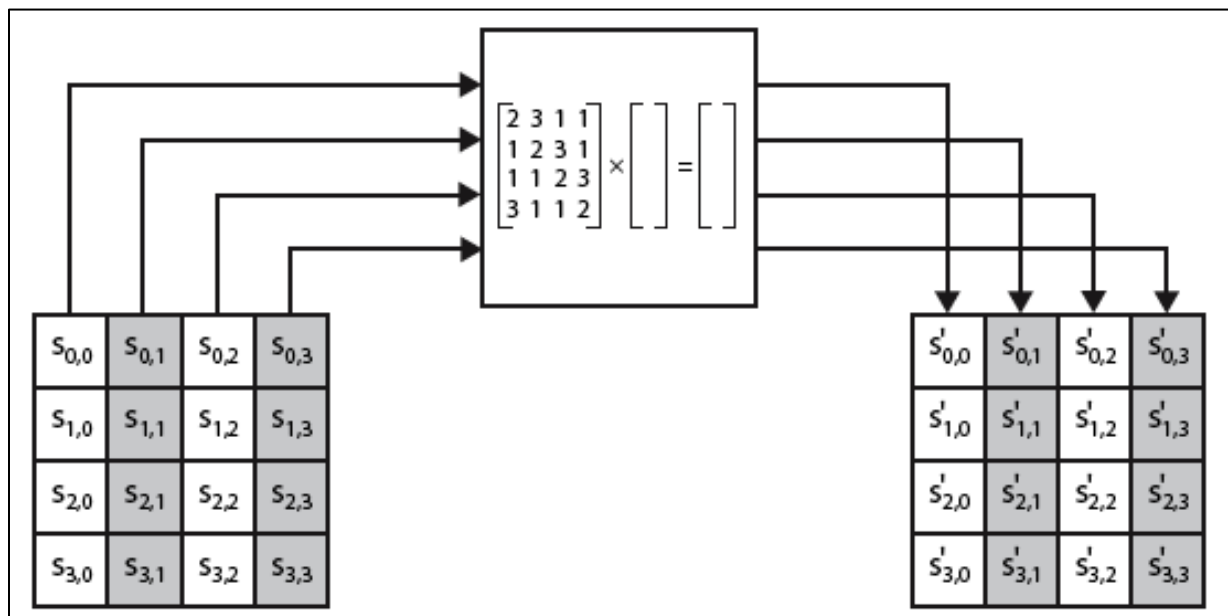


Figure 21: AES–mix column (Stallings, n.d.).



The above figure shows the multiplication between  $S_{a,b}$  and standard matrix to obtain the new Matrix  $S'_{a,b}$ .

5. Key Expansion: To get the key for the next add row key,

- $W_3$  goes through a function 'g' and XOR with  $W_0$  to get  $W_4$
- $W_1$  XOR with  $W_4$  to get  $W_5$
- $W_2$  XOR with  $W_5$  to get  $W_6$
- $W_3$  XOR with  $W_6$  to get  $W_7$

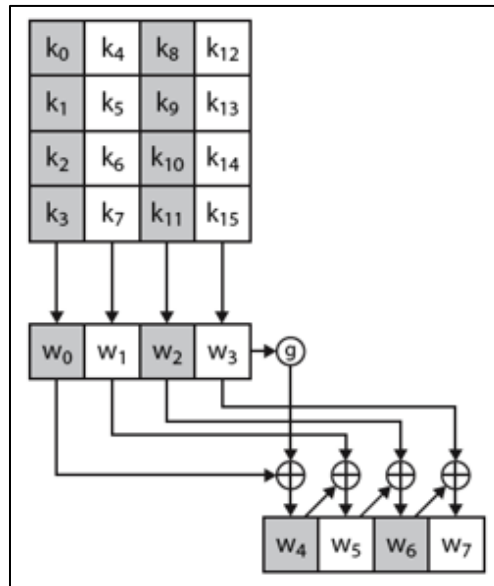


Figure 22: Key expansion (Stallings, n.d.).

The above figure shows the KEY expansion process in which  $W_0$ -3 goes through the function 'g' to obtain the  $W_4$ -7.

6. Add round Key: The newly acquired key [ $w_4, w_5, w_6, w_7$ ] is XOR with the output received from step 4.

The round 1 (Step 2 to step 6) repeated for nine times. For round 10, there is no mix column step. The output obtained from 'round 10' is the ciphertext. The reverse process of encryption is

decryption in which the cipher is given as input to get the plaintext. Inverse substitute byte, inverse shift row, and inverse mix column are used in decryption.

**Triple DES.** Data encryption standard (DES) is a symmetric algorithm. The plaintext and KEY size are 64 bits.

*Encryption.* The steps involved in the encryption process of triple DES discussed below:

1. The 64-bit plaintext sent to an initial permutation. The Initial Permutation is nothing but changing the bit positions of the initial plaintext.

Round 1 (Step 2 to Step 7):

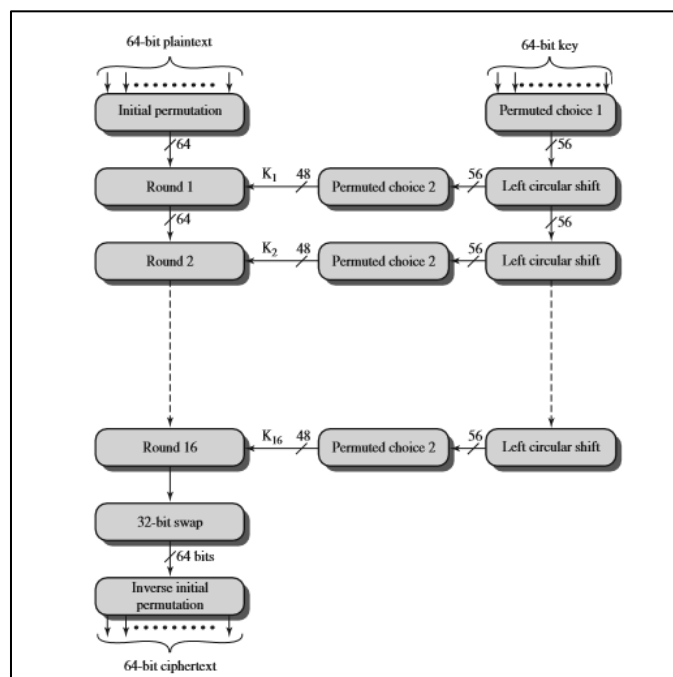


Figure 23: DES block diagram (Stallings, n.d.).

The above figure shows the block diagram of the DES encryption. There are 16 rounds, and each of them has a 64 bit (Initial permutation) and 48 bit (key) as an input to obtain a 64-bit output.

2. The output obtained from the previous step is divided into two left halves ( $L_{i-1}$ ) and right half ( $R_{i-1}$ ) of 32-bit each.

3. Key Expansion.

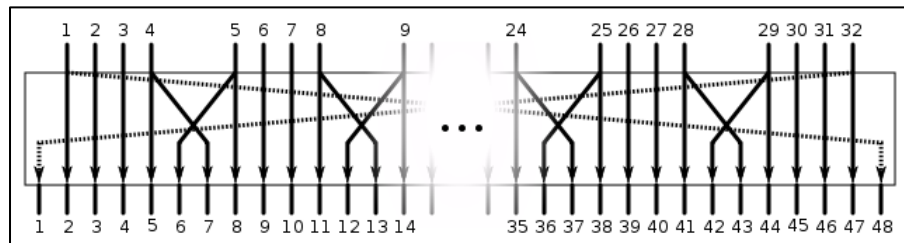


Figure 24: DES-expansion function ("DES Expansion", n.d.).

The above figure shows the DES expansion function in which the Right half 32-bit ( $R_{i-1}$ ) is taken to the expansion function (shown below) and converted into 48-bit.

4. The 48-bit output obtained from the previous step is XOR with Key ( $K_i$ ).
5. The output sent to the substitution box (S-Box) which converts 48-bit into 32-bit as shown below.
6. The 32-bit output - sent to a permutation function.
7. The output is XOR with Left half ( $L_{i-1}$ ) to get  $R_i$  and  $R_{i-1}$ , which becomes the new  $L_i$ .
8. This round repeated for 16 times. After the completion of the sixteenth round, the output is sent to a final permutation to obtain the ciphertext.

*Key generation.* Below given are the steps involved in the KEY generation process:

1. The 64-bit key is sent to a parity drop and converted into 56-bit.
2. The 56-bit output divided into the left half ( $C_{i-1}$ ) and right half ( $D_{i-1}$ ).
3. For the first round, both the left and right half shifted one bit left. The output becomes the new  $C_i$  and  $D_i$ .
4. The output will be sent to permutation choice 2 to convert 56-bit into a 48-bit output.

5. The output from step 4 will be sent to the encryption algorithm as shown in the diagram above.

*Decryption.* is a reverse process of encryption, which accepts the ciphertext and provides the plaintext.

In triple DES, there are 3 DES algorithms namely DES, inverse DES and DES are combined to make the encryption. Similarly, inverse DES, DES and inverse DES are combined to form the decryption algorithm. Below given is the block diagram of triple-DES.

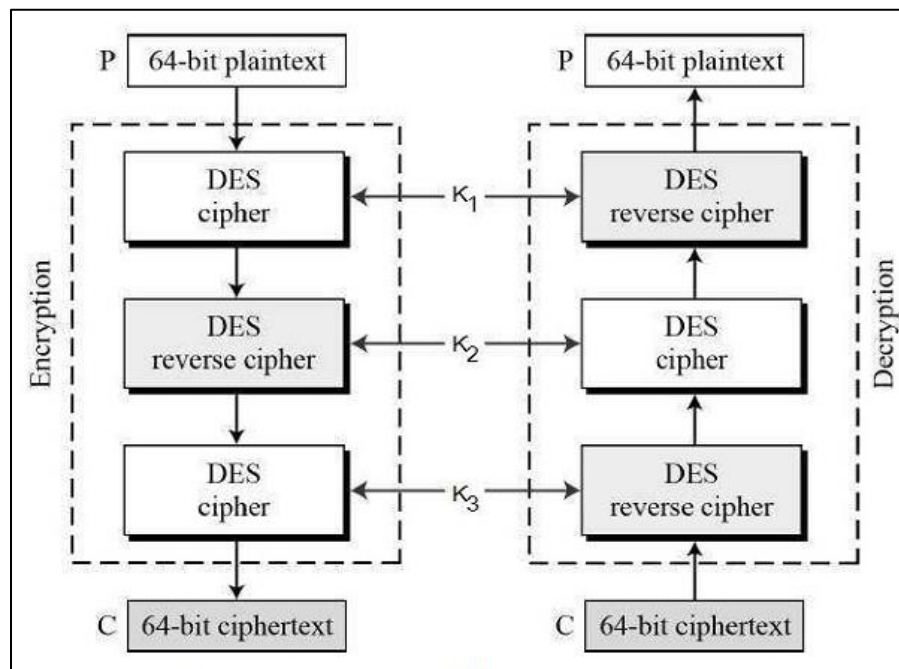


Figure 25: Triple DES ("Triple DES", n.d.).

The above figure shows the block diagram of the Triple DES algorithm. The left and right blocks show the encryption and decryption process respectively. Three different keys (K1, K2, and K3) are used for both the encryption and decryption.

**RSA** is an asymmetric algorithm. Rivest, Shamir, and Adleman designed it in the year 1977. The steps involved in the RSA algorithm is below:

STEP1: Choose any two random prime numbers P and Q. Calculate N by multiplying P and Q. ( $N = PQ$ ).

STEP2: Euler's Totient function [ $\phi(n)$ ] is calculated as given below

P and Q [ $\phi(n) = (P-1)(Q-1)$ ]

STEP3: Select a value for e, such a way that it must smaller and co-prime to  $\phi(n)$ .

STEP4: Calculate  $de = 1 \pmod{\phi(n)}$ . Where 'e' is the value, is from step3. Using the Euclidean algorithm find the value of d.

Private Key: P, Q,  $\phi(n)$ , and d.

Public Key: n and e.

*Encryption.* To calculate the value of ciphertext (C) each character of the plaintext must be through the below formula.

$$C = m^e \pmod{n}.$$

*Decryption.* Decrypt each character of the ciphertext using the given formula to get the plaintext.

$$M = C^d \pmod{n}.$$

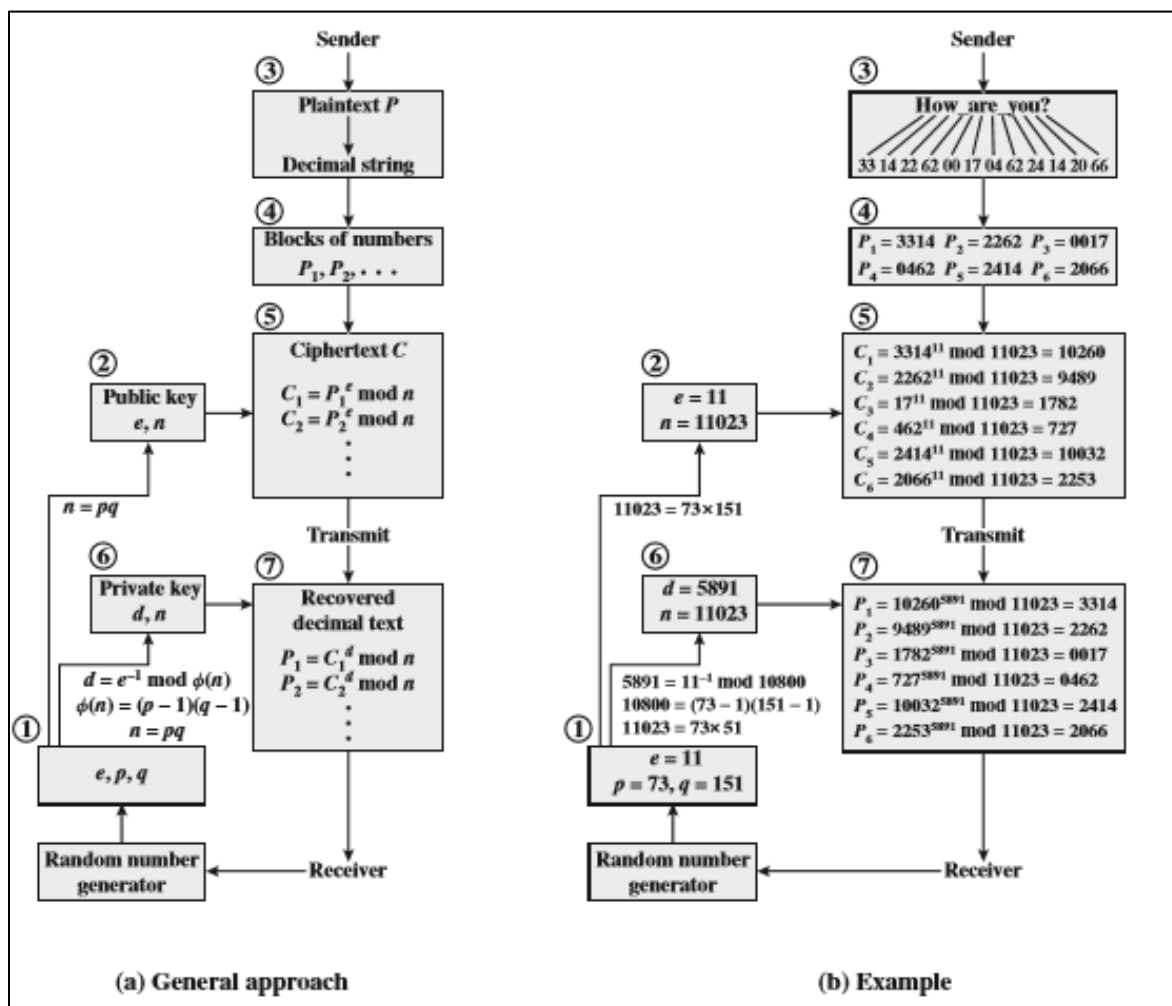


Figure 26: RSA process (Stallings, n.d.).

The above figure is the block diagram of the RSA algorithm. A general approach is on the left side, and an example is on the right side of the picture. ‘How are you?’ is chosen as plain text in the example to generate a ciphertext.

**Blowfish.** The algorithm was designed to replace DES in 1993 by Bruce Schneier. The plaintext was divided into 64 blocks and encrypted individually. Speed and efficiency are the main advantages of the Blowfish algorithm.

Blowfish has 64bit block size, and the KEY length may vary from 32 to 48 bits. It uses a 16 round Feistel cipher and uses large key-dependent S-boxes. Steps involved in Key expansion, encryption and decryption are given below.

*Encryption.* is converting a plain text into a ciphertext.

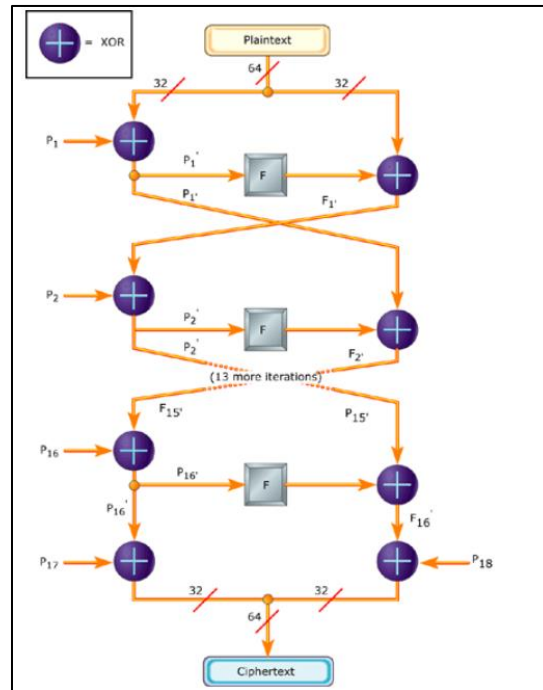


Figure 27: Blowfish process ("Blowfish", n.d.).

The above figure shows the blowfish encryption block diagram. The plaintext of 64 bit is divided into two 32 bits and processed. The steps involved in the process is listed below:

STEP1: Divide the plaintext (64 bit) into two 32 bits (Left half  $L_i$  and Right half  $R_i$ ).

STEP2: XOR the left half ( $L_i$ ) with the first element of P-array ( $P_1$ ) to get  $P_1'$ .

STEP3:  $P_1'$  run through the function ( $f$ ) and then XOR with the Right half ( $R_i$ ) to get  $F_1'$ .

STEP4:  $F_1'$  and  $P_1'$  replaces the left half ( $L_2$ ) and right half ( $R_2$ ) respectively.

STEP5: Step2 to Step4 repeated for 15 rounds with successive P-array ( $P_i$ ).

STEP6: F16' and P16' are XORed with P17 and P18 respectively to get two 32-bit output.

STEP7: The two 32-bit outputs are combined to get the 64-bit Ciphertext.

Function (F): The block diagram of the Function is below.

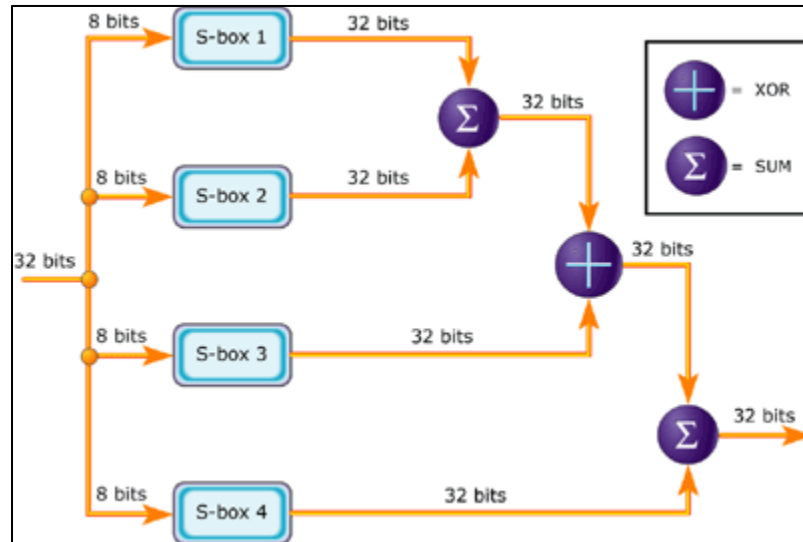


Figure 28: Blowfish function ("Blowfish", n.d.).

The above figure shows the function involved in the Blowfish. A 32-bit input is divided into four 4 bits and processed. The steps involved in the process is below:

STEP1: The 32-bit P1' is divided into 48 bits and given to S-boxes. (First 8 bit sent to S-box 1, second 8 bit sent to S-box 2, and so on.)

STEP2: The result obtained from S-box 1 and S-box 2 added together.

STEP3: The result obtained from S-box 3 and STEP2 are XORed.

STEP4: Add the result from S-box 4 and STEP3 to get the required 32-bit output.

Key Expansion. The P-array and S-array values computed from the key. The Key may get discarded after the conversion into P-array and S-array.



To generate the keys, the P-arrays which consists of 8 32bit subkeys and four 32-bit S-box with 256 entries must be ready.

STEP1: Initialize P-array and S-boxes with a fixed string. The string has the hexadecimal value of Pi (shown below).

P1 = 0x243f6a88,

P2 = 0x85a308d3,

P3 = 0x13198a2e,

P4 = 0x03707344, and so on.

STEP2: The KEY is divided into 32-bit blocks, and XOR with the initial elements of P-array and S-array and the results were return into the arrays.

STEP3: All-zero string is encrypted, and results written into the P-arrays and S-arrays. Now the P-array and S-array are used in the process of encryption.

**Twofish.** This algorithm is a symmetric algorithm. Twofish is the most used algorithm after blowfish due to its simple structure.

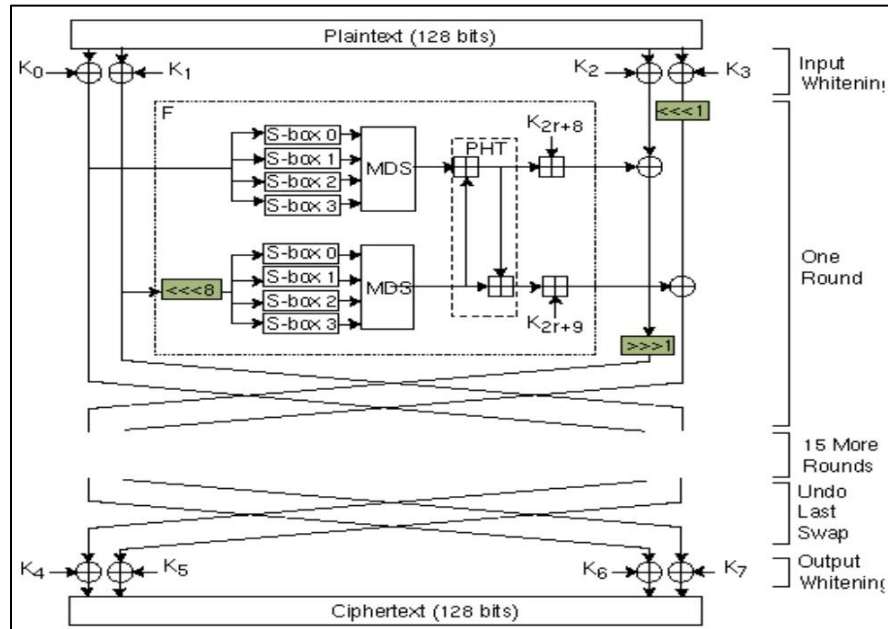


Figure 29: Twofish process ("twofish", n.d.).

The above figure is the block diagram of the Twofish algorithm. The 128-bit plaintext and KEY are accepted to generate a ciphertext of 128 bit. The steps involved in the encryption process is below

*Encryption.* Below given is the process involved in the Twofish encryption:

STEP1: The 128-bit plaintext and key is separated into four 32-bits [Plaintext: P0, P1, P2 and P3, Key: K0, K1, K2, and K3]

STEP2: P0, P1, P2, and P3 are XORed with K0, K1, K2, and K3 respectively to obtain R0, R1, R2, and R3.

Round-1:

STEP1: R0 and R1 are sent through a function(F) to obtain F0 and F1.

STEP2: R2 is XOR with F0, and the output is rotated right by one bit to get C2. R3 is turned left by one bit and XOR with F1 to get C3.

STEP3: R0, R1, C2, and C3 is swapped as C2, C3, R0, and R1 and sent to round two.

Function (F):

STEP1: R0 and R1 are sent to the Function (G) to obtain T0 and T1.

STEP2: T0 and T1 are sent through Pseudo-Hadamard Transformer (PHT) to get a and b.

STEP3: The results of the previous step is combined with two keys namely  $K_{2r+8}$  and  $K_{2r+9}$  to give F0 and F1.

Function (G):

STEP1: A 32-bit input (R0) is divided into four 8-bits and sent to four different S-boxes.

STEP2: The four 8-bit outputs obtained from the previous step is sent to MDS to get a 32-bit output.

MDS:

STEP1: The most significant bits input (Y0, Y1, Y2, and Y3) are multiplied with the MDX matrix to form the 32-bit output.

*Key Schedule.* is the process involved in the KEY generation.

STEP1: Two keys with 32-bits is combined with the output from the whitening to get a key schedule of  $K_0, K_1 \dots K_{39}$ .

STEP2: The global key obtained from the user is  $M [m_0, m_1, m_2 \dots m_{15}]$ .

STEP3: The RS matrix is multiplied with the first half of the user key  $[m_0, m_1, m_2 \dots m_7]$  to get  $S_{0,0}, S_{0,1}, S_{0,2}$ , and  $S_{0,3}$ .

STEP4: The second half of the user key  $[m_8, m_9 \dots m_{15}]$  is multiplied with RS to get  $S_{1,0}, S_{1,1}, S_{1,2}$ , and  $S_{1,3}$ .

STEP5: Both S0 (S0,0, S0,1, S0,2, and S0,3) and S1 (S1,0, S1,1, S1,2, and S1,3) are sent to the function G.

STEP6: The even and odd numbered keys are separated and sent to the function F.

**Encryption of confidential data.** AES can be used to hide some of the confidential information such as account number, email ID, phone number, etc., while hashing functions can be used to authenticate. The average processing speed to encrypt and decrypt the data using the AES algorithm is 0.499 sec and 0.494 sec respectively (Sonakshi, Kumar, & Gopal, 2016). All information obtained from the user can be encrypted using cryptographic algorithms and stored in the database.

Table 2

*Data Stored after Encryption*

Username	Password	email
hXtTFI6LuWpmEus4GZOeZw==	hmChiPFEAic1qw2jUN+mEw==	rjMuFRqhHQN64uEGnJNz4Q==

The above table shows the encrypted form of username, password, and email in a table. The attackers cannot do anything with the above information. Without knowing the algorithm and key used it is not possible to decrypt the data. This method keeps the database more secure while comparing with the standard database.

### **Data Collection**

Data collected under two different conditions:

1. Encrypt only the password field
2. Encrypt both username and password field

**Encrypting only the password field.** The data collection table is below.

Table 3

*SQLIA vs. Cryptographic Algorithms* (Encrypted field: Password)

SQL Injection Attacks	AES	Tipple DES	RSA	Blowfish	Twofish
<b>Tautologies</b>					
<b>Illegal/Logically incorrect queries</b>					
<b>Union Queries</b>					
<b>Piggy Backed Queries</b>					
<b>Stored Procedure</b>					
<b>Inference</b>					
<b>Blind Injection</b>					
<b>Timing Attack</b>					
<b>Alternate Encoding</b>					

X  Prevented

—  Not Prevented

The above table is to record the response of the web application during the attacks. The password field is encrypted using five algorithms while conducting SQLIA. The symbols ‘X’ and ‘-’ are used to show that the attack was prevented and not prevented respectively.

**Encrypting both username and password.** The data collection table is below.

Table 4

*SQLIA vs. Cryptographic Algorithms* (Encrypted field: Password and Username)

SQL Injection Attacks	AES	Tipple DES	RSA	Blowfish	Twofish
<b>Tautologies</b>					
<b>Illegal/Logically incorrect queries</b>					
<b>Union Queries</b>					
<b>Piggy Backed Queries</b>					
<b>Stored Procedure</b>					
<b>Inference</b>					
<b>Blind Injection</b>					
<b>Timing Attack</b>					
<b>Alternate Encoding</b>					

X → Prevented

— → Not Prevented

The above table is to record the response of the web application during the attacks. The password and username fields are encrypted using five algorithms while SQLIA. The symbols 'X' and '-' are used to show that the attack was prevented and not prevented respectively.

The experiment is conducted using PHP, MySQL, HTML, CSS, and JavaScript. The code is implemented in the XAMPP server and tested. PHP is used to code the algorithms. The source code was downloaded from 'phpseclib' and added to the actual libraries.

```

<?php
    include 'Crypt/AES.php';
    $aes = new Crypt_AES();
    $aes->setKey('ThisIsTheKey'); //Provide the Key here
    $plaintext = 'ThisIsThePlainText'; //Provide the plain text here
    $encryptedText = $aes->encrypt($plaintext);
    echo "The Encrypted Text is " . $encryptedText; //Encrypted Text
    $decryptedText = $aes->decrypt($encryptedText);
    echo "The Decrypted Text is " . $decryptedText; //Decrypted Text
?>

```

*Figure 30: Using algorithms in PHP (Wigginton, 2008).*

The above figure shows the PHP code to encrypt and decrypt the plain text and cipher text respectively. The above-given example is for AES algorithm. To use other algorithms change `Crypt_AES()` to `Crypt_Alogrithm()` also include the respective PHP file. Change the plain text and key variables as needed.

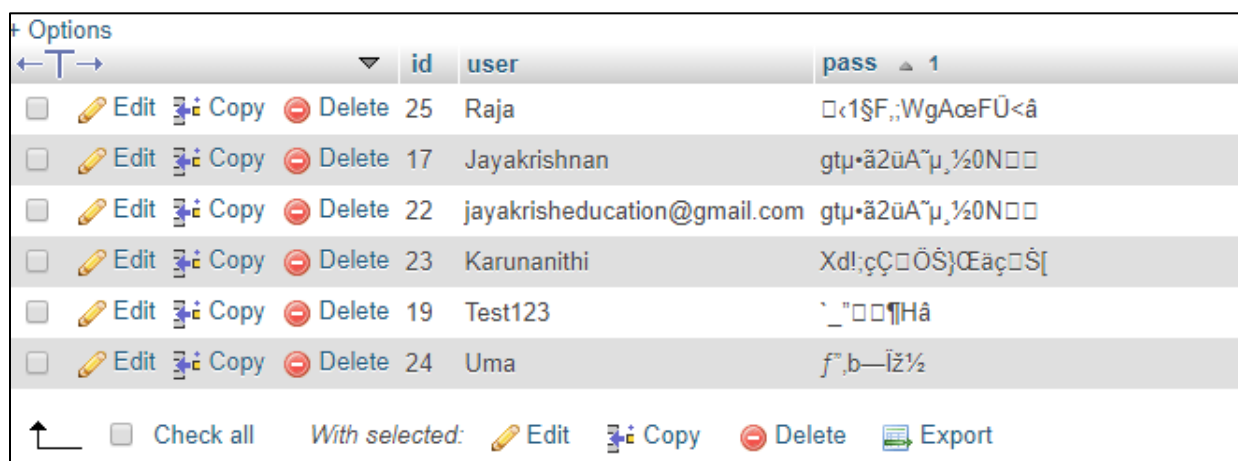
The process of generating the key varies for RSA since it uses two different keys to encrypt and decrypt. The program itself makes the keys. In this methodology, the usernames and passwords are used as plain text and encrypted using various algorithms. During the authentication process, the username and password provided by the user are encrypted and compared with the details already stored in the database. When they matched, the user can log in into the system.

The attacks conducted in two different conditions. Firstly, encrypted only the password field and performed all the attacks and recorded the output. Secondly, encrypting both the username and password field and conducted all and record the output. This method will help to understand the efficiency of the application in both the conditions.

The speed test is also conducted across all the algorithms to determine the fastest algorithms. Along with the security, speed is considered an essential factor. The performance of the web application should not be affected due to the implementation of the cryptographic algorithms. Performance speed, security, and design of each algorithm are measured and recorded.

**SQLIA after implementation.** All the SQL injections attacks performed on the web application after the implementation of the algorithms.

**Encrypting only the password field.** The password field is encrypted, and the username field is not encrypted.



The screenshot shows a database table with three columns: 'id', 'user', and 'pass'. The 'pass' column contains encrypted password values. The table has a toolbar at the top with 'Options', 'Edit', 'Copy', and 'Delete' icons. At the bottom, there is a 'Check all' checkbox and a 'With selected:' section with 'Edit', 'Copy', 'Delete', and 'Export' icons.

	id	user	pass
<input type="checkbox"/> Edit Copy Delete	25	Raja	□<1\$F,;WgAœFÜ<â
<input type="checkbox"/> Edit Copy Delete	17	Jayakrishnan	gtµ•ã2üA~µ,½0N□□
<input type="checkbox"/> Edit Copy Delete	22	jayakrisheducation@gmail.com	gtµ•ã2üA~µ,½0N□□
<input type="checkbox"/> Edit Copy Delete	23	Karunanithi	Xdl;çÇ□ÖŠ}œEäç□Š[
<input type="checkbox"/> Edit Copy Delete	19	Test123	`_ "□□¶Hâ
<input type="checkbox"/> Edit Copy Delete	24	Uma	f",b—İž½

Figure 31: Database with encrypted password.

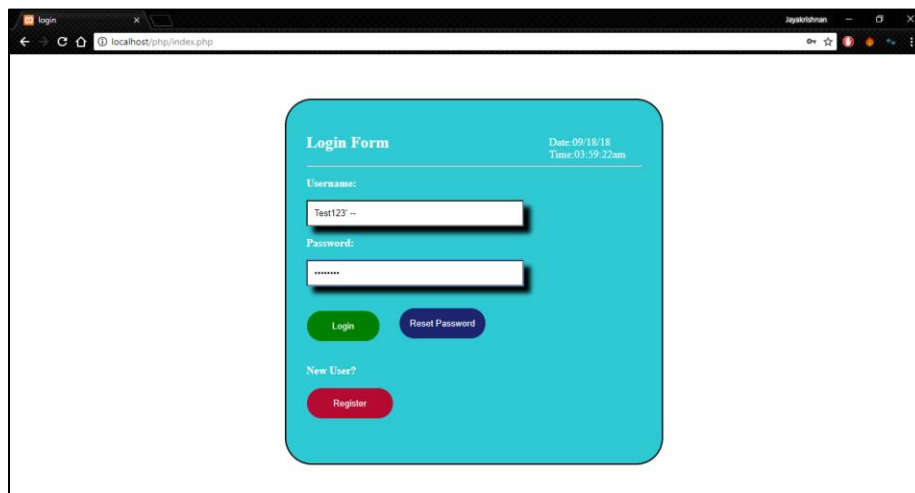
The above figure shows a table in the database after encrypting the password field. Even if the attackers gained access to the database, they would not be able to see the passwords.

**Tautology.** Attack conducted on the web application by providing the below-given username and password.

Username: Test123'; --

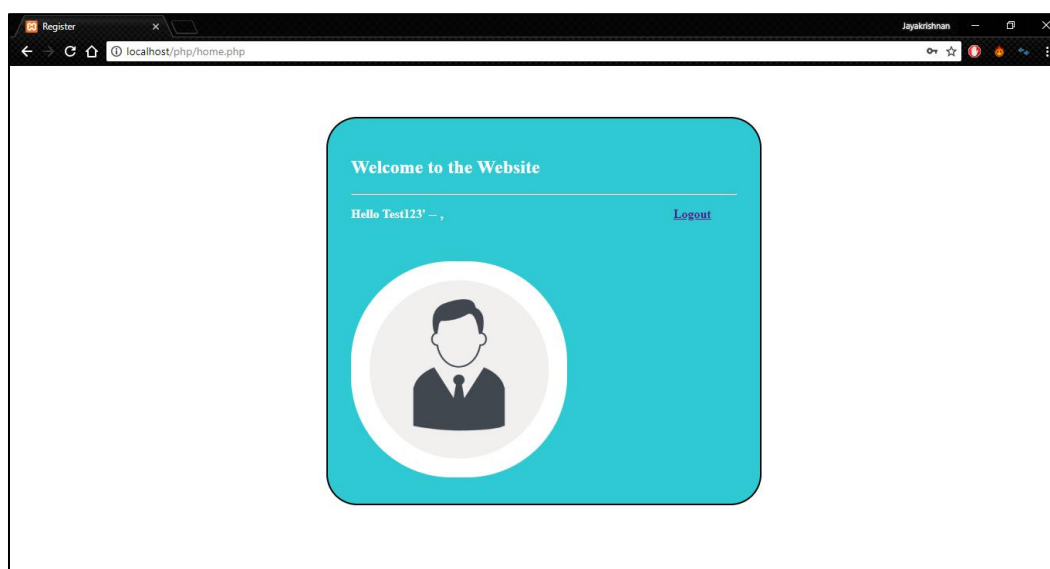
Password: any password





*Figure 32: Tautology attack.*

The above figure shows the login page of the web application with username and password as 'Test123'; -- ' and 'anyPassword' respectively to conduct a Tautology attack.



*Figure 33: Successful tautology attack.*

The above figure shows the welcome page after the successful login with the 'Test123' user account. The actual and modified SQL query in the PHP script is below.

Actual SQL query:

```
SELECT * FROM `login` WHERE user = 'Test123' and pass = '``_”••Hâ'
```

Modified SQL query:

```
SELECT * FROM `login` WHERE user = 'Test123'; -- and pass = '``_”••Hâ'
```

Even though the password is encrypted the second half of the query ignored because of ‘-’-. This attack could be a success if the attacker knows the username. This method could reveal some of the confidential information of the actual user stored in the web application.

**Illegal/logical incorrect queries** attack conducted on the web application by providing some incorrect information to get some knowledge about the application or database. Based on the gathered information the attacker can perform a successful attack. In this study, illegal/logical incorrect query attack conducted by providing the below username and password.

Username: Attack’’’; --

Password: any Password

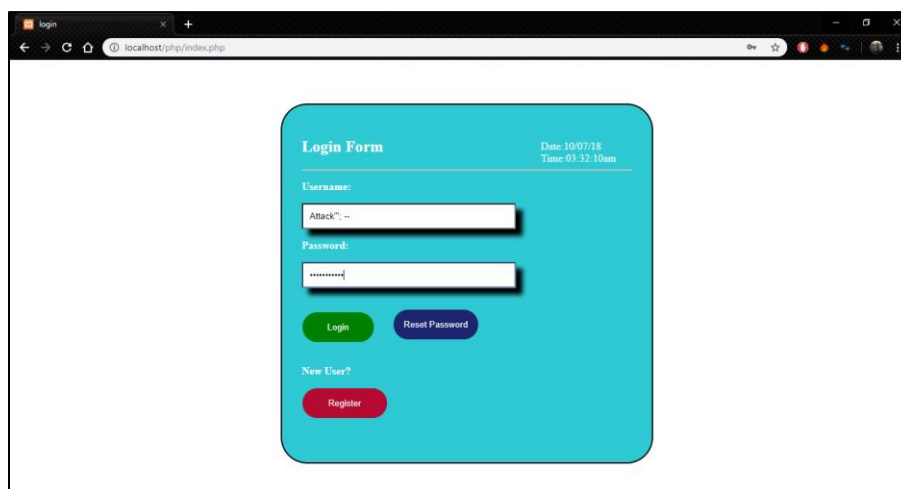


Figure 34: Illegal/logical incorrect queries attack.

The screenshot shows a web browser window with the address bar displaying 'localhost/php/index.php'. The page content includes a warning message: 'Warning: mysqli\_num\_rows() expects parameter 1 to be mysqli\_result, boolean given in C:\xampp\htdocs\php\index.php on line 78'. Below this is an error message: 'Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''; --,+++++ -+++++ -+++++' at line 1'. The main content area features a light blue rounded rectangle containing a 'Login Form'. The form has a title 'Login Form' and a date/time stamp 'Date: 10/07/18 Time: 09:24:26pm'. It includes two input fields: 'Username:' with a placeholder 'Type your username' and 'Password:' with a placeholder 'Type your password'. There are two buttons: a green 'Login' button and a blue 'Reset Password' button. At the bottom, there is a 'New User?' section with a red 'Register' button.

The above figure shows the response from the application below after clicking on the login button. The home page of the web application holds the error message thrown by the SQL server.

The error message says that there is a syntax error also it reveals crucial information about the server. The encrypted form of the password is also known. This information can be used by attackers to conduct further attacks on the web application. Repeatedly conducting this attack can end up in gaining access to the application or database.

*Union query attack.* Conducted on the web application by providing union queries in the user inputs and gaining some useful information. The below username and password is provided to perform the union query attack.

Username: Test123' union SELECT \* FROM `login` WHERE user = '123'; --

Password: any Password



*Figure 36:* Successful union query attack.

The above figure shows the welcome page after the successful login with the 'Test123' user account. The input provided by the user in the username field and the age of the user '123' is on the screen.

The attacker can successfully access the website and gain more information. In the above-given screenshot, the age of the user (123) used in the union query is also visible. If the

attacker knows the username, he/she can use them in the union query and get the details about that account. Even though the password field is encrypted it does not make any differences; the application remains vulnerable. Encryption alone cannot prevent the SQL injection attack, best coding practice, reducing the length of the user input fields also plays an essential role in preventing attackers from gaining access to the application.

*Piggy-backed queries.* Attack conducted by adding some additional SQL queries in the user input field. In this case, the actual database has three columns namely username, password, and age. The piggybacked query attack was conducted to alter the table in the database. The below username and password are used to drop the field named 'age' in the database.

Username: Test123'; ALTER table `login` drop age; - -

Password: any Password

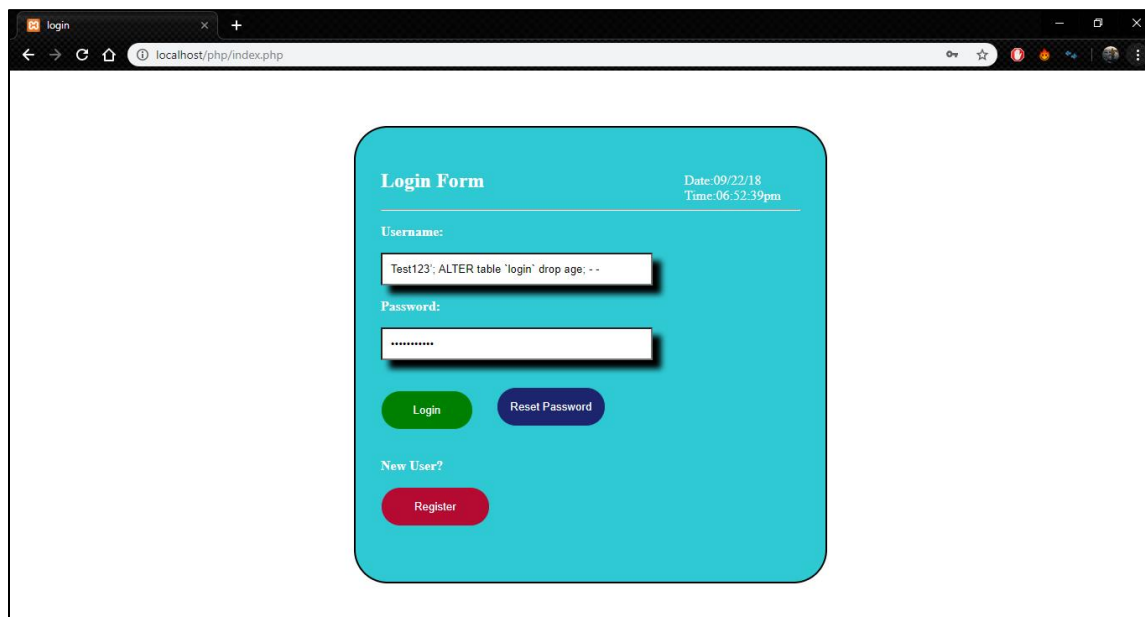


Figure 37: Piggy-backed queries attack.

The above figure shows the login page of the web application with username as 'Test123'; ALTER table `login` drop age; - -' to conduct a Piggy-Backed queries attack.

The actual query in PHP script will be modified as below:

```
SELECT * FROM `login` WHERE user = 'Test123'; ALTER table `login` drop age; --
AND pass = 'anyPassword';
```

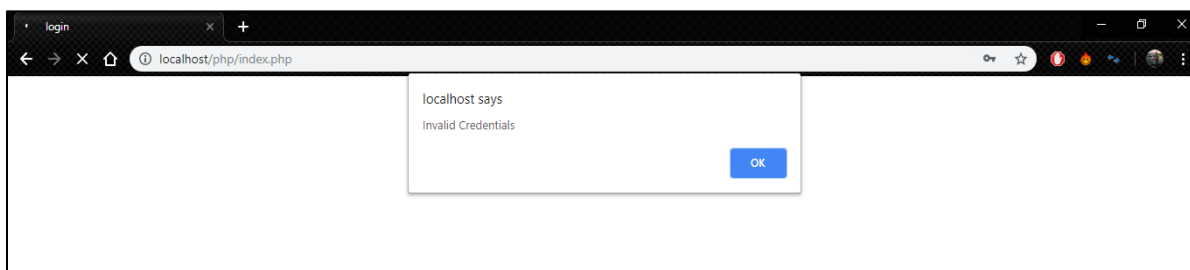


Figure 38: Failure piggy-backed queries

The above figure shows the alert given by the web application saying that the ‘credentials are invalid.’ The user will navigate to the login page on clicking the ‘ok’ button.

This attack was unsuccessful because in PHP to run multiple queries at the same time the code has to be `mysqli_multi_query(connection, query)` but in this application is coded as `‘mysqli_query(connection, query, result mode).’`

*Stored procedures.* Already written procedures in the database. In this application, stored procedures are written to authenticate the user. Below given is the stored procedure.

```
BEGIN

SET @qry = CONCAT("SELECT * FROM login",

" WHERE user = '", username, "' AND pass = '", password, "'");

PREPARE stmt FROM @qry;

EXECUTE stmt;

END
```

This procedure accepts two inputs namely username and password. They are used to authenticate the user. Once after placing this procedure in the database, it can be called from the PHP using the below statements.

```
$query = "Call login($username,$password)";
```

```
$query_run = mysqli_query($conn,$query);
```

Login is the procedure name, username and passwords are the inputs provided by the user. \$query is used to call the procedure 'Login' with the two input parameters. \$query\_run is used to connect to the database and execute the query. Stored procedure attack performed on the web application with the below-given username and password.

Username: Test123' --

Password: any Password

Once after providing the above credentials and clicked on the login button, the user was allowed to login into the application. This attack was made successfully even after the implementation of the stored procedure.

*BLIND injection.* can be conducted in web application by providing `BINARY substring(DATABASE(),1,1) = 'A'` in the user inputs. The query is to check whether the alphabet is available in the name of the database. The attacker can provide an already existing username followed by a binary query as below.

```
Test123' AND (BINARY substring(DATABASE(),1,1) = 'A'); --
```

The query is to check the first letter of the database name. If the first letter of the database is 'A,' the attacker will be able to login into the application or else the access denied. Continuously doing this for 26 times can reveal the first letter of the database. Repeatedly

making this attack for multiple times the attacker can know the name of the database. Once the attacker knows the name of the database, he/she can do a Piggy Backed queries attack to modify the database. Piggy Backed queries conducted on the application by providing the below credentials.

Username: Test123' AND (BINARY substring(DATABASE(),1,1) = 'I'); --

Password: any Password



**Login Form** Date:10/08/18  
Time:05:08:25am

Username:  
Test123' AND (BINARY substring(DATABASE(),1,1) = 'I'); --

Password:  
.....

Login Reset Password

New User?  
Register

*Figure 39: BLIND injection attack.*

The above figure shows the login page of the web application with username as 'Test123' AND (BINARY substring(DATABASE(),1,1) = 'I'); --' to conduct a BLIND Injection Attack.

The actual query in PHP script modified as given below.



```
SELECT * FROM `login` WHERE user = '123' AND (BINARY
substring(DATABASE(),1,1) = 'l');
```

Since the name of the database in this application is 'login,' the first letter is 'l' the SQL query will gain access to the web application.



*Figure 40: Successful BLIND injection attack.*

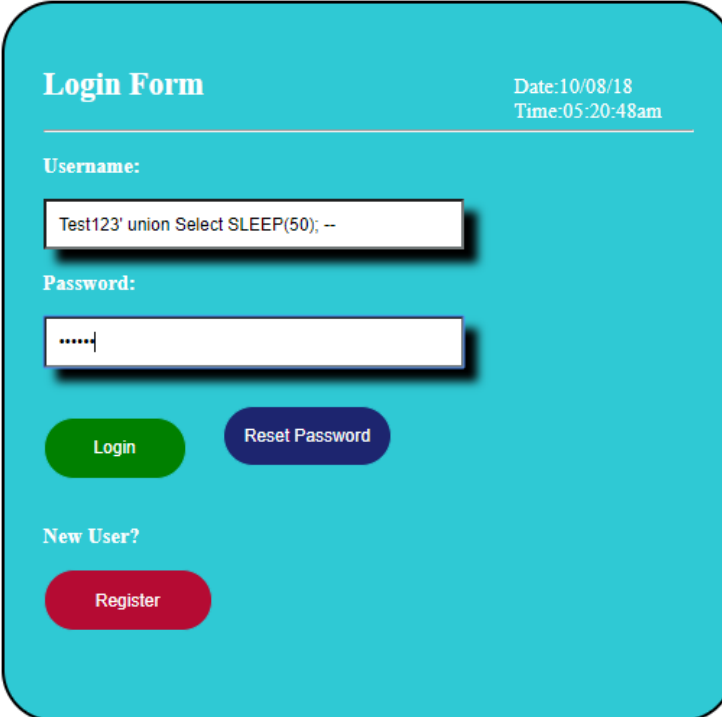
The above figure shows the welcome page after the successful login with the 'Test123' user account. The actual user input given by the user are shown.

The attacker can successfully login into the application if the first letter is 'l'. Repeatedly making this attack will tell the name of the database. Even though the password is encrypted, the attacker can successfully conduct the blind injection attack on the web applications.

*Timing Attack.* conducted to make the database to sleep for a given time. When the database sleep, it will not be able to execute any other queries. Timing Attack conducted on this web application by providing the below-given username and password.

Username: Test123' union Select SLEEP(50); --

Password: anyPassword



**Login Form** Date:10/08/18 Time:05:20:48am

Username:  
Test123' union Select SLEEP(50); --

Password:  
.....

Login Reset Password

New User?  
Register

Figure 41: Timing attack.

The above figure shows the login page of the web application with username as 'Test123' union Select SLEEP(50); -- ' to conduct a Timing Attack. The actual query in PHP script modified as given below.

```
SELECT * FROM `login` WHERE user = 'Test123' UNION select Sleep(50);
```

This query will make the database to go to sleep for 50 seconds. Since there was no replay from the database for more than 30 seconds, a fatal error thrown back to the web application. The error clarifies that the attack was a success. Below given is the screenshot.

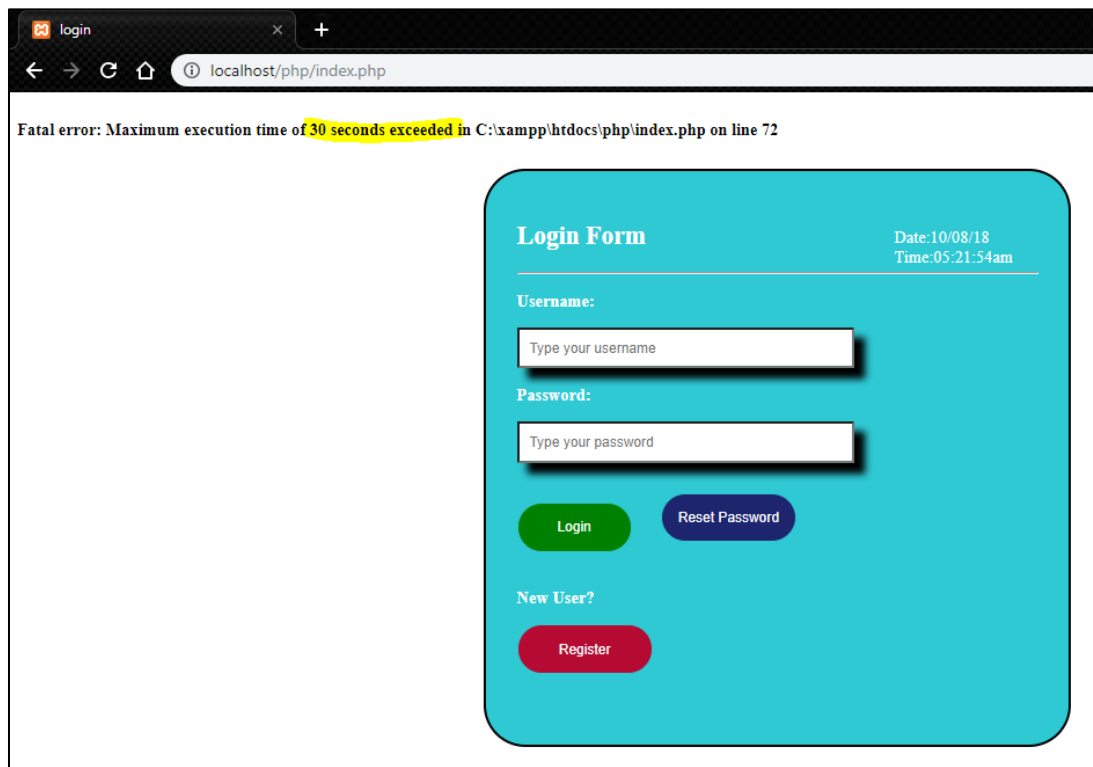


Figure 42: Successful timing attack.

The above figure shows the home page of the web application with the error message saying, 'the application exceeded the maximum execution time of 30 seconds'. This attack can be conducted repeatedly to make the database to sleep. The performance of the application can be reduced and may cause difficulties to the application users.

*Alternate encoding.* Attack conducted on the web application by providing the hexadecimal values of some malicious code and making them execute. The below-provided username is the hexadecimal version of "SHUTDOWN". If this gets performed successfully, the database will be shut down. The attack can affect the performance of the application.

Username: Test23'; exec(char(Ox73687574646j776e)) -- -'

Password: any Password

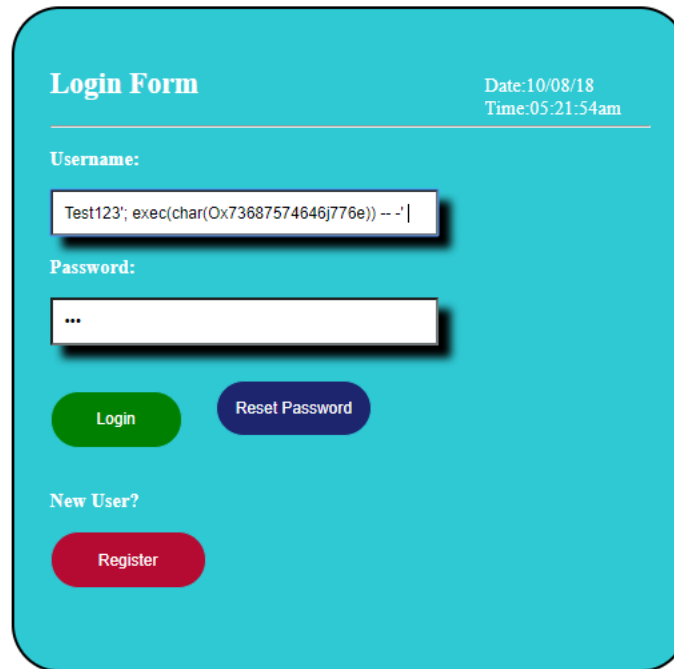


Figure 43: Alternate encoding attack.

The above figure shows the login page of the web application with username as 'Test23'; exec(char(Ox73687574646j776e)) -- -' to conduct an Alternate Encoding attack. This attempt is to SHUTDOWN the database.

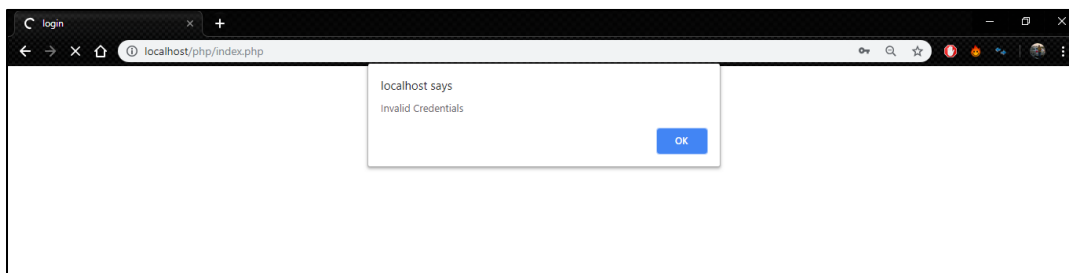
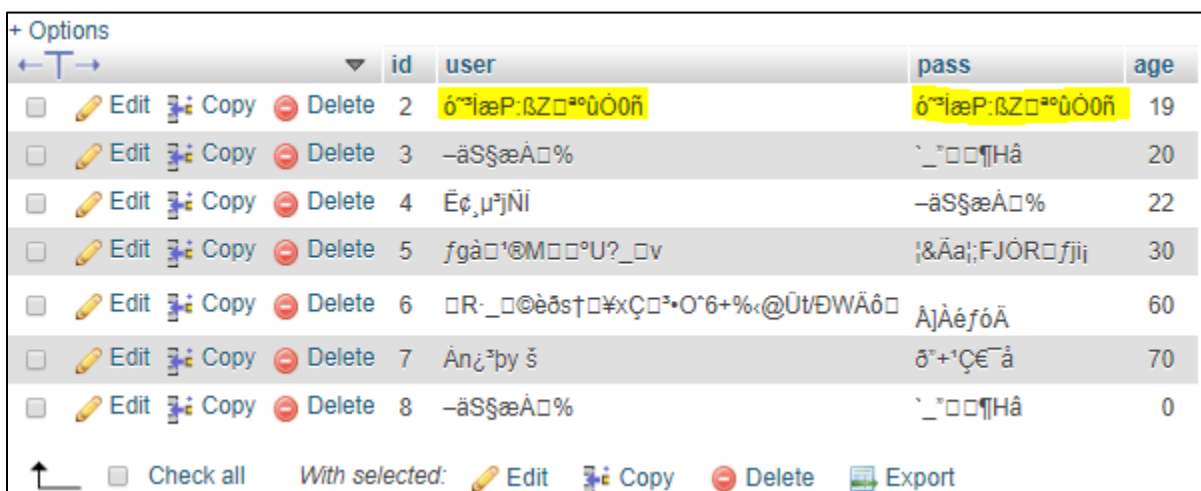


Figure 44: Failure of alternate encoding attack.

The above figure shows the alert thrown by the web application after providing the malicious code in the user inputs to shut down the database.

The attacker was not allowed to log in because the malicious code has two SQL queries. This application was designed to accept only the SQL query. If the app was designed to receive multiple SQL queries, then this could have been a successful attack.

**Encrypting both username the password field.** The web application has two inputs fields namely username and password. Both the fields are encrypted.



	id	user	pass	age
<input type="checkbox"/> Edit Copy Delete	2	ô~ æP:ßZ□°ûÔ0ñ	ô~ æP:ßZ□°ûÔ0ñ	19
<input type="checkbox"/> Edit Copy Delete	3	-äS\$æÄ□%	`_□□¶Hâ	20
<input type="checkbox"/> Edit Copy Delete	4	Ēç,µjÑi	-äS\$æÄ□%	22
<input type="checkbox"/> Edit Copy Delete	5	fğà□'©M□□°U?_□v	;&Äa!;FJÖR□fji	30
<input type="checkbox"/> Edit Copy Delete	6	□R·_□©èðs†□¥xÇ□°•O*6+%;@Ût/DWÄô□	ÄjÄéfóÄ	60
<input type="checkbox"/> Edit Copy Delete	7	Äñç°py š	ð°+¹Ç€~ä	70
<input type="checkbox"/> Edit Copy Delete	8	-äS\$æÄ□%	`_□□¶Hâ	0

Figure 45: Database-encrypted both username and password.

The above figure shows a table in the database after encrypting the username and password fields. Since all the input fields are encrypted, the attacker might not be able to get some useful information from the database.

Attackers might find it challenging to insert malicious code through user inputs since everything will be encrypted and become meaningless. All the SQL injections attack conducted on the web application after encrypting both the username and password fields.

**Tautology.** Attack conducted on the web application by providing the below-given username and password.

Username: Test123' --

Password: any Password

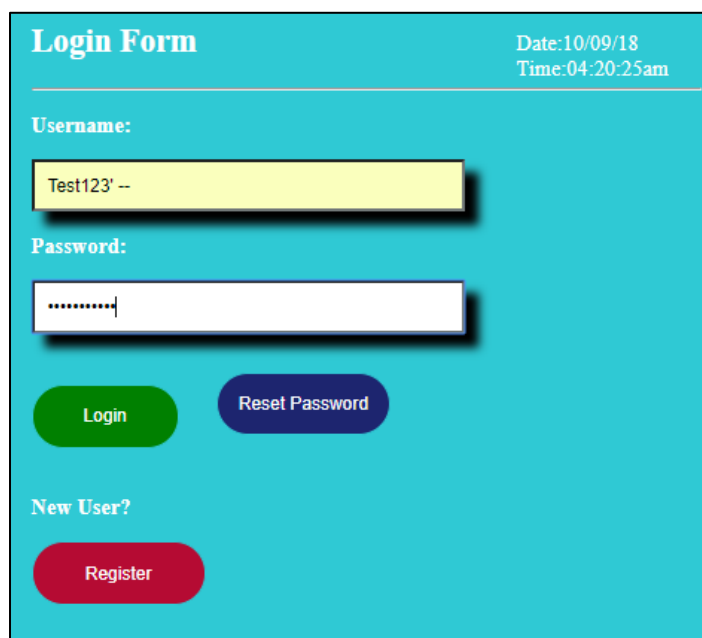


Figure 46: Tautology attack after encrypting both the fields.

The above figure shows the login page of the web application with username as 'Test123' -- ' to conduct a Tautology attack. This attempt is to get into the Test123 account without the correct password.

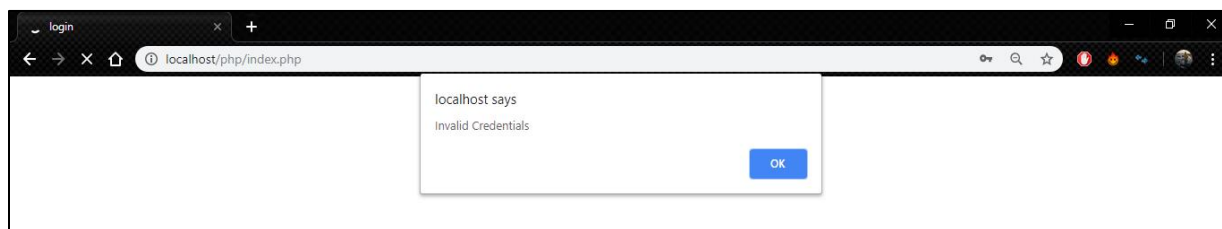


Figure 47: Failed tautology attack after encrypting both the fields.

The above figure shows the alert thrown by the web application after clicking the login button. The actual and modified SQL query after the Tautology attack is below.

Actual SQL query:

```
SELECT * FROM `login1` WHERE user = '-äS§æÁ•%' and pass = '`_”••¶Hâ';
```

Modified SQL query:

```
SELECT * FROM `login1` WHERE user = '¶,,•F•Íu•môG-Òì' and pass = '•F•Íu•¶,,•';|
```

Even though the malicious code injected through the user inputs, it went encrypted and compared with the encrypted text in the database. Since there was a mismatch, an error thrown by the application. Any tautology attack prevented by using this method. Encrypting the whole user inputs makes them meaningless and stop the attacker from gaining access to the database. This method does not require user input validation. This method is most comfortable and efficient to prevent attackers from attacking the web application.

*Illegal/Logical Incorrect Queries.* Attack conducted on the web application by providing the below username and password.

Username: Attack'''; --

Password: any Password

Please find the response from the application below after clicking on the login button.

Successful Illegal/Logical Incorrect Queries attack

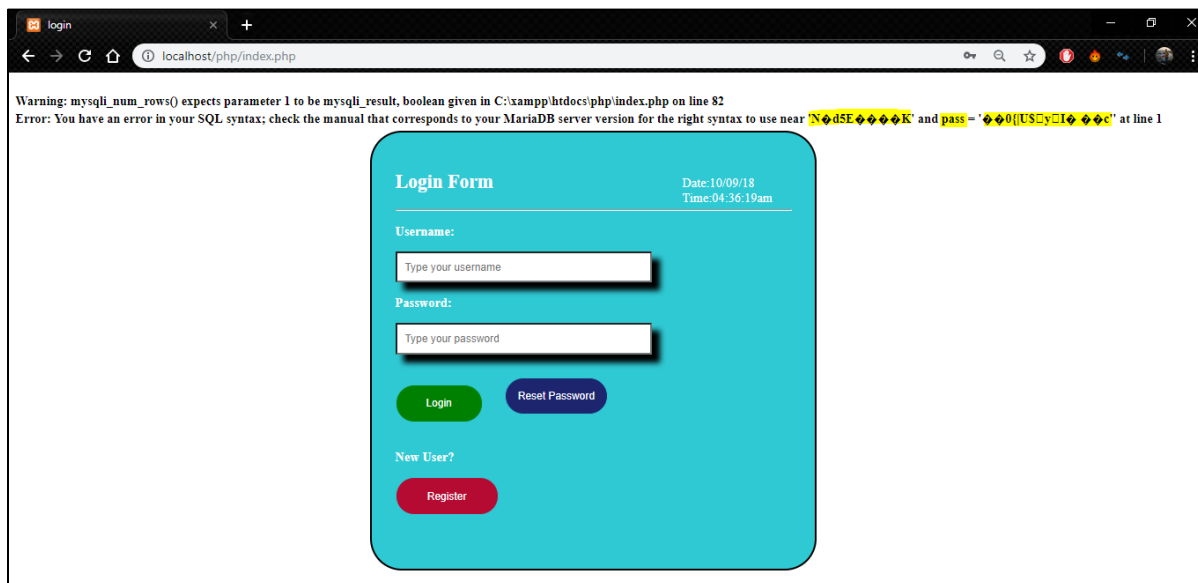


Figure 48: Successful illegal/logical incorrect queries attack—encrypting all the field.

The above figure shows the login screen with an error message after conducting the Illegal/Logical Incorrect Queries attack on the web application. The error message is below.

Error: An error in the SQL syntax; check the manual that corresponds to MariaDB server version for the right syntax to use near 'N?d5E???K' and pass = '?0{|U\$yI? ??c' at line 1

Information obtained from the error messages are listed below

1. The encrypted form of the username
2. The encrypted form of the password
3. The field name of the password (pass)

An attacker can make use of this information to attack the web application in an even efficient way. Also though the username and password fields are encrypted Illegal/Logical Incorrect Queries attack can be conducted successfully. Learning from errors make the attackers make a successful attack after multiple failures.



*Union query attack.* Conducted on the web application by providing union queries in the user inputs. The below-given username and password is provided on the input fields to perform the union query attack.

Username: Test123' union SELECT \* FROM `login` WHERE user = '123'; --

Password: any Password

The modified SQL query is below

```
SELECT * FROM `login1` WHERE user = 'F•u^n^}6x00?•?\•' and
pass = 'GU?C?~•A?5?p•';
```

The username "Test123' union SELECT \* FROM `login` WHERE user = '123'; --" is completely encrypted and becomes "'F' u'n'^}'6x'O'""?\". The query becomes completely meaningless in SQL query language and hence becomes harmless. Some of the malicious code such as ' , ; , and – are encrypted and becomes some random symbols, so they are ignored.

*Piggy-backed queries.* attack conducted with the same username and password used in the web application by encrypting only the password field. This attack made to drop one of the columns named 'age'. Below given are the username and password.

Username: Test123'; ALTER table `login` drop age; - -

Password: any Password

Expected:

```
SELECT * FROM `login` WHERE user = 'Test123'; ALTER table `login` drop age; --
AND pass = 'any Password';
```

Actual:

```
SELECT * FROM `login` WHERE user = '.?p?C?qk?B?-
<6?:?xq??' AND pass = '?0{|U$yI? ?c';
```

The attempt was unsuccessful because all the 'special characters' and Alter query was encrypted and become meaningless to the SQL language. This method prevents the database and the application from the piggybacked query attack.

*Stored procedures.* attack conducted on the web application by providing the below-given username and password.

Username: Test123'; --

Password: anyPassword

These two user inputs are encrypted using cryptographic algorithms and sent to the stored procedure. Below given is the modified stored procedure.

BEGIN

SET @qry = CONCAT("SELECT \* FROM login",

" WHERE user = '", N?d5E????K, "' AND pass = '", ?0{U\$yI? ??c, """);

PREPARE stmt FROM @qry;

EXECUTE stmt;

END

The steps involved in the executing the stored procedure is listed below

1. The stored procedure created and stored in the database.
2. The input fields obtained from the user.
3. Both the fields are encrypted using the cryptographic algorithm.
4. This encrypted username and password sent to the stored procedure.
5. A stored procedure executed.

6. Based on the number of rows obtained by running the stored procedure, the user is allowed to login into the application.

This encrypted username and password becomes harmless to the database and prevents the application from Stored Procedures attack.

*BLIND Injection.* can be conducted in a web application while both the username and password are encrypted. This attempt is to check whether the first letter of the database name is 'I'.

Username: Test123' AND (BINARY substring(DATABASE(),1,1) = 'I'); --

Password: anyPassword

This attack relies on the responses obtained from the database. Any small information can help an attacker to conduct a successful attack against the application. An actual and expected query which gets executed once after clicking the login.

Expected:

```
SELECT * FROM `login` WHERE user = '123' AND (BINARY
substring(DATABASE(),1,1) = 'I'); -- and pass = 'anyPassword'
```

Actual:

```
SELECT * FROM `login` WHERE user = "h4bG~hu'z'v"
AND pass = 'Nd5EK'
```

The encrypted text becomes harmless to the application. Any malicious code in the form of encrypted version is safe until or otherwise, the attacker knows the cryptographic algorithm and key used. This attack was unsuccessful at this point since all the inputs are encrypted.

*Timing attack.* is conducted to make the database to sleep with the below-given username and password.

Username: Test123' union Select SLEEP(50); --

Password: anyPassword

Figure 49: Timing attack after encrypting both input fields.

The above figure shows the login page of the web application with username as 'Test123' union Select SLEEP(50); --' to conduct a Timing Attack. This attempt is to make the database to sleep for 50 seconds. The time increased by changing the value after sleep in the username field. The actual and modified query is below.

Expected:

```
SELECT * FROM `login` WHERE user = 'Test123' UNION select Sleep(50); -- and pass = 'anyPassword';
```

Actual:

```
SELECT * FROM `login` WHERE user = "h4bG~hu
d5E'z'v" AND pass = 'Nd5E'K'
```

Once after clicking the login button the actual query gets executed and prompts an error saying that 'Username or Password is incorrect.' Encrypting the text can prevent most of the attacks that happen through the user inputs. Encryption is one of the best methods to avoid these SQLIA. Only encryption cannot help, validating the user inputs and best coding practice all together can help to prevent these SQLIA through user inputs.



Figure 50: Unsuccessful timing attack after encrypting both input fields.

The above figure shows the alert message given by web application saying that the credentials provided was invalid.

*Alternate encoding.* attack conducted on the web application by providing the hexadecimal values and making the web application to execute it. This attack is mainly to bypass the validation. Alternate encoding conducted on the web application after encrypting both the username and password. The username has a hexadecimal value to SHUTDOWN, and the password is of any choice.

Username: Test23'; exec(char(Ox73687574646j776e)) -- -'

Password: anyPassword

Expected:

```
SELECT * FROM `login` WHERE user = 'Test23'; exec(char(Ox73687574646a7776e)) --
-" AND pass = 'anyPassword'
```

Actual:

```
SELECT * FROM `login` WHERE user = "h4bG~hu'z'v"
AND pass = 'Nd5EK'
```

Once after clicking the login button the inputs given by the user is encrypted and substituted in place of username and password values in the SQL query. The query compares the data with the details stored in the database (encrypted version). If they match, then the user is allowed else they are pushed out with a message saying 'Invalid Credentials'. Since the authentication process involved only with the encrypted version, the attacker has no way to enter the application.

**Speed test.** Speed test was conducted across all algorithms to come out with a best one to use in the web application. The same plaintext and key are used to test the speed of the algorithms. The micro time was recorded before encryption or decrypting the plain text, once after encrypting or decrypting the plain text the micro time re-recorded. The difference between the two micro times gives the actual time taken by the algorithm to encrypt or decrypt the plain text. The same methodology used across all algorithms and their speeds recorded. The code sample in determining the time taken to encrypt or decrypt the data shown below.

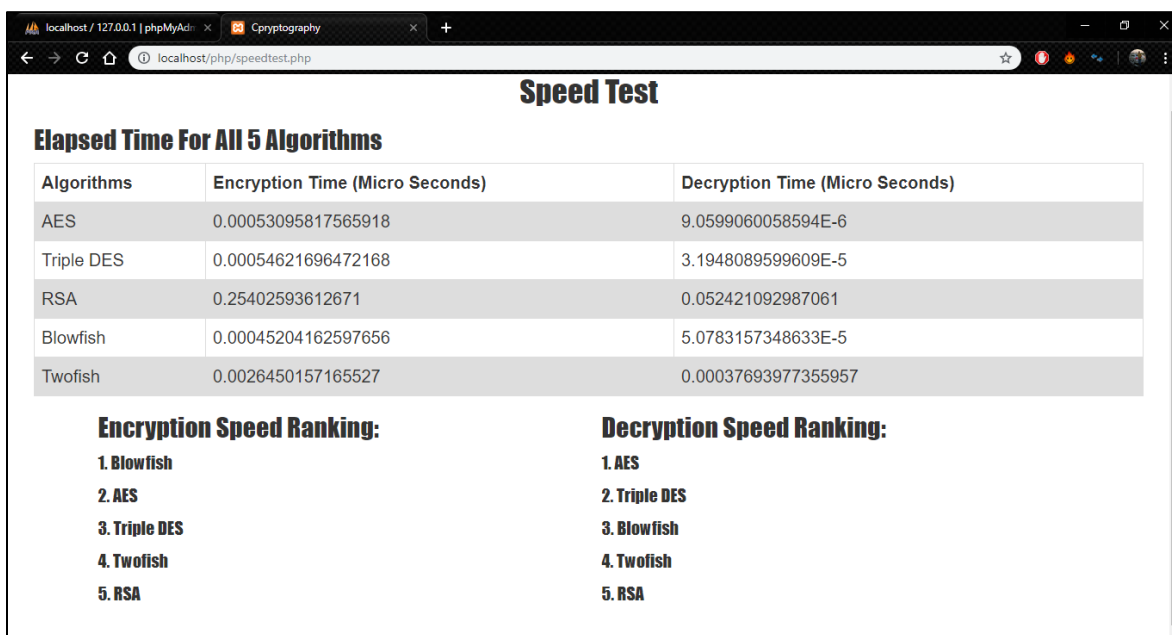
```
$start_3DES = microtime(true); //Micro time was recorded

$des = new Crypt_TripleDES(); //encryption function is called

$des->setKey('abcdefghijklmnopqrstuvwx'); // Key is provided

$cipher_3DES = $des->encrypt($plaintext); // encrypting the plain text
```

```
$elapsed_3DES = microtime(true) - $start_3DES; // Elapsed time is recorded.
```



Algorithms	Encryption Time (Micro Seconds)	Decryption Time (Micro Seconds)
AES	0.00053095817565918	9.0599060058594E-6
Triple DES	0.00054621696472168	3.1948089599609E-5
RSA	0.25402593612671	0.052421092987061
Blowfish	0.00045204162597656	5.0783157348633E-5
Twofish	0.0026450157165527	0.00037693977355957

<b>Encryption Speed Ranking:</b> <b>1. Blowfish</b> <b>2. AES</b> <b>3. Triple DES</b> <b>4. Twofish</b> <b>5. RSA</b>	<b>Decryption Speed Ranking:</b> <b>1. AES</b> <b>2. Triple DES</b> <b>3. Blowfish</b> <b>4. Twofish</b> <b>5. RSA</b>
---	---

Figure 51: Speed test web page.

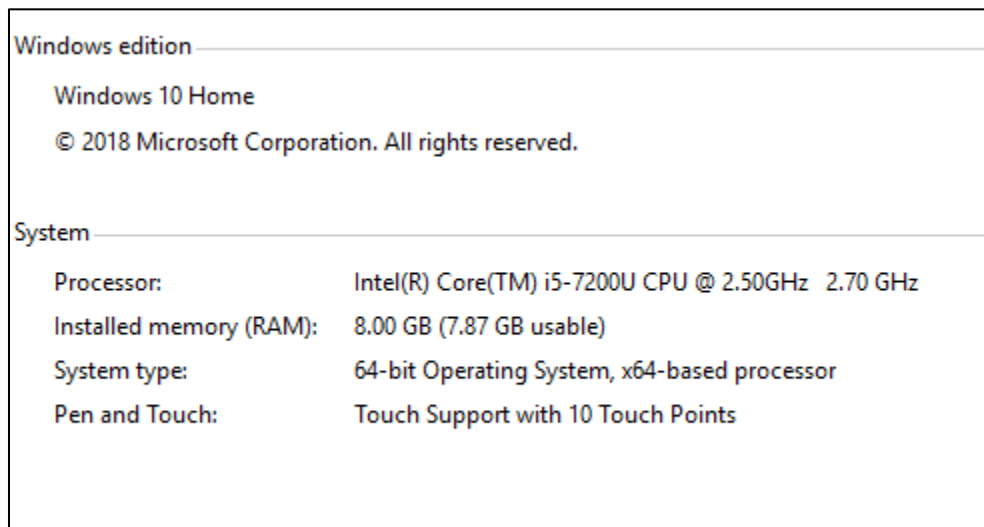
The above figure shows the web page to get the time taken by each algorithm to encrypt and decrypt the data. This page shows the list of algorithms along with the encryption time and decryption time. The time varies each time while encrypting or decrypting the plain text. This page also sorts the time taken by each algorithm and provide a ranking for encryption and decryption.

## Data Analysis

The main tools used to analyze the data is XAMPP server. The front-end coding was done using HTML, CSS, and JS. The server-side programming language used in this study is PHP. All the cryptographic algorithms are in PHP. The PHP code of all algorithms is from the 'phpseclib'. This website provides the source code for various cryptographic algorithms. Those source code libraries used in this project to encrypt and decrypt the data.

The latest versions of the programming languages and software used in this project.

1. HTML 5
2. CSS 3
3. phpMyAdmin 4.8.3
4. 8.0.3-rc-log - MySQL Community Server (GPL)
5. PHP 7.2
6. XAMPP Version: 7.2.3
7. Bootstrap 4



*Figure 52: Configuration.*

The above figure is the configuration of the machine used to conduct this experiment.

## Summary

This chapter provides the detail about the various injection attacks, cryptographic algorithms along with the working and response of the web application during the attacks. The responses were recorded under two conditions, encrypting only the password field and encrypting both the username and password fields. The speed test was also conducted across all



cryptographic algorithms and recorded. Analysis of the data shows that encrypting both the username and password fields will be more secure and use of AES algorithm will be the best option to encrypt and decrypt the data.

## Chapter IV: Data Presentation and Analysis

### Introduction

The results obtained from this study are discussed in this chapter. To come out with the best algorithm a speed test was conducted across all algorithms. Algorithms take the encryption, decryption and overall time is recorded and briefly discussed in this chapter.

### Data Presentation

The data collected during this experiment presented under different sections. The sections are listed below.

1. Application response–Encrypting only password field
2. Application response–Encrypting both username and password fields
3. Each algorithm takes time for encryption and decryption
4. Graphical Representation: Algorithms vs. Time
5. Ranking of algorithms–Encryption speed vs. Decryption speed

**Application response–Encrypting only password field.** The web application designed in such a way that when a user registers under the web application, the password will be encrypted using cryptographic algorithms and saved in the database. When the user attempt to login into the web application, the password provided by them is encrypted and compared with the password already stored in the database. If they match the user can login into the system else, their access denied.

All the SQLIA taken for this study is conducted on this web application when the password field is encrypted. Analyzing the results shows that there was no difference in response of the web application even after encrypting the password field because all the attacks were

made successfully except piggy Backed query and alternate encoding. These two attacks prevented because this application cannot run more than one SQL query at a time. Most of the SQL injection attack conducted on the username field and the password field was ignored using commenting (--, /\*). The responses record after attempting the attacks is below.

Table 5

*Application Response–Encrypting only Password Field*

SQL Injection Attacks	AES	Tipple DES	RSA	Blowfish	Twofish
<b>Tautologies</b>	-	-	-	-	-
<b>Illegal/Logically incorrect queries</b>	-	-	-	-	-
<b>Union Queries</b>	-	-	-	-	-
<b>Piggy Backed Queries</b>	X	X	X	X	X
<b>Stored Procedure</b>	-	-	-	-	-
<b>Inference</b>					
<b>Blind Injection</b>	-	-	-	-	-
<b>Timing Attack</b>	-	-	-	-	-
<b>Alternate Encoding</b>	X	X	X	X	X

X → Prevented

— → Not Prevented

The above table shows the response of the web application after the attacks while the password field was encrypted. The symbols ‘X’ and ‘-’ are used to show that the attack was

prevented and not prevented respectively. All the SQLIA conducted on the web application except Piggy Backed Queries and Alternate Encoding.

**Application response–Encrypting both username and password field.** The web application designed in such a way that both the username and password fields are encrypted and stored in the database. Both the 'username' and 'password' are encrypted and stored in the database. When the user attempt to login into the web application, both the username and password obtained and encrypted and compared with the details in the database. If they match, then they can log in into the application else their access denied.

On analyzing the results after conducting the attacks, most of the attacks prevented by encrypting all the user input fields. The SQL injection attack involves injecting the malicious code to the web application through user inputs. When all the user inputs are encrypted, these malicious codes will also get encrypted and becomes meaningless.

Illegal/Logically incorrect query attack remain unpreventable because of the error messages pushed by the application. Error handling methods can prevent this attack. These error message reveals some of the most critical information about the database and the web application and makes the web application more vulnerable. Pushing the error messages into a separate file is the best method. Continuously monitoring the web application can tell what all the attacks conducted on the web application based on the error message. The responses of the web application recoded after encrypting the username and password field is shown below.

Table 6

*Application Response–Encrypting both Username and Password Fields*

SQL Injection Attacks	AES	Tipple DES	RSA	Blowfish	Twofish
<b>Tautologies</b>	X	X	X	X	X
<b>Illegal/Logically incorrect queries</b>	-	-	-	-	-
<b>Union Queries</b>	X	X	X	X	X
<b>Piggy Backed Queries</b>	X	X	X	X	X
<b>Stored Procedure</b>	X	X	X	X	X
<b>Inference</b>					
<b>Blind Injection</b>	X	X	X	X	X
<b>Timing Attack</b>	X	X	X	X	X
<b>Alternate Encoding</b>	X	X	X	X	X

X → Prevented

— → Not Prevented

The above table shows the response of the web application after the attacks while the username and password fields are encrypted. The symbols ‘X’ and ‘-’ are used to show that the attack was prevented and not prevented respectively. All the SQLIA are successfully prevented except Illegal/Logically incorrect queries attack.

**Time take by each algorithm for encryption and decryption.** The response speed of the web application is also important as security. When the cryptographic algorithms included in encrypting and decrypting the user inputs, the response speed of the web application. Time taken

by each algorithm is recorded to come out with a speediest algorithm. The time is in microseconds. The encryption and decryption time filed separately. Same plain text and key are used across all algorithm except RSA because it has a separate logic for key generation. RSA uses two different keys to encrypt and decrypt the plaintext and ciphertext respectively. This experiment conducted for more than ten times and the average value is recorded and shown.

Table 7

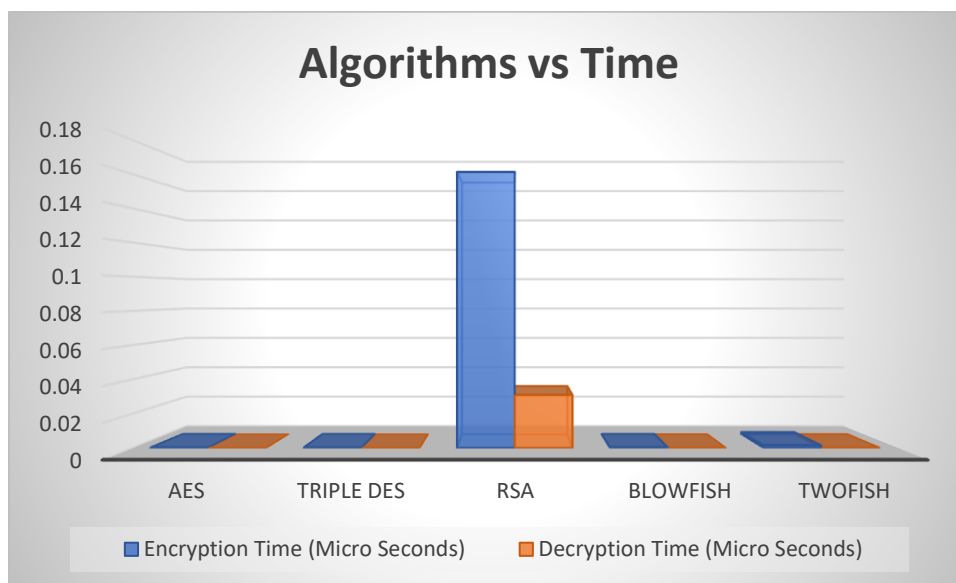
*Time Taken by each Algorithm for Encryption and Decryption*

Algorithms	Encryption Time (Micro Seconds)	Decryption Time (Micro Seconds)
AES	0.000531197	1.09673E-05
Triple DES	0.000544071	3.40939E-05
RSA	0.315055132	0.05095005
Blowfish	0.000361919	9.20296E-05
Twofish	0.003043175	0.000346899

The above table shows the time taken by each algorithm to encrypt and decrypt the plain text. Same plain text and key were used across all algorithms. AES stands in first and second place in decryption and encryption respectively.

**Graphical representation: Algorithms vs. time.** The encryption time is more critical as the data provided by the user was encrypted. To compare the time taken by the algorithms for encryption and decryption is shown in a graphical representation. Algorithms are in the x-axis and the of time in the y-axis. Time for encryption shown in blue and the time taken for the decryption shown in brown. The results show that encryption takes more time than decryption. All the algorithms take more time to encrypt the data. RSA takes more time than the other algorithms since it has two keys (encryption key and decryption key). AES takes very less time

while comparing with the different algorithms and this would be the best algorithm to use in the web application.



*Figure 53:* Graphical representation: Algorithms vs. time.

The above figure is the graphical representation of time taken by each algorithm to encrypt and decrypt the plain text and cipher text respectively. The graph shows that RSA takes more time while comparing with other algorithms due to its complex structure.

**Ranking of algorithms—encryption speed vs. decryption speed.** Based on the encryption and decryption speed the algorithms are ranked from 1 through 5. AES is ranked as first in decryption and second in encryption speed. Blowfish takes the first place in encryption speed, but it was pushed to third place by Triple DES when it comes decryption speed. Twofish and RSA remain in fourth and fifth places for both encryption and decryption speed. The encryption and decryption ranking are listed below.

**Encryption speed ranking:**

1. Blowfish
2. AES
3. Triple DES
4. Twofish
5. RSA

**Decryption speed ranking:**

1. AES
2. Triple DES
3. Blowfish
4. Twofish
5. RSA

AES would be the first preference when it comes to encryption or decryption in the web application. Second preference would be blowfish, and third preference would be Triple DES. The last choices would be between Two fish or RSA.

**Data Analysis**

The data obtained during this experiment analyzed by representing them in the tabular and graphical representation. The analysis of the data collected shows that encrypting all the user input fields will increase the security of the web application. Encrypting only field (password) makes the web application to remain vulnerable.

Choice of the algorithm is most important because the response speed of the web application depends on the cryptographic algorithm. Speed analysis of the algorithms helps us to



come out with the best choice. AES would be the preferred cryptographic algorithm to use in the web application. The reasons are listed below:

1. Symmetric algorithm (requires only one key)
2. Durable design and stands in second place in encryption speed and first place in decryption speed
3. Various options for key length
  - a. 128 - bit Keys
  - b. 192 – bit Keys
  - c. 256 – bit keys

Key size can be chosen based on the need of the web application and design. All these features make AES one of the best algorithms. Even though AES is one of the oldest algorithms, the stability in design, speed, and varied key options makes it more secure.

### **Summary**

This chapter has detailed information about the results obtained during this experiment. Data's represented in graphical and tabular format to make it clearer and more understandable. On a security point of view, all the user input fields must be encrypted to make the application more stable and secure. The choice of the algorithm would be AES since it has a durable design and stands in second place in encryption speed and first place in decryption speed. Being a symmetric algorithm, AES becomes more straightforward to use.

## **Chapter V: Results, Conclusion, and Recommendations**

### **Introduction**

This chapter discusses the results obtained and provides some recommendations in designing a web application from a security point of view. The methodology used in the study is also discussed briefly along with the conclusion. The future work involves designing a website without prompting any error to the end user instead of pushing them into a separate file. Details about the planned study discussed in this chapter.

### **Results**

A website built with the HTML, CSS, JS, MySQL, and PHP. XAMPP server installed and this web application was executed using this server. This website made without any security features. These methods will help to understand the behavior of the application during the SQLIA. All the SQLIA taken in the study attempted on the web application after encrypting the password field, and the response recorded. Later both the username and password field are encrypted, and the responses recorded.

The behavior of the cryptographic algorithms remains same under all conditions during the attack. A speed test conducted across algorithms. AES took very less time while comparing with the other algorithms. RSA takes more time, on the other hand, it has two keys which makes it more secure than the symmetric key algorithms. Algorithms can be chosen based on the need of the security level and application requirement. Based on the results obtained encrypting all the user input field and used the AES algorithm for encryption might help to secure the web application.

**Study Question**

1. What is the primary classification of SQL Injection attack?

Order-wise, Blind and Against database are the three major types of SQLIA. This research focused on the SQLIA – Against database.

2. Why is web application security an important issue?

Web applications are overgrowing. Many businesses really on web application and their customer database. Any damage caused to the web application can make a massive loss to business and exploitation of their customer information.

3. What is background information necessary to understand web application security?

SQL injection is the most common attack on the web application. Basic understanding of attacks on the web application is necessary to understand web security.

4. What are the effects of a SQL injection attack?

A successful SQL injection attack can destroy the database and exploit the information stored in the database. These attacks may cause a massive loss to the business.

5. What is being currently done to secure web application?

In most of the web application, the password is hashed and stored in the database. This method cannot sure the web application all attacks.

6. Based on the research, what are the thoughts about web application security from SQLIA and why?

SQLIA is one of most powerful attack against the web application. A successful attack can do massive damage. Cryptography is one of the best methods to stop SQLIA with less effort.

7. Based on my research, what should be done to secure web application from SQLIA?

Based on my research, to prevent web application SQLIA all the user input fields must be encrypted, and error handling must be perfect so that attackers cannot view any error messages.

## **Conclusion**

SQLIA is one of the most potent attacks on the web application. The powerful cryptographic tool is to secure the application from SQLIA. Encrypting all the user input field will convert the malicious codes into meaningless characters. This method will prevent the web application from injecting malicious code into the web application. Including algorithms into the web application can slow down the response speed. In that case, AES would be the best option since it is stable and takes very less time to decrypt the data. Error handling plays a vital role in security. Moving the error or warning messages into a separate file or a table in the database can prevent the web application from showing them on the screen. Cryptography alone cannot stop the injection attack proper coding; proper error handling also plays an essential role in securing the web application. Stopping an attacker is not possible instead slow them down by closing as many doors as possible.

## **Future Work**

The actual behavior of the application pushes all the error message on the web page. These errors will help the attackers to understand some of the critical information about the web

application. To prevent the web application from showing warning or error messages from SQL and PHP, move them to a file or table in the database.

**Moving all SQL errors into a table.** SQL prompt an error message when there is a syntax or any other error. These could be and moved to a PHP variable. All the details such as error message, date and time are pushed into a table. The code used to obtain those values is below.

```
$error = mysqli_error($conn); // get the error message prompted from SQL

$error = str_replace("'", "", $error);

$date = date("Y-m-d"); //Obtain date

$time = date("h:i:sa"); //Obtain time

if ($error != "") //move date, time and error into a table
{
    $query = "insert into error (message,date,time) values('$error','$date','$time)";
    $query_run = mysqli_query($conn,$query);
}
```

The table after inserting the details is below. Date and time the error raised, and the error messages inserted into the table.

id	date	time	message
16	2018-10-15	06:19:15am	You have an error in your SQL syntax; check the ma...
17	2018-10-15	06:19:27am	You have an error in your SQL syntax; check the ma...
18	2018-10-16	05:13:20am	You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'Në"d5E0,,0ÉK' and pass = 'Qÿ# 'g2'' at line 1]

Figure 54: Moving all SQL errors into a table.

The above figure shows the table which records the error messages pushed by the database server. This table can be continuously monitored to check whether any SQL injection happened. This method will help to increase the stability and security of the web application.

**Moving all error messages and warnings from PHP into a file.** Whenever there is a malfunction, PHP prompts some error or warning messages on the web page. This help attackers to gain knowledge about the application. The error messages can be moved to a file. The code involved in writing the error log is below:

```
error_reporting(E_ALL); // Error engine - always ON!

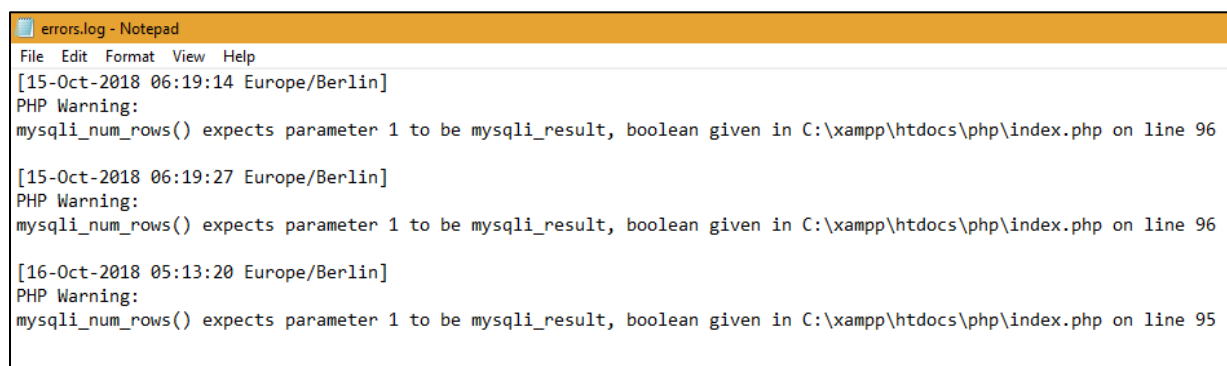
ini_set('display_errors', FALSE); // Error display - OFF in production env or real server

ini_set('log_errors', TRUE); // Error logging

ini_set('error_log', 'error/errors.log'); // Logging file

ini_set('log_errors_max_len', 1024); // Logging file size
```

This code will make the application to write all the error or warning messages into a file.



*Figure 55: Moving all error messages and warnings from PHP into a file.*

The above figure shows the file which holds all the error messages pushed by the PHP script. Continuously monitoring the log file will help us to improve the application and any attempt to attack can also be monitored.

## References

- Anitha.V, Lakshmi.A, S., Revathi.M, & Selvi.K. (2014). Detecting various SQL injection vulnerabilities using string matching and LCS method. *Sixth International Conference on Advanced Computing (ICoAC)*, 237-241(6). doi:10.1109/ICoAC.2014.7229717
- "Asymmetric vs Symmetric encryption differences". (n.d.). Retrieved from SSL2BUY: <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>.
- Basharat, I., Azam, F., & Muzaffar, A. W. (2012). Database security and encryption: A survey study. *International Journal of Computer Applications* (0975–888).
- "Blowfish". (n.d.). Retrieved from mcmaster: <http://wiki.cas.mcmaster.ca/index.php/Blowfish>.
- Deepika, N. S. (2015). Database security: Threats and security techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*.
- "DES Expansion". (n.d.). Retrieved from wikimedia: [https://commons.wikimedia.org/wiki/File:Des\\_expansion.svg](https://commons.wikimedia.org/wiki/File:Des_expansion.svg).
- "E-commerce statistics". (n.d.). Retrieved from Ready cloud: <https://www.readycloud.com/info/95-ecommerce-statistics-to-know-in-2017>.
- "Encryption". (n.d.). Retrieved from skyhighnetworks: <https://www.skyhighnetworks.com/cloud-security-university/tokenization-vs-encryption/>.
- Gaithuru, J. N., Bakhtiari, M., Salleh, M., & Muteb, A. M. (2015). A comprehensive literature review of asymmetric key cryptography algorithms for establishment of the existing gap. *2015 9th Malaysian Software Engineering Conference (MySEC)*.

- Jie Wang, R. C.-W., Whitley, J. N., & Parish, D. J. (n.d.). Augmented attack tree modeling of SQL injection attack. *IEEE*.
- Kumar, P., & Pateriya, R. (2012). A survey on SQL injection attacks, detection and prevention techniques. *IEEE-20180*.
- Kumar, S. S., Prabhu, S. A., R.Durga, & Jeevitha, S. (2016). A survey on various asymmetric algorithms. *International Research Journal of Advanced Engineering and Science*.
- Mewada, S., Sharma, P., & Gautam, S. S. (2016). Exploration of efficient symmetric algorithms. *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference*.
- Mukherjee, S., Bora, S., Sen, P., & Pradhan, C. (2015). SQL injection: A sample review. *6th ICCCNT*.
- Mushtaq, M. F., Jamel, S., Disina, A. H., Pindar, Z. A., Shakir, N. S., & Deris, M. M. (2017). A survey on the cryptographic encryption algorithm. *International Journal of Advanced Computer Science and Applications*.
- Mushtaque, M. A., Dhiman, H., Hussain, S., & Maheshwari, S. (2014). Evaluation of DES, TDES, AES, blowfish and two fish encryption algorithm: Based on space complexity. *International Journal of Engineering Research & Technology (IJERT)*.
- Nadeem, A., & Javed, D. M. (n.d.). *A performance comparision of data encryption algorithm*.
- Panda, M. (2016). Performance analysis of encryption algorithms for security. *International conference on Signal Processing, Communication, Power and Embedded System*.



- Rani, D. R., Kumar, B., Rao, L. R., Jagadish, V., & M.Pradeep. (2012). Web security by preventing SQL injection using encryption in stored procedures. *International Journal of Computer Science and Information Technologies*.
- Rivest, R. L., Shamir, A., & Adelman, L.(1978). A method for obtaining digital signature and public –key cryptosystems. *Commun. ACM*, 21, 120-126.
- (PDF) *Impact of security over System performance*. Available from:  
[https://www.researchgate.net/publication/275155958\\_Impact\\_of\\_security\\_over\\_System\\_performance](https://www.researchgate.net/publication/275155958_Impact_of_security_over_System_performance) [accessed Dec 09 2018].
- Roy, S., & Sairam, A. K. (July 2011). Detecting and defeating SQL injection attacks. *International Journal of Information and Electronics Engineering*.
- S, D. P., Kumar, D. S., & Rahman, D. M. (May 2015). Preventing SQL injection attacks using cryptography methods . *International Journal of Scientific Research Engineering & Technology (IJSRET)*, ISSN 2278–0882.
- Sadeghia, A., Zamani, M., & Abdullah, S. M. (2013). A taxonomy of SQL injection attacks. *International Conference on Informatics and Creative Multimedia*.
- Singh, P., & Kaur, D. K. (n.d.). Database security using encryption. *2015 1st International Conference on Futuristic Trend in Computational Analysis and Knowledge Management*.
- Sonakshi, Kumar, R., & Gopal, G. (2016). Prevention of SQL injection Attacks using RC4 and blowfish encryption techniques. *International Journal of Engineering Research & Technology (IJERT)*.
- Sood, M., & Singh, S. (2017). SQL injection prevention technique using encryption. *International Journal of Advanced Computational Engineering and Networking*.

"SQL Injection". (n.d.). Retrieved from W3resource: <https://www.w3resource.com/sql/sql-injection/sql-injection.php>.

Stallings, W. (n.d.). *Cryptograpgy and network security*.

"Triple DES". (n.d.). Retrieved from Tutorial point: [https://www.tutorialspoint.com/cryptography/triple\\_des.htm](https://www.tutorialspoint.com/cryptography/triple_des.htm).

"twofish". (n.d.). Retrieved from slideshar: <https://www.slideshare.net/ghanbarianm/twofish>.

"Web Application Attack Statistics". (n.d.). Retrieved from <https://malware.news/t/web-application-attack-statistics-q2-2017/15110>.

"Web Application Attack Statistics". (2017, Sep). Retrieved from Malware News: <https://malware.news/t/web-application-attack-statistics-q2-2017/15110>.

Wigginton, J. (2008). *Phpseclib*. Retrieved from <http://phpseclib.sourceforge.net/>: <http://phpseclib.sourceforge.net/>.