Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувачка кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«14» червня 2022р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього ступеня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Механізм безпечного зберігання паролів із використанням алгоритму Blowfish»

Виконавець: студентка IV курсу, групи КБ-41

_____ **Тетяна ЮЖАКОВА**
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Андрій ФЕСЕНКО	

Нормоконтроль	Олександр ТОРОШАНКО	

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувачка кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентці **КБ-41** **Южаковій Тетяні Валеріївні**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Механізм безпечного зберігання паролів із використанням алгоритму Blowfish

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Структура веб-додатку, технології для розробки веб-додатку, алгоритм шифрування Blowfish, блокові алгоритми шифрування.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Проблематика безпечного зберігання паролів, способи зберігання паролів, алгоритм шифрування Blowfish, блокові алгоритми шифрування, хеш-функції для хешування паролів, існуючі реалізації алгоритму Blowfish, розробка веб-додатку із аутентифікацією користувачів.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність є створення веб-додатку, в якому облікові дані

користувачів (логін та пароль) безпечно зберігаються у базі даних, при цьому пароль шифрується алгоритмом Blowfish.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29.10.2021 року

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ініціали, прізвище)

Завдання прийняв до виконання _____ (підпис) Тетяна ЮЖАКОВА (ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Аналіз літератури про алгоритм Blowfish	29.10.2021 – 10.02.2022	*виконано*
2	Розгляд проблематики безпечного зберігання паролів	11.02.2022 – 20.02.2022	*виконано*
3	Розгляд способів безпечного зберігання паролів	21.02.2022 – 28.02.2022	*виконано*
4	Розгляд структури алгоритму Blowfish	01.03.2022 – 20.03.2022	*виконано*
5	Розгляд аналогів алгоритму Blowfish	21.03.2022 – 10.04.2022	*виконано*
6	Аналіз безпеки алгоритму Blowfish	11.04.2022 – 17.04.2022	*виконано*
7	Обрання мови програмування для реалізації технічного завдання	18.04.2022 – 25.04.2022	*виконано*
8	Створення веб-додатку	26.04.2022 – 18.05.2022	*виконано*
9	Оформлення дипломної роботи	19.05.2022 – 05.06.2022	*виконано*
10	Підготовка до захисту дипломної роботи	06.06.2022 – 13.06.2022	*виконано*

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ініціали, прізвище)

Завдання прийняв до виконання _____ (підпис) Тетяна ЮЖАКОВА (ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2022 року

УДК 004.056.5

РЕФЕРАТ

Пояснювальна записка: вступ, три розділи, висновки, список використаних джерел та два додатки. Основний текст міститься на 65 сторінках, включає в себе зміст, вступ, три розділи, висновки та список використаних джерел. Дипломна робота містить два додатка із загальною кількістю сторінок – 11. Пояснювальна записка містить 24 рисунка.

Метою роботи є реалізація механізму безпечного зберігання паролів із використанням алгоритму Blowfish.

Для досягнення мети необхідно виконати такі завдання:

- дослідити проблематику безпечного зберігання паролів;
- проаналізувати безпеку блокових алгоритмів шифрування та визначити оптимальний алгоритм для шифрування паролів;
- створити сайт із аутентифікацією користувачів та шифруванням паролів із використанням алгоритму Blowfish.

Об'єктом дослідження є процес безпечного зберігання паролів із використанням алгоритму шифрування Blowfish.

Предметом дослідження є блокові алгоритми шифрування та їх використання для безпечного зберігання паролів.

Практичною цінністю даної роботи є створення веб-додатку, в якому облікові дані користувачів (логін та пароль) безпечно зберігаються у базі даних завдяки шифрування паролю алгоритмом Blowfish.

Методи дослідження: порівняння та аналіз.

Ключові слова: шифрування, паролі, безпечне зберігання паролів, алгоритм Blowfish, шифрування паролів, алгоритми симетричного шифрування, хеш-функції, bcrypt, злом паролів.

Розроблено: веб-додаток, в якому є форма реєстрації та входу в інтернет-магазин, дані шифруються за допомогою алгоритму Blowfish.

ЗМІСТ

ВСТУП……... 7
РОЗДІЛ 1 ПРОБЛЕМАТИКА БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ…….. 9
1.1 Використання паролів у різних сервісах…… 9
1.2 Політика паролів та способи їх захисту…….. 9
1.3 Дослідження проблеми зберігання паролів ……... 14
1.4 Статистика злочинності, пов’язаної з викраденням паролів…… 15
1.5 Постановка завдання……. 15
Висновки за розділом 1…….. 16
РОЗДІЛ 2 ОГЛЯД АЛГОРИТМУ BLOWFISH ТА ЙОГО АНАЛОГІВ…….. 17
2.1 Історія створення Blowfish……... 17
2.2 Структура алгоритму шифрування Blowfish……. 19
2.3 Міні версії алгоритму Blowfish……... 25
2.4 Математичні операції в Blowfish……. 26
2.5 Стійкість алгоритму…….. 27
2.6 Порівняння алгоритму Blowfish з його аналогами……..... 29
2.7 Порівняння використання Blowfish та хеш-функцій для зберігання паролів……..... 31
2.8 Використання алгоритму Blowfish…….. 32
Висновки за розділом 2…….. 33
РОЗДІЛ 3 РОЗРОБКА МЕХАНІЗМУ БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ ІЗ ВИКОРИСТАННЯМ АЛГОРИТМУ BLOWFISH…….. 34
3.1 Огляд наявних реалізацій алгоритму……... 34
3.2 Розробка механізму для збереження паролів користувачів при реєстрації на сайті…….. 39
3.2.1 Архітектура веб-додатку…… 40
3.2.2 Програмна реалізація веб-додатку……. 42
Висновки за розділом 3…….. 58

ВИСНОВКИ……..... 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…… 60
ДОДАТКИ……..... 64
ДОДАТОК А ……. 64
ДОДАТОК Б……... 67

ВСТУП

Інтернет дозволяє підтримувати зв’язок на великій відстані, дозволяє отримувати будь-яку інформацію, завдяки інтернету працює велика кількість веб-додатків, мобільних додатків. Проте, використання різних додатків може іноді призводити до витоку конфіденційної інформації, якою, не завжди хочеться і потрібно ділитись з іншими. Витік інформації призводить іноді до катастрофічних наслідків:

- руйнування компанії через витік інформації, яка завдає шкоди по репутації;
- руйнування особистого життя людини через витік інформації, фото особистості, які завдають шкоди особі;
- втрата коштів, якщо відбудеться витік інформації про активи компанії, про її клієнтів;
- втрата активів компанії, якщо відбудеться витік інформації або її видалення з робочих комп’ютерів через шкідливе програмне забезпечення.

Це лише частина наслідків. Для того, щоб їх запобігти, необхідно дотримуватись правил кібербезпеки.

Найчастіше на сайтах унеможливлюють або обмежують користування сервісами, якщо користувач не зареєструється в системі. А так як мало людей задумується про безпеку при користуванні інтернет-послугами, то зазвичай використовують одні і ті самі облікові дані для багатьох сервісів. Це може призвести до поганих наслідків, якщо хакери отримають пароль та логін від одного сервісу, то можуть отримати доступ і до інших сервісів. Для того, щоб зменшити ризики компроментації даних в інтернеті, необхідно або створювати надійні паролі і запам’ятовувати їх, або зберігати їх в менеджері паролів.

Створення надійних паролів це лише перший крок до забезпечення безпеки. Наступним кроком є надійне зберігання паролів. Зазвичай, кожен веб-сервіс забезпечує шифрування або хешування паролів. Існує чимало алгоритмів

шифрування та хеш-функцій. Переваги та недоліки першого та другого способів зберігання паролів буде розглянуто в дипломній роботі.

Шифрування паролів алгоритмом Blowfish забезпечує надійне зберігання даних, тому що Blowfish доволі довгий час вважається надійним алгоритмом шифрування. Хоч наразі існує багато аналогів алгоритму, проте він ще використовується або в стандартному вигляді алгоритму, або в модифікованому.

Метою дипломної роботи є розробка механізму безпечного зберігання паролів із використанням алгоритму шифрування Blowfish. Дана тема є актуальною, тому що використання веб-сервісів наразі є доволі поширеним явищем, а оскільки дані користувачів необхідно безпечно зберігати, запобігати компроментації даних, то для цього і необхідно використовувати шифрування даних. Лише комплексні заходи допоможуть захищати дані, а шифрування є однією із складових захисту.

РОЗДІЛ 1
ПРОБЛЕМАТИКА БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ

1.1 Використання паролів у різних сервісах

В сучасному світі інформаційні технології стрімко розвиваються. Повноцінне існування стало неможливим без використання різних гаджетів для зв’язку та спілкування, сервісів, в яких ми можемо отримати потрібні нам послуги для забезпечення потрібного рівня проживання. Зазвичай, для отримання різних послуг на веб-сайтах, в мобільних або десктопних додатках необхідно ввести щонайменше пароль і логін для реєстрації.

Оскільки, інформація про те, які послуги отримує користувач або які дані зберігає в додатках (наприклад, банківські картки, інформація про покупки на веб-сайтах, інформація про оплату послуг та інші) повинна бути конфіденційною, необхідно захищати доступ до неї, тобто потрібно, щоб доступ до неї був лише у власника даних та в адміністраторів сервісів. Для цього необхідно, щоб користувач, при вході в систему, вводив логін і пароль, який знає лише він, причому пароль має бути надійним, він не має містити дані про ім’я користувача чи прізвище, дату народження або часткову назву електронної пошти, яка використовується як логін при реєстрації. Пароль має відповідати політиці паролів, яка повинна бути створена в кожному сервісі. Якщо така політика існує, то при реєстрації, якщо пароль слабкий, то система повідомляє про це користувачеві, і якщо користувач не створить сильний пароль, то не зареєструється в системі.

1.2 Політика паролів та способи їх захисту

В кожній установі, в якій впроваджена інформаційна система, необхідно створити політику паролів. В даній політиці повинні бути вказані вимоги до паролів, задля того, щоб зменшити ризики розкриття конфіденційної інформації, у разі злому

слабкого пароля будь-якого працівника. Для прикладу політика паролів має виглядати так [1]:

- пароль не повинен включати в собі назву облікового запису, електронну пошту або будь-яку їх частину, а також загальні слова, які часто зустрічаються в технічній термінології (user, password та інші);
- пароль повинен складатися, як мінімум, із восьми літер, цифр чи символів;
- пароль має містити символи з-поміж наступних чотирьох категорій:

1. великі літери, такі як A-Z;
2. малі літери, такі як a-z;
3. цифри;
4. символи, такі як !, $, +, :, #, % та інші.

З метою забезпечення інформаційної безпеки, символи пароля, що вводиться, не повинні зображатися на екрані у явному вигляді (рис.1.1) [1].

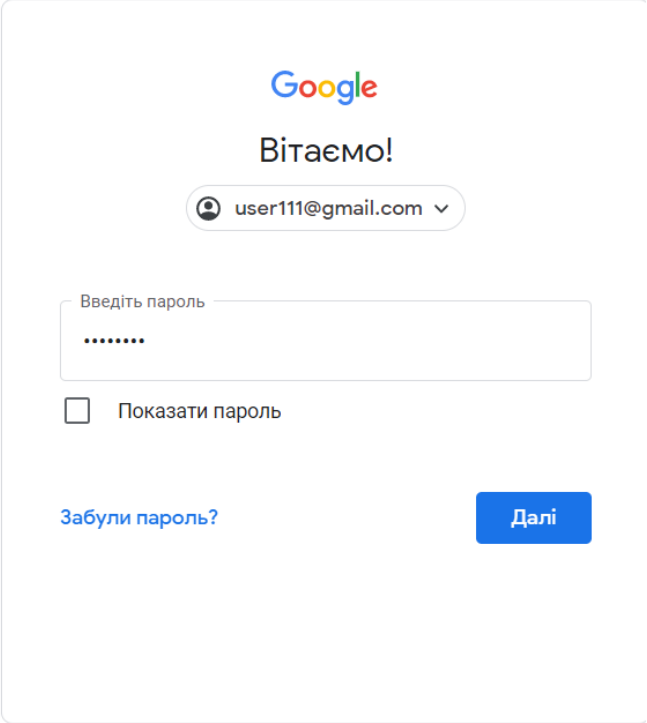

Рисунок 1.1 - Відображення пароля у вигляді кружечків

Оскільки при вході в систему зареєстровані користувачі вводять логін та пароль, системі необхідно порівняти їх з існуючими даними, тобто дані для входу порівнюються з даними з бази даних, де зберігаються усі дані користувачів. Паролі

можуть зберігатись у вигляді хешу або у зашифрованому вигляді, ще для підвищення стійкості паролів може використовуватись метод засолювання [2].

Шифрування призначено для кодування паролів, тексту, файлів, щоб ніхто, крім власників цих даних не мав доступ. Лише власники мають ключ, яким можна розшифрувати дані [2].

Хешування це метод зміни даних за допомогою обчислень односторонньої функції. Завдяки цим обчисленням, хешовані дані неможливо перетворити назад, у звичайний вигляд. Це надає особливі переваги для таких видів захисту та ідентифікації, як цифрові підписи, багатофакторна аутентифікація [2].

Соління даних - це процес, під час якого до даних додається сіль (набір випадкових даних), а потім дані разом із сіллю шифруються або хешуються. Цей процес підвищує стійкість зашифрованих/хешованих даних [2].

Шифрування – це процес перетворення даних за допомогою різноманітних алгоритмів шифрування. Ці алгоритми називаються шифрами. Відмінність шифрування і хешування в тому, що зашифровані дані можна повернути в їх початковий вигляд, використавши ключ, відповідно до якого відбувалось шифрування, а хешовані дані неможливо перетворити в початковий вигляд, тому що хешування це односторонній процес. Ця відмінність потрібна, наприклад при передачі даних, коли один користувач шифрує дані публічним ключем, потім передає ці дані через середовище передачі інформації, а та особа, яка отримала ці дані, розшифровує дані за допомогою приватного ключа. Після процедури шифрування будь-яким алгоритмом шифрування, дані є захищеними, лише власник може їх розшифрувати власним ключем. Є два різновиди шифрування: симетричне, при якому дані зашифровуються і розшифровуються за допомогою одного і того самого ключа, а також асиметричне, як було згадано в прикладі вище, в якому дані шифруються публічним ключем і розшифровуються приватним ключем [2]. Шифр Blowfish, який розглядається в дипломній роботі це симетричний блоковий шифр [3].

Якщо розглянути хешування, то криптографічні хеш-функції це, як уже було згадано, односторонні функції, в результаті їх виконання завжди на вихід буде

виходити хеш відповідної довжини. Довжина хешу визначається типом алгоритму хешування, їх існує немало, і кожен з них дає на вихід рядок, обчислений саме за цим алгоритмом [2]. Корисні функції хешування[2]:

- односторонні функції – дані неможливо розшифрувати;
- якщо на вхід хеш-функції дати два рази ідентичні дані, то на виході будуть ідентичні хеш-значення;
- якщо на вхід давати різні значення, то дуже мала ймовірність отримати однаковий хеш;
- при зміні навіть одного символу результат односторонньої функції уже буде інший, буде сильно відрізнятись від того результату, коли дані ще не були змінені.

Зазвичай результати хеш-функцій використовують для порівняння фрагментів даних, наприклад паролів, коли користувач при вході на сайт вводить пароль, він хешується і результат порівнюється з тим, який уже є в базі даних, якщо хеші співпадають, то вхід успішний. Також хеші використовуються для перевірки цілісності файлу, наприклад офіційний розробник на сайті вказує хеш файлу, якщо користувач скачує файл і робить хеш, і даний хеш співпадає з тим, який надав розробник, то файл не був модифікований, а якщо вони не співпадають, то хтось вніс зміни в файл, можливо було додано зловмисні частини коду[4]. Незважаючи на те, що криптографічні хеш-функції створюють важкі для вгадування контрольні суми, користувачі все одно повинні використовувати складний пароль для всіх своїх онлайн і локальних облікових записів користувачів, оскільки при проведенні атаки по словнику перебираються популярні паролі, і якщо при генеруванні хешу популярних паролів буде співпадіння згенерованого хешу з хешем пароля користувача, то зловмисник легко отримає доступ до його даних [4].

Метод засолення це додавання випадкових даних до паролю, перш ніж він буде переданий на вхід хеш-функції. Як уже було згадано, якщо користувачі використовують слабкі паролі для сервісів, додатків чи соціальних мереж, то ці паролі легко дізнатись завдяки проведенню атаки Brute force. Під час цієї атаки створюється хеш паролю і порівнюється з хешем паролю користувача. Щоб

зменшити успіхи цієї атаки необхідно до паролю додавати випадкові дані і вже тоді створювати хеш. Тоді уже якщо будуть перебирати паролі, то мала ймовірність знайти пароль «1234gposjhdubw», ніж пароль «1234». В даному випадку сіль це «gposjhdubw», якщо її додавати до паролів, то хеш паролю з сіллю буде інший і таким чином хеші будуть безпечніше зберігатись [2]. Ще один приклад можна побачити на рис. 1.2 - 1.3, там показано різницю перетворення в хеш пароля з сіллю та без солі.

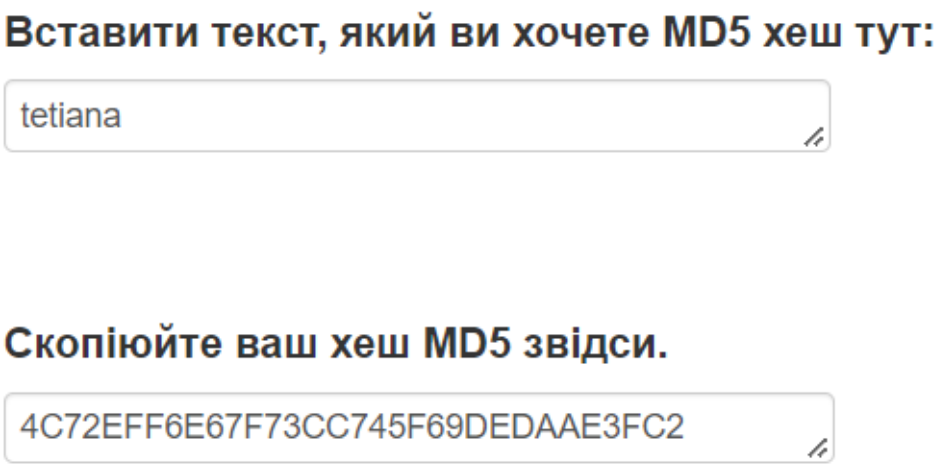

Рисунок 1.2 - Хешування паролю без додавання солі

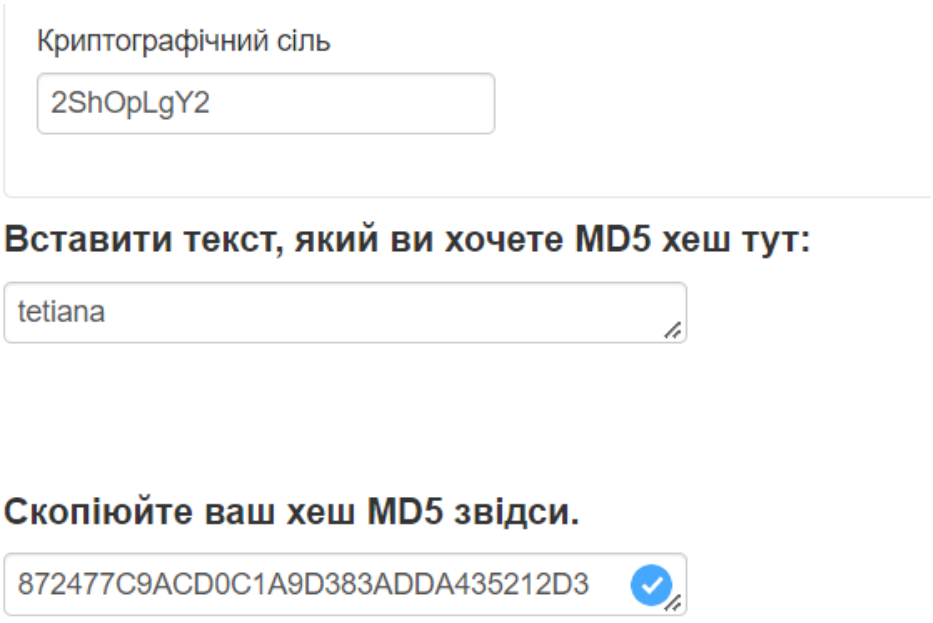

Рисунок 1.3 - Хешування паролю із додавання криптографічної солі

1.3 Дослідження проблеми зберігання паролів

Проблему безпечного зберігання паролів досліджували такі вчені, серед яких Savita Devidas Patil [5], Ashish.T. Bhole [6], Chirag Singh Sisodia та Aparajiit Shrivastava [7]. Вчені у своїх роботах звертають увагу на те, що для того, щоб отримати приватні дані, необхідно пройти аутентифікацію [5]. Аутентифікація зазвичай однофакторна, з використанням логіну і паролю, інколи буває двофакторна, наприклад, отримання смс для підтвердження входу або аутентифікація за допомогою біометричних даних [8]. Сервісів існує доволі багато, і для того, щоб забезпечити конфіденційність даних користувача, користувач має придумати та встановити сильний пароль для входу в сервіси. Проте часто люди не хочуть запам’ятовувати всі паролі і для всіх сервісів використовують один, часто ненадійний, пароль. Отже, зростає ймовірність того, що якщо хакери під час проведення атаки на один із тих сервісів отримають пароль від одного сервісу, який використовує користувач, і вони зможуть отримати доступ до усіх акаунтів користувача [2].

Для того, щоб зменшити ризики розкриття конфіденційних даних можна використовувати менеджери паролів або забезпечити безпечне зберігання паролів завдяки процедурі шифрування або хешування. Для того, щоб використовувати менеджер паролів для збереження паролів, необхідно буде запам’ятати один, але сильний пароль, який повинен складатись з щонайменше 12 літер латинського алфавіту (великі та малі літери), цифр, символів [9]. Тобто використовуючи один пароль можна буде зайти до менеджера паролів і відшукати потрібний пароль до акаунта або сервісу. Це дуже зручний та безпечний спосіб зберігання паролів.

Інший спосіб забезпечити безпеку даних це зберігати паролі у базах даних в зашифрованому вигляді. Зазвичай в базах даних знаходяться паролі у вигляді хешу, тобто використовуються хеш-функції для перетворення паролів, проте також можна використовувати шифри. Якщо шифрувати пароль за допомогою шифру Blowfish, то виходить також певна комбінація цифр та літер, проте, на відміну від хешів, для того, щоб перетворити цей шифр в початкову форму, необхідно знати ключ, який,

через особливості Blowfish, генерується для кожного шифрування, або також може міститись у самому паролі. Тому, якщо спробувати провести атаку brute force, то на це можна витратити близько 7 років [10].

1.4 Статистика злочинності, пов’язаної з викраденням паролів

Згідно з даними компанії Verizon, у 81% випадків викрадення, модифікація, розкриття даних виникають через вкрадені або ненадійні паролі. Існує чимало ризиків, пов'язаних зі слабкою аутентифікацією за допомогою паролю, для зменшення цих ризиків необхідно впроваджувати багатофакторну аутентифікацію, а також створювати лише надійні паролі для кожного нового облікового запису. Багатофакторна аутентифікація доволі ефективна в забезпеченні безпеки, тому що використовуються інші фактори для аутентифікації, окрім паролів, наприклад, відбиток пальця, розпізнавання обличчя, смс-повідомлення із кодом та інші. Не дивлячись на ці переваги багатофакторної аутентифікації, якщо користувачі постійно, декілька разів на день проходять через цю процедуру, то це може негативно позначитися ефективності користувача, тому що це не дуже зручно постійно проходити аутентифікацію, проте, це забезпечить безпеку даних [11]. Якщо забезпечити безпечне зберігання паролів, створювати сильні паролі, не вводити паролі на фішингових сайтах, не виказувати свої паролі інших особам (соціальна інженерія), то зловмисники не зможуть зламати облікові записи та скомпрометувати дані.

1.5 Постановка завдання

Проаналізувавши дані про проблеми, пов’язані із викраденням паролів та використанням слабких паролів, основним завданням написання дипломної роботи є забезпечити зберігання паролів користувача після реєстрації на сайті інтернет-магазину. Паролі будуть шифруватись за допомогою криптографічного алгоритму

шифрування Blowfish, до паролів також повинна додаватись сіль для зменшення ризику компрометації зашифрованих даних.

Висновки за розділом 1

В даному розділі було проаналізовано основні проблеми, пов’язані зі створенням паролів (від того, наскільки пароль сильний, залежить його стійкість до зламу) та їх безпечним зберіганням. Було описано основні способи захисту паролів, такі як перетворення паролю в хеш (хешування), перетворення паролю в шифр (шифрування), та комбінація солі з хешом (соління). Вияснено, що тема безпечного зберігання паролів зараз дуже актуальна, тому що, велика кількість випадків розкриття конфіденційних даних відбувається через викрадення паролів, а викрадення паролів, зазвичай, відбувається або через небезпечне зберігання або через слабкі паролі, які легко дізнатись через атаку brute force.

РОЗДІЛ 2
ОГЛЯД АЛГОРИТМУ BLOWFISH ТА ЙОГО АНАЛОГІВ

2.1 Історія створення Blowfish

В 1993 році криптограф та спеціаліст з кібербезпеки (комп'ютерної безпеки) Брюс Шнайер (рис. 2.1) створив алгоритм шифрування Blowfish. На той час, криптограф уже видав свою книгу «Прикладна криптографія», в якій чітко описав суть алгоритмів, порядок їх використання та де їх можна використовувати [12].

Криптограф в своїй статті 1995 року «Алгоритм шифрування Blowfish — рік потому» вказував, що прийшла необхідність змінити алгоритм DES, тому що він уже 19 років використовувався, і довжина ключа, яка використовується в DES, не може забезпечити безпеку даних. Хоч протягом 20 років криптоаналізу DES, DES все ще використовували навіть спецслужби, проте довіра до алгоритму падала, тому прийшов час замінити DES алгоритмом Blowfish. Були ще декілька алгоритмів, які були здатні замінити DES, наприклад, IDEA, Triple-DES, RC4 та SAFER, проте вони в той час не були настільки поширені та популярні, як алгоритм Blowfish [13].

Рисунок 2.1 - Брюс Шнайер – автор алгоритму Blowfish [14]

Алгоритм Blowfish – це симетричний алгоритм шифрування. Його було створено, щоб замінити менш ефективні алгоритми, такі як DES та IDEA. Blowfish, у порівнянні з цими алгоритмами є безплатний, незапатентований та в рази швидший. Будь-хто може взяти алгоритм та використати його у своїх роботах, проєктах, не потрібно при цьому купувати ліцензію.

Сам алгоритм Blowfish вперше був продемонстрований в Кембриджі на лекції із шифрування, а наступного року був надрукований в журналі «Dr. Dobb's Journal» [15].

Шнайєр був одним із перших, хто запропонував алгоритм для шифрування конфіденційних даних. В 1990-ті роки Інтернетом починало користуватись все більше і більше людей. Спілкування, передачу і отримання даних потрібно було захистити, спеціалісти з безпеки вирішили шифрувати дані, так як це один із найефективніших методів захисту. В нагоді їм став алгоритм шифрування – Blowfish, який дозволяв перетворювати дані в зашифровану послідовність символів, а потім розшифровувати за допомогою ключа і перетворювати послідовність символів в початкові дані [16].

На сьогодні, не так багато атак на даний алгоритм було досліджено, проте, відповідно до досліджень, які уже відбулись, алгоритм вважається досить надійним.

Популярності алгоритму Blowfish принесло те, що Брюс Шнайєр створив саме алгоритм і не запатентував його, тому що в той час майже всі криптографічні алгоритми були запатентовані, дорогі, ненадійні, секретні та не дуже доступні для невеликих компаній [12].

Алгоритм Blowfish містить ключ, довжина якого може варіюватись від 32 біт до 448, що є доволі ефективним для застосування і експорту ключа [15].

Засновник алгоритму не ставить обмеження на використання, проте, так як він має власний сайт, то виставляє на ньому програми, які використовують даний алгоритм. На сайті уже вказано чимало комерційних програм, серед яких менеджери паролів, програми шифрування e-mail, програми для back-up та інші [15].

Як вказував автор алгоритму, вимоги до Blowfish при його створенні були такими: він повинен в рази швидшим, ніж попередні алгоритми шифрування;

алгоритм не повинен бути ускладненим, повинен бути простим для реалізації; повинен бути компактним та з легким налаштуванням стійкості [17].

І в автора це вийшло зробити, стійкість налаштовується завдяки зміні ключа шифрування, алгоритм займає лише 5Кб пам’яті (компактний), та простий для реалізації завдяки використанню нескладних математичних операцій. На 32-бітних процесорах шифрування алгоритмом Blowfish відбувається в рази швидше, ніж це було алгоритмом DES [18].

2.2 Структура алгоритму шифрування Blowfish

Існує декілька варіацій алгоритму Blowfish. Основна це та, яку створив Брюс Шнайєр.

Основні властивості [19]:

- довжина ключа варіюється від 32 біт до 448 біт;
- використовується 16 ітерацій для шифрування;
- використовується мережа Фейстеля;
- використовуються нескладні математичні операції;
- є операція створення підключів;
- є операція ініціалізації масивів P та S за допомогою секретного ключа.

Розглянемо як відбувається ініціалізація масивів P та S. Початкові значення даних масивів можна взяти з даних, які надає автор алгоритму. Ці дані є цифрами числа пі, проте перетворені у шістнадцятковий формат. Вони розподіляються відповідно для масиву S та частина для масиву P. Усі значення можна знайти перейшовши за даним посиланням: https://web.archive.org/web/20080903120910/http://www.schneier.com/code/constants.txt

S-блоки – це складова алгоритму, яка містить у собі масив даних, і використовується для операції заміни (підстановки). Такі блоки використовують для заплутування зв’язків між ключем та зашифрованим текстом. В алгоритмі Blowfish вони створюються динамічно під час операції ініціалізації S блоків за допомогою секретного ключа [20].

P-блоки – це складова алгоритму, яка містить сукупність даних, кількість якої дорівнює кількості ітерацій алгоритму плюс два значення для операцій після ітерацій, в алгоритмі Blowfish, 16 значень для 16 ітерацій циклу шифрування, та ще 2 значення для операцій ксору з правою і лівою частиною в кінці алгоритму [13].

Під час ініціалізації масивів P та S секретним ключем, відбувається генерація підключів для шифрування. Підключі у випадку Blowfish це значення P. Послідовність дій для генерації підключів [13]:

1. Створення масивів P та S та заповнення їх значеннями цифр числі пі (без врахування цифр 3,1,4). Значення повинні бути в шістнадцятковому форматі.
2. Створення циклу для операції XOR значення 32 бітів ключа K з першим підключем P1, наступна операція XOR відбувається із наступними 32 бітами ключа K та наступним підключем P2, потім XOR наступних 32 бітів ключа із P3 і так далі, поки не переберуться усі початкові підключі P і їм не буде присвоєно нове значення після операцій XOR.
3. Створення нульового рядка та присвоєння його новій змінній.
4. Шифрування створеного рядка із використанням нових значень масиву P. Для шифрування нульовий рядок подається на вхід у основну частину алгоритму шифрування Blowfish.
5. Отриманий результат потрібно помістити у значення P1 та P2.
6. Шифрування отриманого результату із зміненими значеннями P1 та P2.
7. Отриманий результат потрібно помістити у значення P3 та P4.
8. Повтор кроку 4-7 до отримання результатів для усіх підключів масиву P. Проте у кроці 4 уже шифрується не нульовий рядок, а результат отриманий після останнього шифрування із зміненими підключами. Для цього використовується 9 ітерацій.
9. Таким самим чином відбувається операція створення нових значень для блоків заміни. Для вхідної інформації для шифрування береться не нульовий рядок, а результат останніх обчислень при створенні нових значень масиву P. Для цього потрібно ще 1024 ітерацій, так як в блоці S є 4 масиви, і в кожному з цих масивів є 256 значень.

Схему для програмної реалізації частини генерації підключів можна побачити на рис. 2.2

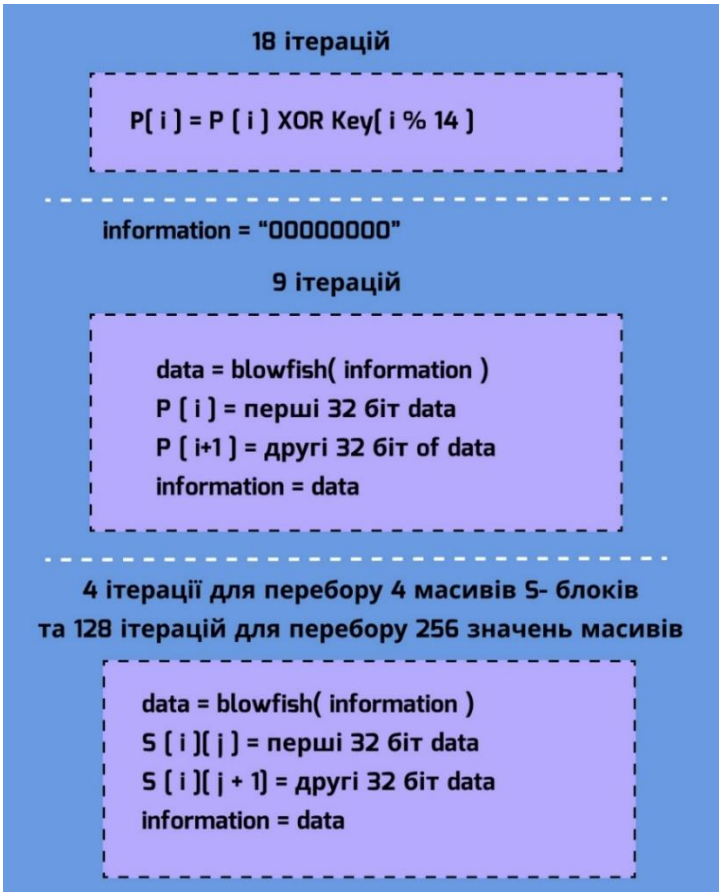

Рисунок 2.2 - Генерація підключів в алгоритмі Blowfish

Розглянемо основну частину алгоритму. Як уже було сказано вище, алгоритм заснований на мережі Фейстеля, тому першу частину по ініціалізації блоків підстановки та підключів уже пройдено. Лише після цього можна розпочинати шифрування [13].

Спочатку розглянемо мережу Фейстеля для того, щоб зрозуміти схему алгоритму шифрування Blowfish. Мережа Фейстеля це структура, яка використовується для побудови блокових шифрів. На вхід блоку подаються дані та розподіляються на дві рівні частини, частина L та R. Кількість раундів визначається обраним алгоритмом. В кожному раунді на вхід подається ключ із індексом, який дорівнює номеру раунду а також частини L та R. Також частина L передається в так звану раундову функцію F, результат раундової функції ксориться з частиною R. Після цього ліва частина без змін (початкове її значення) стає правою частиною, а

права частина після операції XOR із результатом F стає лівою частиною. Це триває впродовж визначених алгоритмом раундів, окрім останнього. В останньому раунді ліва частина залишається лівою, а права частина XOR результат F стає правою частиною [21]. Схему можна побачити на рис. 2.3

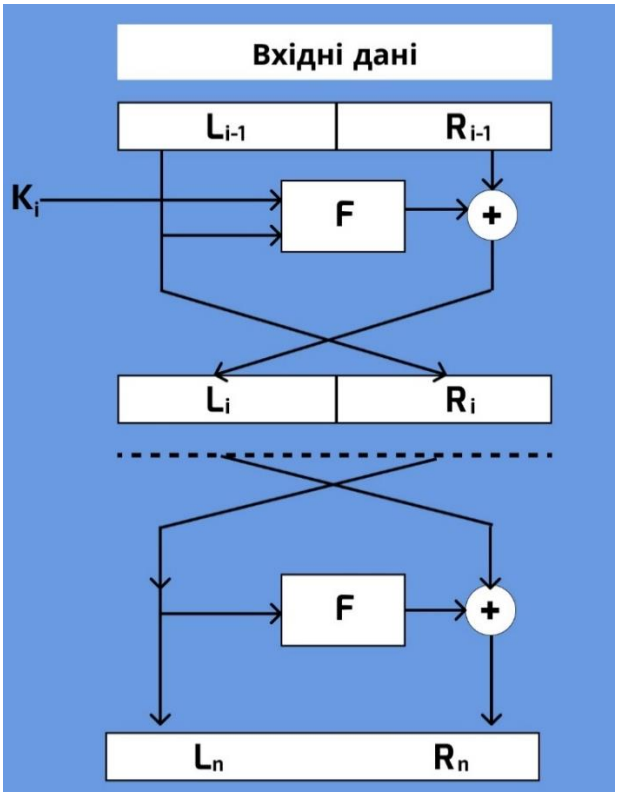

Рисунок 2.3 - Схема мережі Фейстеля

Тепер перейдемо до шифрування. Послідовність дій для шифрування [22]:

1. На вхід подаються дані, розміром 64 біта.
2. Дані поділяються на дві рівні частини по 32 біта. Ліва частина це L, права це R.
3. Дані передаються для шифрування. 16 ітерацій.
4. В кожній ітерації ліва частина виконує операцію XOR з підключем P, з індексом, який дорівнює номеру ітерації мінус 1, якщо це перша ітерація, то в програмуванні дані в масиві індексуються, починаючи з нуля, тому в першій ітерації використовується підключ P[0]. Після цієї операції дані передаються на вхід у раундову функцію. Результат цієї операції виконує операцію XOR з частиною R. І

відбувається зміна: результат F XOR R стає лівою частиною, а частина L стає правою. Так повторюється впродовж 16 ітерацій.

5. Після 16 ітерацій ліва частина стає частиною R, права частина стає частиною L.

6. Після цієї зміни, ліва частина виконує операцію XOR з підключем P18, а права – з підключем P17.

7. Результати цих операцій конкатенуються і передаються на вихід як зашифровані дані.

Детальну схему можна побачити на рис. 2.4

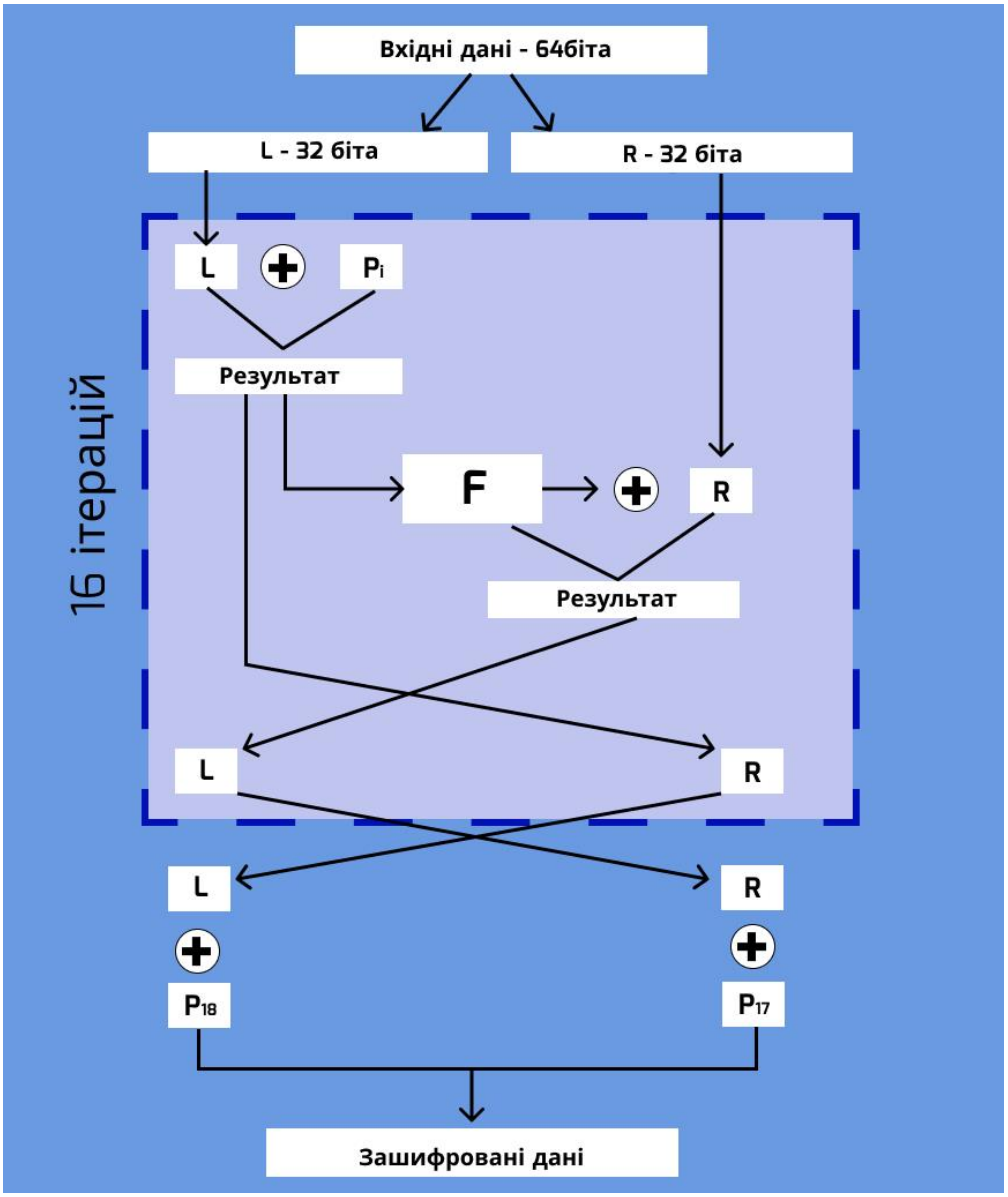

Рисунок 2.4 - Схема алгоритму Blowfish

Процес розшифрування відбувається так само, окрім моменту з використанням
ключів.

Ключі використовуються у зворотньому порядку, від P18 до P0 [22].

Розглянемо раундову функцію. Вона складається з таких етапів [22]:

1. Після передачі результату L XOR P[i], результат (32 біта) ділиться на 4 рівні частини, по 8 бітів.

2. Частини нумеруються як L1, L2, L3 та L4. Це все для того, щоб визначити який масив блоку заміни використовувати. Як було згадано вище, S-блок містить 4 блоки із 256 значеннями.

3. L1 бере використовує перший масив S-блоку. Для того, щоб визначити яке значення присвоюється після заміни частини L1 на значення з блоку, двійкове значення 8 бітів частини L1 переводиться в десяткову систему числення. Після переведення L1 бере значення з індексом масиву, який дорівнює переведеному значенню в десяткову систему.

4. Після цього крок 3 повторюється до частини L2, тільки вона уже використовує значення з другого масиву S-блоку.

5. Після заміни значень L1 та L2, ці значення додаються по модулю 2^{32}.

6. Крок 3 повторюється для частини L3, але із використанням третього масиву S-блоку.

7. Результат кроку 5 виконує операцію XOR із результатом заміни з кроку 6.

8. Крок 3 повторюється для частини L4, тільки із використанням четвертого масиву S-блоку.

9. Результат кроку 7 та кроку 8 додаються по модулю 2^{32}.

10. Результат кроку 9 повертається після виконання раундової функції і далі використовується для операцій, які уже були згадані.

Детальну схему можна побачити на рис. 2.5

Проаналізувавши структуру алгоритму Blowfish, можна зазначити, що найдовшим етапом у даному алгоритмі є ініціалізація P та блоків заміни S. Саме шифрування блоку розміром 64-біт відбувається доволі швидко.

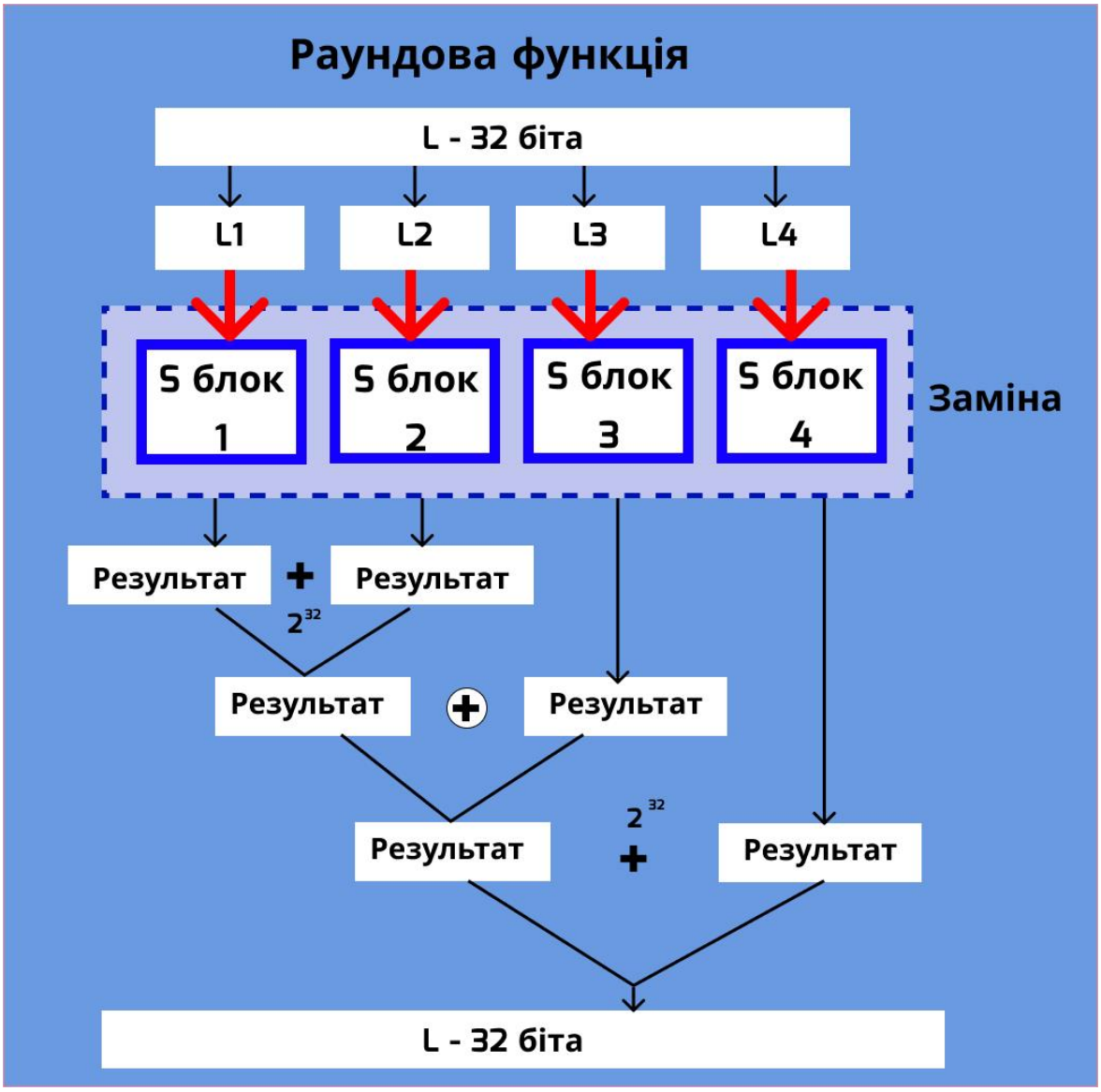

Рисунок 2.5 - Схема раундової функції в алгоритмі Blowfish

2.3 Міні версії алгоритму Blowfish

Окрім основної версії алгоритму існує декілька малих версій. Проте вони не використовуються для шифрування. Малі версії використовуються лише для криптоаналізу [19].

Версія алгоритму Blowfish-32 передбачає [19]:

- блок, розміром 32 біта;
- S-блоки із 16 масивами;
- P-блоки, розміром по 16 біт.

Версія алгоритму Blowfish-16 передбачає [19]:

- блок, розміром 16 біт;
- S-блоки із 4 масивами;
- P-блоки, розміром по 8 біт.

Також, при реалізації алгоритму, можна зменшувати кількість ітерацій. В Blowfish їх 16, проте можна зменшити до 8. Зменшення кількості ітерацій відбувається майже без послаблення захисту. Також кількість ітерацій залежить від обраної довжини ключа (від 32 біт до 448 біт). Так як в кожному раунді відбувається операція XOR лівої частини даних із підключем, то вони повинні бути однакові за розміром, і кількість підключів повинна бути на 2 більше, ніж кількість ітерацій. Тому відповідно до цих даних, необхідно обирати довжину ключа і кількість ітерацій [19].

2.4 Математичні операції в Blowfish

Відповідно до схеми алгоритму, зображеної на рис. 2.3 – 2.4, в Blowfish використовують такі математичні операції:

- XOR;
- додавання за модулем 2^{32};
- конкатенація.

Математична операція ксор (XOR) це бітова операція, яка передбачає додавання за модулем 2. Під час цієї операції дані побітово додаються за модулем 2. Результатом є 1, якщо перший біт 1 та другий 0, або перший біт 0, а другий 1, інакше результатом є 0. В випадку алгоритму Blowfish, для додавання за модулем 2 використовуються 32-бітні послідовності, результатом цієї операції також є 32-бітна послідовність [23].

Математична операція додавання за модулем 2^{32} це бітова операція. Для того, щоб два числа додати за модулем 2^{32} необхідно між собою додати два числа і потім взяти остачу від ділення результату додавання на число 2^{32}.

Наступна математична операція, яка використовується в алгоритмі Blowfish це конкатенація. Простими словами це об’єднання. Вона є останнім кроком перед отриманням результату шифрування. Після того, як ліва та права частина виконали операцію ксор із відповідними підключами, їх результати, розміром по 32 біта об’єднуються і передаються на вихід як 64-бітна послідовність [13].

2.5 Стійкість алгоритму

Наступного року після створення алгоритму Blowfish, почалось дослідження криптостійкості алгоритму. Під час проведення криптоаналізу, розроблялись атаки для злому, проте ефективні атаки були проти міні-версій Blowfish. Джон Келсі провів атаку, яка дозволяла зламати міні-версію, яка містила всього 3 раунди, порівняно з повною версією в 16 раундів. Було використано основну раундову функцію і основні операції, які в ній використовуються, XOR та додавання за модулем 2^{32}. Далі ця атака не змогла просунутись. Була також спроба створити програму для атаки brute force [13].

Наступна спроба атаки була спроба Сержа Водене. Він використав також не повний алгоритм, використав S-блоки, ініціалізація яких відбувається один раз, і не використовується залежність значень S-блоків від значень ключа. Ця атака була успішною лише для міні версії Blowfish із 8 раундами. Для успішної реалізації атаки необхідна була більша кількість відкритого тексту [13].

Проаналізувавши результати спроб атак, автор алгоритму Blowfish вказав, що атаки допомогли йому встановити деякі слабкі місця в алгоритмі. Слабким місцем є слабкий ключ. Його можна перевірити лише після ініціалізації всіх S, P блоків. Перевірка відбувається саме S-блоків на наявність однакових значень, якщо вони є, то висновок – слабкі ключі присутні [13].

Як було згадано раніше, Blowfish створений на основі мережі Фейстеля. Так як вона заснована так, що має зберігати криптографічні властивості, навіть при найпростішій реалізації цієї мережі, то є декілька способів гарантії того, що ключ,

який використовується в Blowfish є достатньо довгим, щоб забезпечити захист даних [19].

Перший спосіб – це розробка алгоритму таким чином, щоб ентропія випадкового ключа зберігалась, інакше тоді буде доступна атака brute force для перебору усіх ключів [19].

Другий спосіб – це можливе збільшення довжини ключа на стільки бітів, на скільки потрібно, щоб при зменшенні ефективності ключа ненабагато, атакуючі не отримали багато переваг для злому [19].

Blowfish приймає дані на вхід по 64 біта. Це робить алгоритм вразливим до атаки дня народження. Тому його не дуже рекомендують використовувати для шифрування даних великого розміру, тому що цей процес буде займати багато часу через те, що шифрування відбувається по 64 біта та можлива поява ідентичних зашифрованих послідовностей. В даному випадку використання алгоритму AES безпечніше і швидше, тому що він використовує блоки розміром по 128 біт. Як було згадано вище, алгоритм, точніше його міні-версії із зменшеною кількістю раундів, вразливі до атаки через використання слабких ключів [24].

Криптостійкість алгоритму Blowfish можна збільшувати або зменшувати. Все залежить від кількості раундів у алгоритмі, довжини блоку P та довжини S блоків. Якщо зменшувати кількість S-блоків, то в даному випадку буде вразливість у вигляді слабких ключів, які уже згадувались раніше, і зменшиться пам'ять. Якщо збільшити кількість S-блоків, розмір P блоків та структуру раундової функції, то за допомогою атак, які були згадані вище, не можна буде отримати дані, які шифрувались. На сьогодні, більша частина атак можлива лише в таких випадках, якщо при реалізації алгоритму були допущені помилки в коді, в інших випадках, можлива лише атака дня народження [17].

Проаналізувавши дані про деякі існуючі вразливості, можна визначити, що вразливості суттєві, проте навіть із цими вразливостями, наразі не можливо створити надійний захист даних, використовуючи лише якийсь один алгоритм шифрування, тому якщо використовувати Blowfish, то для кращого захисту можна використати ще наприклад AES.

2.6 Порівняння алгоритму Blowfish з його аналогами

Розглянемо відомі блокові алгоритми шифрування та їх переваги та недоліки. Алгоритм, який вважається надійним і є досить поширеним– Blowfish.

Переваги алгоритму Blowfish [25]:

- швидший, ніж алгоритм DES;
- незапатентований;
- простий для реалізації;
- не існує повноцінної атаки на алгоритм із 16 раундами;
- заснований на мережі Фейстеля;
- потребує невеликої кількості пам’яті для обчислень.

Недоліки алгоритму Blowfish [25]:

- розмір блоку для входу лише 64 біта;
- шифрування займає багато часу, на відміну від інший відомих алгоритмів симетричного шифрування;
- при постійній зміні ключів, шифрування відбувається тривалий час;
- можлива атака дня народження;

Алгоритм, який було створено для вдосконалення Blowfish – алгоритм Twofish, проте він не такий поширений, як Blowfish.

Переваги алгоритму Twofish [26]:

- вдосконалена версія Blowfish;
- в рази швидший за Blowfish;
- розмір даних для шифрування, який подається на вхід – 128 біт;
- заснований на мережі Фейстеля;
- використовує input whitening для заплутування зв’язків ключа та даних.

Недоліки алгоритму Twofish [26]:

- існують алгоритми, які забезпечують безпеку даних краще;
- при зміні ключа для шифрування кожного разу, шифрування займає тривалий час;

- складний для аналізу на вразливості;
- є ефективні атаки на алгоритм.

Алгоритм, який уже вважається застарілим, так як йому на заміну створили алгоритм Blowfish – алгоритм DES.

Переваги алгоритму DES [27]:

- є стандартизованим;
- нескладний у реалізації;
- шифрування залежить від ключа.

Недоліки алгоритму DES [27]:

- малий розмір ключа для шифрування;
- застарілий алгоритм;
- існують ефективні атаки на алгоритм, наприклад атака brute force;
- однакові фрагменти даних на виході будуть однаковими послідовностями.

Для підвищення криптостійкості алгоритму DES було створено алгоритм 3DES, в якому, порівняно з DES використовуються різні ключі для шифрування даних.

Переваги алгоритму 3DES [28]:

- надійніший за алгоритм DES;
- використовуються різні ключі для шифрування;

Недоліки алгоритму 3DES [29]:

- є ефективні атаки на алгоритм (наприклад Sweet32);
- доволі повільне шифрування;
- алгоритм не рекомендовано використовувати у проектах із 2023 року через наявні вразливості.

Алгоритм, який наразі є надійним та доволі розповсюдженим – AES.

Переваги алгоритму AES [29]:

- на сьогодні, це безпечний алгоритм;
- є стандартом;
- швидше шифрування, порівняно з DES;

- використовує ключі різної довжини;
- найбільш популярний алгоритм, завдяки надійності його рекомендують використовувати замість Blowfish;
- розмір блоку для вхідних даних – 128 біт.

Недоліки алгоритму AES [30]:

- не існує ефективної атаки на даний алгоритм;
- атак, які існують (наприклад, атака по часу та атака по електроспоживанню), можна запобігти, впровадивши заходи по захисту;

Розглянувши переваги та недоліки аналогів алгоритму Blowfish, можна зазначити, що звісно є набагато кращі та ефективніші алгоритми шифрування, проте основою для створення тих алгоритмів був саме алгоритм Blowfish. Наприклад, алгоритм Twofish, створений тією самою особою, хто і створив Blowfish, є покращеною версією алгоритму Blowfish. AES також був заснований завдяки аналізу ефективності існуючих блочних шифрів, тому і містить у собі удосконалені властивості, які дозволяють йому залишатись на сьогоднішній день надійним та популярним алгоритмом шифрування. Алгоритм Blowfish є оптимальним варіантом для шифрування паролів, тому що він доволі швидко шифрує невеликий об’єм даних та потребує мало пам’яті для зберігання, що є важливим для веб-додатків.

2.7 Порівняння використання Blowfish та хеш-функцій для зберігання паролів

Існують варіанти зберігання паролів у зашифрованому вигляді та у вигляді хешу. Для шифрування використовуються алгоритми симетричні (Blowfish, AES, Twofish) та асиметричні (RSA, Діффі-Хеллмана). Для хешування використовуються криптографічні хеш-функції, такі як MD5, SHA3, SHA1, Купина та інші.

При шифруванні алгоритми використовують ключ шифрування, це дозволяє для шифрування кожного пароля створювати унікальний ключ, яким можна буде розшифрувати дані.

При хешуванні не використовується ключ. Хеш-функції перетворюють паролі

в послідовність, яку неможливо розшифрувати. Із додаванням солі, безпечність зберігання паролів підвищується, тому що паролі не перетворюються в ідентичну послідовність.

Взагалі порівнювати, що краще хешування чи шифрування, некоректно, тому що вони часто використовуються для різних дій, як уже було згадано. Проте для зберігання паролів обидва способи надійні, головне обрати безпечну хеш-функцію або криптографічний алгоритм. Наприклад, є функція bcypt в PHP, це хеш-функція, проте вона заснована на алгоритмі шифрування Blowfish, і вона наразі є найбільш безпечною функцією для зберігання паролів. Тобто з алгоритму шифрування зробили хеш-функцію, покращили властивості і тепер можна використовувати для хешування.

Тому обирати треба відповідно до параметру безпеки, який спосіб буде безпечнішим, такий і можна використовувати. Але, якщо обирати з хеш-функцій, то дійсно потрібно використовувати bcypt, далі його буде детальніше розглянуто.

2.8 Використання алгоритму Blowfish

На сьогодні, існує чимало додатків різного призначення, які використовують алгоритм Blowfish. Хоч він трохи уже застарілий, але все ще використовується. Є такі основні сфери його використання [31]:

- зберігання та управління паролями;
- програми для back-up;
- захист файлів у операційній системі Linux, OpenBSD;
- шифрування файлів;
- шифрування пошти;
- захист даних в базах даних та інше.

На сайті автора алгоритму зібрані назви додатків, які створені із використанням Blowfish. Розглянемо деякі з них [31]:

- Продукти шифрування файлів та дисків: уже згаданий bcrypt, GnuPG, ShareCrypt, DriveCrypt та інші;

- Додатки для управління паролями: 1Password, Web Confidential, Password Wallet та інші;
- Додатки для резервного копіювання: Crashplan, Leo Backup, Symantec NetBackup та інші.

Для швидкого шифрування даних та дисків використовується алгоритм Blowfish, тому що він швидший за деяких своїх аналогів і забезпечує безпеку даних. Для безпечного віддаленого підключення та аутентифікації користувачів також використовують алгоритм Blowfish. Прикладами програм для віддаленого підключення, які використовують даний алгоритм є такі відомі програми як OpenSSH і PuTTY. Також для забезпечення конфіденційності даних операційна систем Linux використовує алгоритм Blowfish [16].

Висновки за розділом 2

В другому розділі було розглянуто схему алгоритму Blowfish, його особливості використання, математичні операції, які в ньому використовуються. Також проаналізувавши аналоги алгоритму шифрування Blowfish, було визначено, що є алгоритми менш ефективні, ніж Blowfish, та є алгоритми, які більш безпечні, ніж Blowfish. Не зважаючи на той факт, що на даний алгоритм наразі існує одна атака, проте її можна запобігти, шифруючи невеликий об’єм даних. Blowfish використовують у багатьох проектах для шифрування даних. Також на його основі побудована утиліта bcypt, яка є найбільш безпечною криптографічною хеш-функцію і яку рекомендовано використовувати для захисту конфіденційних даних. Ознайомившись із особливостями алгоритму Blowfish було проаналізовано, що даний алгоритм можна використовувати для забезпечення безпечного зберігання паролів. Було вияснено, що можна використовувати і алгоритм шифрування і хешування, а обирати потрібно відповідно до фактору безпеки, якщо алгоритми безпечні, то можна використовувати. Якщо обирати алгоритм шифрування, то це буде алгоритм AES або Blowfish, так як він вважається безпечний, а якщо хеш-функцію, то можна обрати bcrypt.

РОЗДІЛ 3
РОЗРОБКА МЕХАНІЗМУ БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ ІЗ ВИКОРИСТАННЯМ АЛГОРИТМУ BLOWFISH

3.1 Огляд наявних реалізацій алгоритму

Оскільки алгоритм Blowfish протягом тривалого періоду вважався достатньо надійним алгоритмом, то його впроваджували у багато програм, бібліотек, проектів. Існує чимало реалізацій алгоритму, від реалізацій на різних мовах програмування до реалізацій в бібліотеках. Реалізації алгоритму на різних мовах програмування можна знайти на сайті автора алгоритму, а також на платформі github, де кожен програміст по-своєму реалізував алгоритм. Розглянемо, які існують утиліти та бібліотеки, які містять в собі реалізований алгоритм Blowfish, та розглянемо, яким чином він реалізований та як його можна використати для шифрування даних.

Libgcrypt – це бібліотека, яка містить в собі багато криптографічних алгоритмів симетричного шифрування та асиметричного, алгоритми хешування, різні режими для шифрування. Дана бібліотека містить такі алгоритми [32]:

- AES;
- Blowfish;
- GOST28147;
- Salsa20;
- Twofish;
- MD2, MD4, MD5;
- різні версії SHA;
- Whirlpool;
- RSA;
- DSA;
- ECDH та інші.

Бібліотека Libgcrypt може використовуватись на будь-якій платформі: Windows, Linux та інших. Написана на мові програмування C. Для використання достатньо запустити декілька функцій, наприклад, gcry_cipher_open – дана функція потрібна для створення дескриптора контексту, який потрібен для подального шифрування. Як аргументи в дану функцію потрібно вказати алгоритм, яким будуть шифруватись дані, режим для шифрування, якщо потрібен, та додаткові аргументи, якщо потрібні [33].

В 2021 році в версії бібліотеки 1.9.0 була виявлена вразливість, експлуатація якої призводить до переповнення буферу. В версії 1.9.1 розробники ліквідували дану вразливість. Відповідно до цієї інформації, не потрібно використовувати версію 1.9.0, тому що використання може призвести до витоку конфіденційних даних.

Crypt-Blowfish – це модуль для шифрування алгоритмом Blowfish, який використовується для мови програмування Perl. Користуватись ним доволі легко. Достатньо створити новий об’єкт Blowfish, та застосувати до нього функцію зашифрування encrypt() чи розшифрування decrypt(). При створенні об’єкту Blowfish потрібно вказати ключ для шифрування. Детальнішу інформацію можна прочитати на сайті https://metacpan.org/pod/Crypt::Blowfish.

Наступна бібліотека, яку варто розглянути це бібліотека mcrypt. Вона включає чимало алгоритмів [34]:

- Blowfish;
- Twofish;
- AES;
- DES;
- 3DES;
- SAFER;
- IDEA;
- RC2 та інші.

Перелік усіх доступних алгоритмів можна побачити на сайті даної бібліотеки. Mcrypt дозволяє обирати режим для шифрування: CBC, CTR та інші. Для того, щоб зашифрувати дані алгоритмом Blowfish, потрібно запустити функцію

mcrypt_decrypt, в якості аргументів вказати алгоритм для шифрування, тобто Blowfish - MCRYPT_BLOWFISH, наступний аргумент це ключ для шифрування, наступний – режим шифрування, наприклад, MCRYPT_MODE_ECB. Дана бібліотека використовується найчастіше в мові програмування PHP, проте також можна використовувати на Linux системах [35].

Javascript-blowfish – це бібліотека, яка використовується для шифрування даних мовою програмування JavaScript. Вона написана на JS, JQuery та CoffeeScript. Для того, щоб користуватись даною бібліотекою, потрібно створити новий екземпляр класу Blowfish, який міститься в бібліотеці. Для шифрування використовується функція encrypt(), для розшифрування – функція decrypt(). На рис. 3.1 можна побачити приклад застосування бібліотеки [36].

```
let blowfishExemplar = new Blowfish(key);
let resultOfEncrypting = blowfishExemplar.encrypt("Tetiana
  Yuzhakova");
let decrypted = blowfishExemplar.decrypt(resultOfEncrypting);
```

Рисунок 3.1 - Використання javascript-blowfish для шифрування та розшифрування

Для того, щоб спробувати DEMO версію даного алгоритму, можна зайти на сайт (https://plnkr.co/edit/CbOyJKaRcspIAokgiomT?p=preview&preview), який створили розробники даної бібліотеки, та спробувати зашифрувати дані та розшифрувати для того, щоб краще зрозуміти, яким чином відбувається цей процес із використанням даної бібліотеки.

Розглянемо найпопулярнішу реалізацію Blowfish у вигляді bcrypt. Bcrypt – це хеш-функція, створена на основі алгоритму Blowfish, проте вона забезпечує захист в рази краще, за інші хеш-функції, вона є наразі найкращою для забезпечення безпечного зберігання паролів. Її використовують в операційній системі OpenBSD та в деяких інших UNIX системах, по замовчуванню для хешування паролів. Bcrypt реалізований на різних мовах програмування (JavaScript, Python, Java, PHP та інших) [37].

Bcypt у своєму алгоритмі використовує складну систему для генерації підключів, які використовуються для шифрування у кожному раунді. На значення підключів впливає значення випадково згенерованої солі та значення введеного паролю. Для кожного раунду вхідне значення є значення попереднього підключа, тобто значення результату наступного раунду залежить від результату попереднього підключа. Кількість раундів можна налаштовувати. Початкові значення S та P блоків, як і в Blowfish, є цифрами числа Pi [37]. На рис. 3.2 можна побачити як застосовувати bcrypt

```
const bcrypt = require("bcrypt");
const countRounds = 12;
const text = "TetianaYuzhakova";
bcrypt
  .hash(text, countRounds)
  .then(hash => {
    console.log(hash);
  })
  .catch(err => console.error(err.message));
```

Рисунок 3.2 - Використання bcrypt у Javascript

Є чимало сайтів, які дозволяють хешувати дані онлайн за допомогою bcrypt. На рис. 3.3 можна побачити приклад використання такого сайту.

Bcryptjs – це бібліотека bcrypt, проте вона більше розроблена саме для використання у мові програмування JavaScript. Для використання її потрібно встановити з допомогою npm менеджера. Та імпортувати у свій проект можна через команду required(). Функція bcryptjs запускається так само як функція bcrypt(), тобто потрібно запустити функцію genSalt() для генерування випадкової солі та запустити функцію hash() для хешування даних. При запуску функції генерування солі необхідно вказати кількість раундів для генерації солі. А при запуску функції хешування потрібно вказати пароль для шифрування та сіль. Дана бібліотека підтримується не на всіх платформах, лише на 32-бітному та 64-бітному Windows, 64-бітному Linux та на MacOS [38].

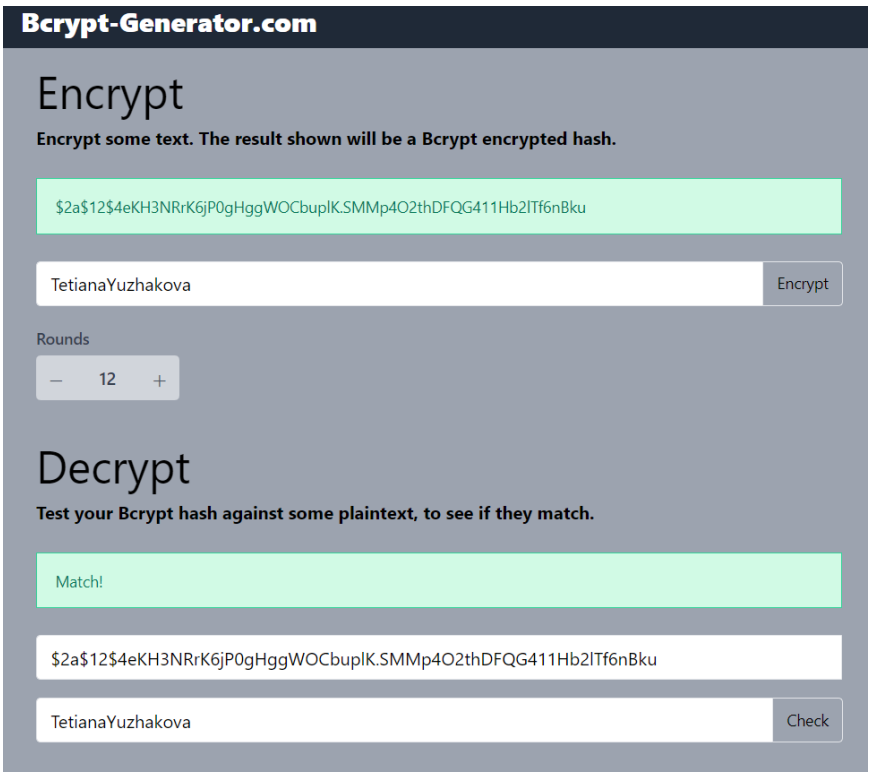

Рисунок 3.3 - Хешування даних та порівняння хешу з початковими даними

Наступний модуль, який включає в себе алгоритм Blowfish, є crypt. Crypt – це модуль для хешування даних. Він містить в собі хешування SHA-256, SHA-512, Blowfish, MD5 та crypt. Найсильнішим методом серед названих хеш-функцій є SHA-512. Для створення хешу потрібно вказати текст для хешування та згенерувати або обрати сіль для певної хеш-функції [39, 40]. Приклад використання модулю crypt можна побачити на рис.3.4

```
def hash(passw):
    s=crypt.METHOD_BLOWFISH
    h=crypt.crypt(passw,s)
    print ("Blowfish: ",h)

    s1=crypt.METHOD_CRYPT
    h1=crypt.crypt(passw,s1)
    print("Crypt: ", h1)
```

Рисунок 3.4 - Використання модулю crypt у Python

Також існує модуль crypto-js. Crypto-js – це модуль, який використовується у

JavaScript. Для того, щоб його використовувати, його треба встановити за допомогою менеджеру пакетів. За допомогою модуля crypto-js можна хешувати дані за допомогою хеш-функцій md5 та різних версій SHA. Також можна шифрувати дані за допомогою алгоритму AES, Blowfish, 3DES [38]. Синтаксис для використання даного модулю можна побачити на рис. 3.5

```
var importedModule = require("crypto-js");

var ciphertext = importedModule.BF.encrypt('Tetiana Yuzhakova', '551122').toString();

var bytes  = importedModule.BF.decrypt(ciphertext, '551122');
var originalText = bytes.toString(importedModule.enc.Utf8);
```

Рисунок 3.5 - Використання crypto-js

3.2 Розробка механізму для збереження паролів користувачів при реєстрації на сайті

Для розробки веб-додатку із формою реєстрації, формою входу та шифруванням паролів за допомогою алгоритму Blowfish було обрано такі технології:

- HTML;
- CSS;
- JS;
- WEB SQL.

Дані користувачів зберігаються в базі даних в браузері клієнта, база даних – WEB SQL. Було обрано дану базу даних для демонстрації результатів шифрування одразу в браузері. WEB SQL працює так само, як і інші бази даних, на основі SQL запитів. Проте дана база даних не працює в інших браузерах, окрім Chrome, Safari та Opera.

3.2.1 Архітектура веб-додатку

Веб-додаток складається з трьох веб-сторінок.

Перша сторінка – форма реєстрації (рис. 3.6), друга сторінка – форма входу в інтернет-магазин (рис.3.7), третя сторінка – це сам інтернет-магазин (рис.3.8).

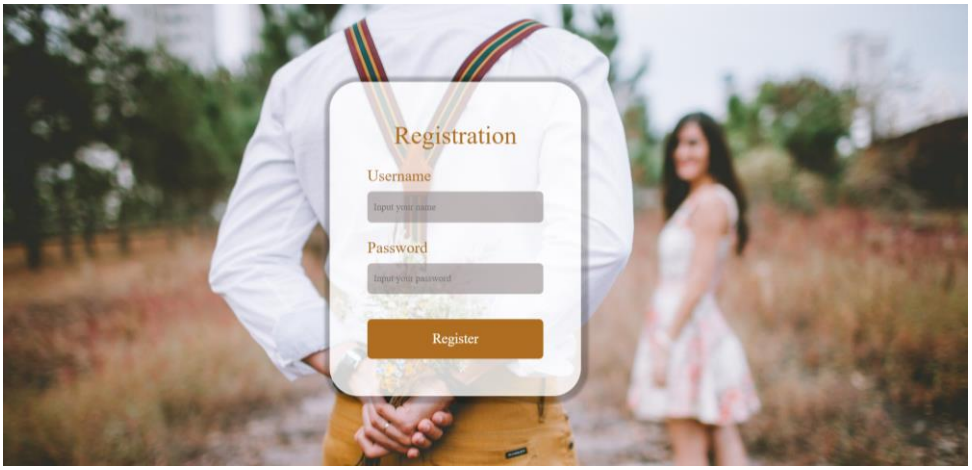

Рисунок 3.6 – Форма реєстрації в інтернет-магазині

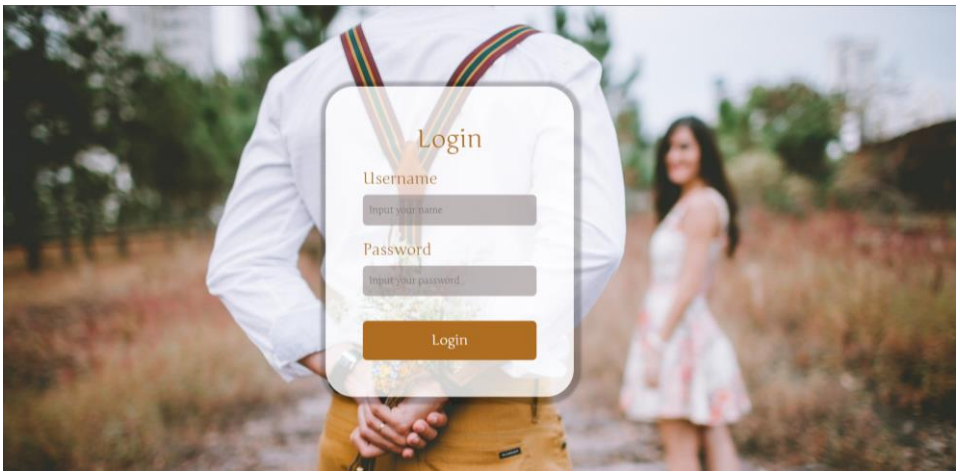

Рисунок 3.7 – Форма входу в інтернет-магазин

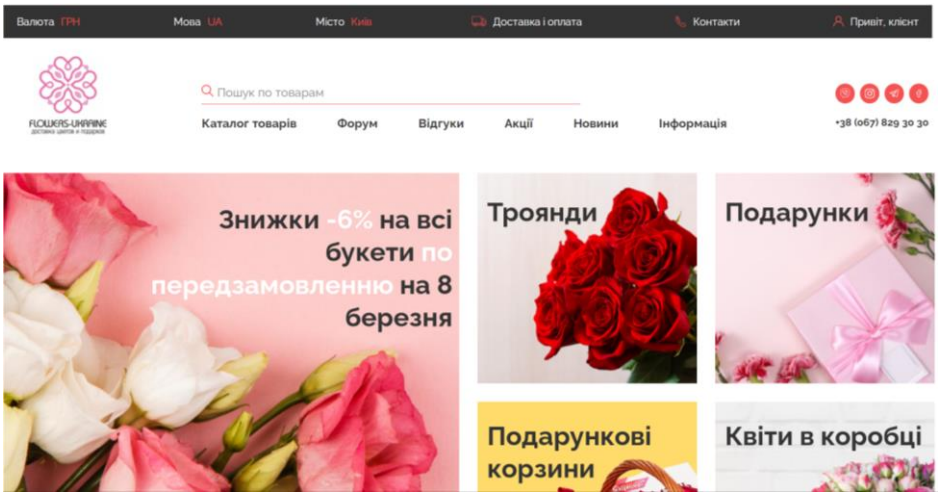

Рисунок 3.8 – Інтернет-магазин квітів

Після успішної реєстрації, яка зображена на рис. 3.6, дані користувача зберігаються в WEB SQL, це можна побачити на рис. 3.9.

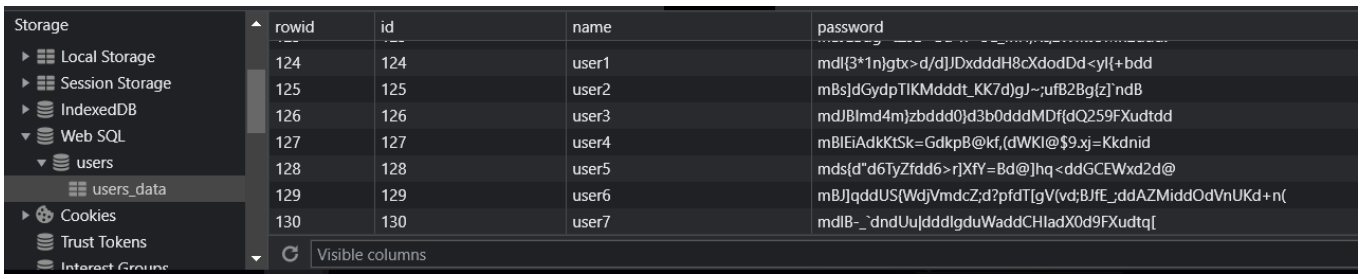

Рисунок 3.9 – Збереження id, імені та зашифрованого паролю користувача

Id користувача потрібно зберігати, тому що саме від id залежить яка сіль згенерується для того, щоб потім її додати до основного паролю, і зашифрувати. Сіль зазвичай використовують для хеш-функцій, проте для шифрування також можна використовувати.

Після реєстрації користувача, з’являється повідомлення про те, що можна увійти (рис.3.10), і одразу відкривається сторінка для логіну.

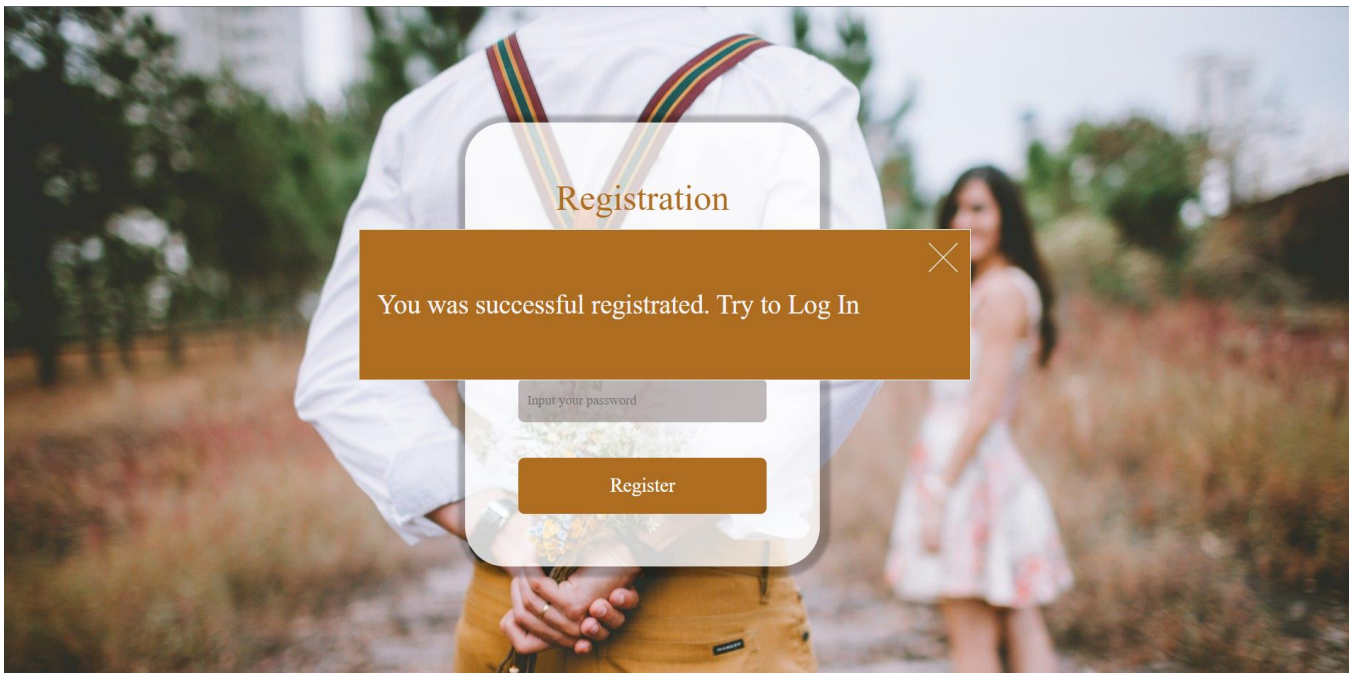

Рисунок 3.10 – Повідомлення про реєстрацію

На сторінці для логіну в інтернет-магазин також потрібно ввести логін та пароль, після натискання на кнопку входу перевіряється логін із наявними в базі даних, потім пароль шифрується і, якщо зашифрований пароль збігається з паролем із бази даних, то відбувається перехід на сторінку інтернет-магазину квітів.

3.2.2 Програмна реалізація веб-додатку

Логіка веб-сайту, тобто шифрування паролів та перехід на сторінки, реалізовано на мові програмування JavaScript. Є два файли, підключені до сторінок, перший призначений для сторінки реєстрації, відповідає за шифрування паролю, поміщення даних користувача у створену базу даних, інший файл – для сторінки входу, він відповідає за шифрування введеного паролю, вибірку даних з бази даних, та порівняння введених даних користувача із даними в базі даних.

Основна частина веб-сайту це саме логіка, тому розглянемо перший файл javascript. В моєму проекті це файл script_reg.js. Блок-схему даної програми можна побачити на рис. 3.11

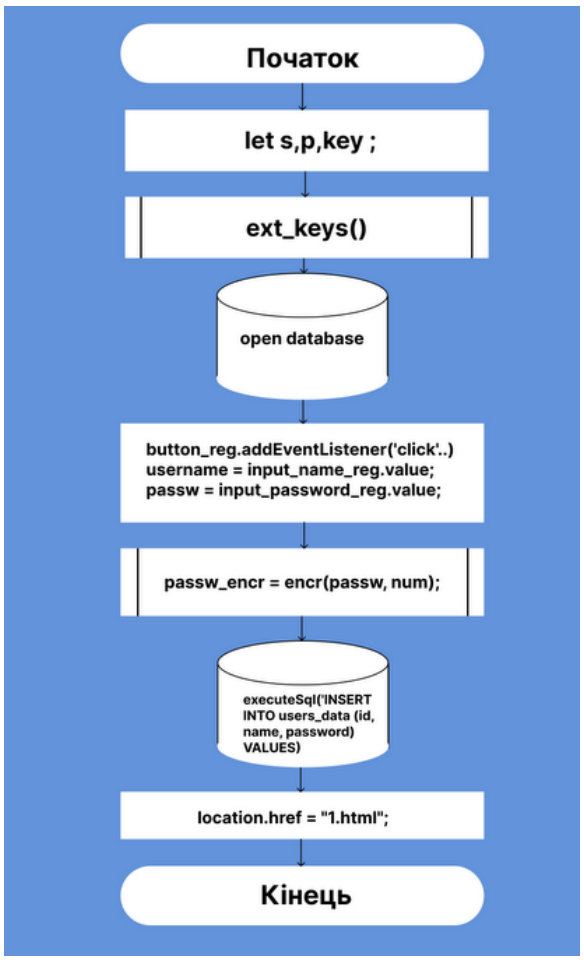

Рисунок 3.11 - Блок-схема програми script_reg.js

Перша частина складається з ініціалізації s-блоку, p-блоку та ключа для шифрування. Значення для s та p блоків було взято з сайту https://web.archive.org/web/20080903120910/http://www.schneier.com/code/constants.txt

На цьому сайті вказані стандартні, початкові значення цих блоків, ці значення є цифрами числа Pi, переведені в 16-річну систему числення, їх надав для використання автор алгоритму Blowfish.

Наступна частина програми це відкриття бази даних:

*const allMyData = openDatabase('users', '1.0', 'Yuzhakova_DB', 3 * 1024 * 1024);*

Відкривається база даних для того, щоб можна було записувати дані в неї. База даних містить перше поле це rowid, воно є незмінним, інші поля це id, name та password.

Наступний етап це ініціалізація та присвоєння значення змінним, які відповідають за кнопки та поля вводу.

```
input_name_reg = document.querySelector('input[name="name_reg"]');
input_password_reg = document.querySelector('input[name="passw_reg"]');
button_reg = document.querySelector('input[type="submit"]');
modal = document.querySelector('.modal');
cross = document.querySelector('.cross');
```

Змінним присвоюються значення після знаходження елементів у DOM сайту. Наприклад, змінна *input_name_reg* має значення поля вводу імені користувача, через цю змінну можна буде дізнатись, яке значення ввів користувач у поле вводу Username.

Після ініціалізації змінних встановлюється обробник подій click на кнопку Register.

```
button_reg.addEventListener('click', function(e){});
```

Функція, яка виконується при натисканні на кнопку «Зареєструватись»:

```
e.preventDefault();
username = input_name_reg.value;
passw = input_password_reg.value;
allMyData.transaction(function(regData) {
    regData.executeSql("SELECT * FROM users_data", [], function(regData, data) {
    num = data.rows.length;
    num++;
    passw_encr = encr(passw, num); }); });
allMyData.transaction(function(regData) {
regData.executeSql('INSERT INTO users_data (id, name, password) VALUES (?, ?, ?)', [num, username, passw_encr]);
    input_name_reg.value = "";
    input_password_reg.value = "";
    modal.style.display = "flex"; });
```

Змінним username та passw присвоюються значення даних із полів вводу логіна та паролю. Після цього виконується запит до бази даних *SELECT * FROM*

users_data", [], function(regData, data), в ньому проводиться вибірка усіх даних з таблиці і за допомогою функції, перебираються дані та дізнаємось скільки рядків є у таблиці, це потрібно для того, щоб дізнатись ідентифікатор для наступного користувача, також під час цього запиту викликається функція *encr()* для шифрування паролю. Інший запит до бази даних це запит щодо запису даних в неї:

INSERT INTO users_data (id, name, password) VALUES (?, ?, ?)', [num, username, passw_encr]

В даному запиті вказується, що в таблицю *users_data* потрібно записати значення ідентифікатора, який був визначений в попередньому запиті до бази даних, значення імені користувача та значення зашифрованого паролю.

Ще одна маленька частина коду, яка встановлює обробник для подій натискання на кнопку закриття модального вікна, яке з’являється після успішної реєстрації:

```
cross.addEventListener('click', function() {
    modal.style.display = "none";
    location.href = "1.html";
});
```

При натисканні кнопки закриття модального вікна відкривається одразу форма входу в інтернет-магазин.

Наступні функції, які є в коді це функції, які реалізують шифрування паролю. Перша з них – це функція ініціалізації значень s та p блоків за допомогою ключа. Блок-схему даної функції можна побачити на рис. 3.12.

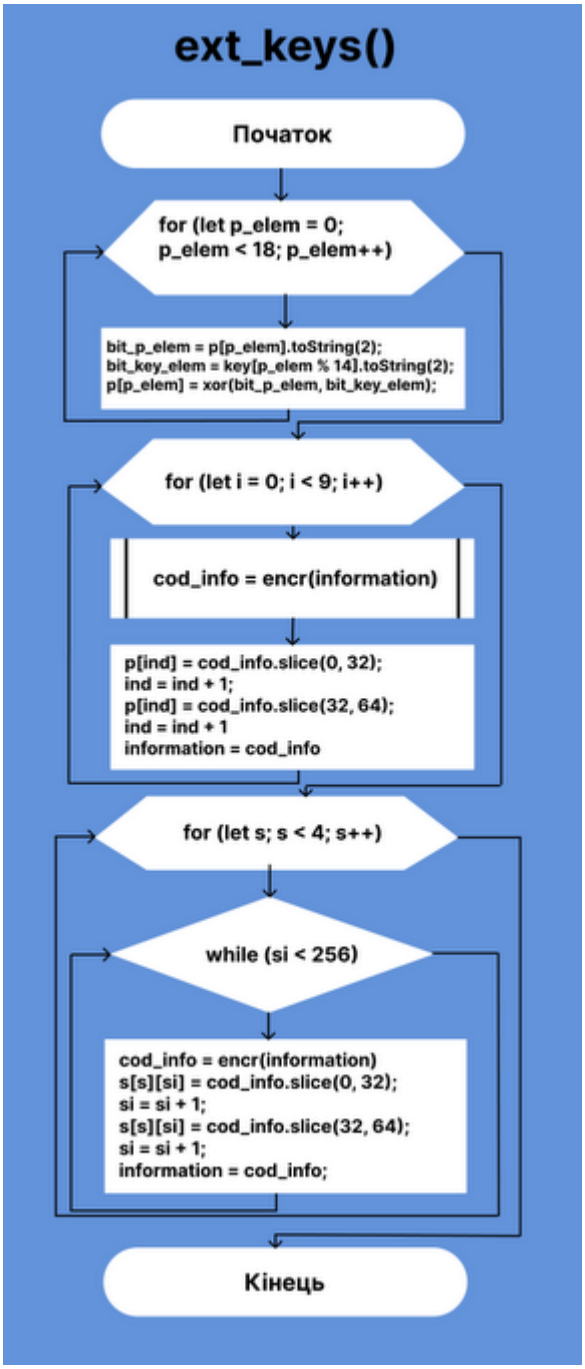

Рисунок 3.12 - Блок схема функції ініціалізації значень S-блоків та підключів

Розглянемо її по частинах:

```
for (let p_elem = 0; p_elem < 18; p_elem++) {
        bit_p_elem = p[p_elem].toString(2);
        bit_p_elem = add_zeros(bit_p_elem);
        bit_key_elem = key[p_elem % 14].toString(2);
        bit_key_elem = add_zeros(bit_key_elem);
        p[p_elem] = xor(bit_p_elem, bit_key_elem);
```

```
}
```

В даному циклі перебираються значення масиву P, які були на початку ініціалізовані, переводяться в двійкову систему числення за допомогою методу toString(2), до них додаються нулі на початок за допомогою функції *add_zeros()*, щоб довжина значення P була 32 біта. Така сама процедура проводиться для першого підключа, і після того, як були отримані значення в двійковій системі числення, то значення P та підключа виконують операцію XOR і отриманий результат присвоюється значенню P. Це відбувається, поки не переберуться всі значення P.

Друга функція відповідає за шифрування спочатку інформації у вигляді нулів, а потім присвоєння першої половини даних (функція *cod_info.slice(0, 32)*) з результату шифрування першому значенню P блоку, а другої половини (функція *cod_info.slice(32, 64)*) – наступному значенню P блоку:

```
let information = "00000000";
for (let i = 0; i < 9; i++) {
        cod_info = encr(information)
        p[ind] = cod_info.slice(0, 32);
        ind = ind + 1;
        p[ind] = cod_info.slice(32, 64);
        ind = ind + 1
        information = cod_info
}
```

Остання функція – це уже присвоєння нових значень масивам S-блоку. Відбувається така сама процедура, як і в попередній функції, проте значення присвоюються масивам S-блоку:

```
for (let s; s < 4; s++) {
        while (si < 256) {
                cod_info = encr(information)
                s[s][si] = cod_info.slice(0, 32);
                si = si + 1;
                s[s][si] = cod_info.slice(32, 64);
```

```
        si = si + 1;
        information = cod_info;}
    }
```

Функції encr(password, num), enc_blocks(text), blowfish_encryption(password), main_function(left_side) це функції, які забезпечують шифрування паролю. Викликаються вони в такому ж самому порядку, як і написані. Тобто спочатку, як було згадано вище, функція encr(password, num) викликається, коли перед записом даних в WEB SQL, пароль шифрується відповідно до вказаного ідентифікатора. Блок-схему даної функції можна побачити на рис. 3.13.

Розглянемо дану функцію. Спочатку відбувається ініціалізація змінних, та змінній sol присвоюється випадкова послідовність з цифр та літер різних регістрів, з якої буде генеруватись випадкова сіль для паролю, відповідно до ідентифікатора користувача. В змінну text буде записуватись пароль та сіль, переведені в двійкові систему числення.

```
let text = "";
let sol =
'HnqvKxQXUWFI1BZ9fsjORa4wptue263CPADrYVM0cd5NzLbhSyo8m7TgGEilkJ'
```

Після ініціалізації змінних йде генерація солі, відповідно до ідентифікатора користувача. Сіль генерується розміром в 40 символів:

```
while (s < 41)
{
    r += sol[num % s];
    s = s + 1;
}
```

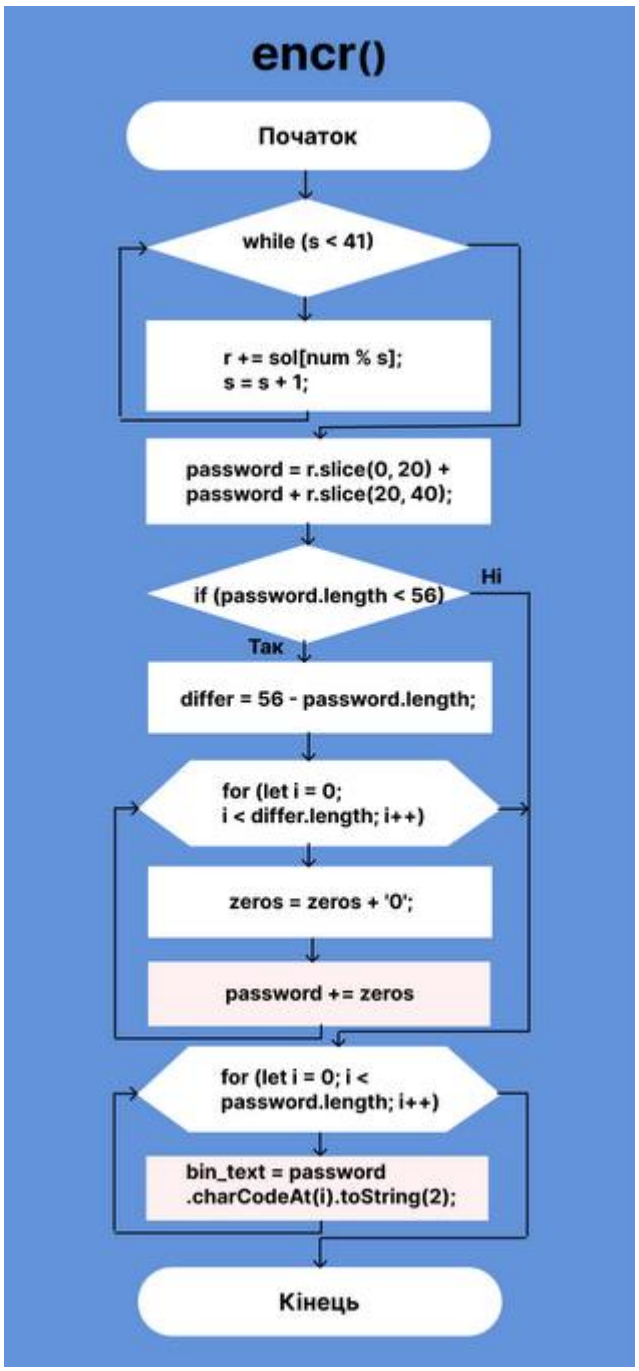

Рисунок 3.13 – Блок-схема функції, де пароль перетворюється в двійкову систему числення

Після цього дані які будуть шифруватись розміщуються в такому порядку: спочатку йде 20 символів солі, потім усі символи паролю, потім інші 20 символів солі. Це робиться для переміщування солі з паролем для покращення надійності. Якщо довжина паролю із сіллю менше 56 символів, то відбувається додавання нулів до них:

if (password.length < 56) {

```
        differ = 56 - password.length;
        zeros = ""
        for (let i = 0; i < differ.length; i++) {
            zeros = zeros + '0';
        }
        password += zeros
    }
```

Після того, як довжина паролю досягла 56 символів, ці символи перетворюються в двійковий формат завдяки функції *charCodeAt(i).toString(2)*, яка дозволяє спочатку отримати код символу, а потім перетворити символ в двійкову систему числення, після того усі біти записуються в змінну text.

```
    for (let i = 0; i < password.length; i++)
    {
        bin_text = password.charCodeAt(i).toString(2);
        if (bin_text.length < 8)
        {
            differ = 8 - bin_text.length;
            for (let d = 0; d < differ; d++) {
                zeros = zeros + '0';}
            bin_text = zeros + bin_text;}
        text += bin_text;
        }
    }
```

Оскільки пароль уже сформований і записаний у вигляді 0 та 1, то викликається функція *enc_blocks* із параметром *text.*

```
result = enc_blocks(text);
```

В функції *enc_blocks()* відповідний пароль поділяється на частини по 64 біта, тому що Blowfish шифрує дані блоками по 64 біта. І викликається функція *blowfish_encryption(),* в яку передається 64-бітна послідовність паролю. В цій

функції уже відбувається шифрування за алгоритмом Blowfish. Блок-схему функції *enc_blocks()* можна побачити на рис. 3.14.

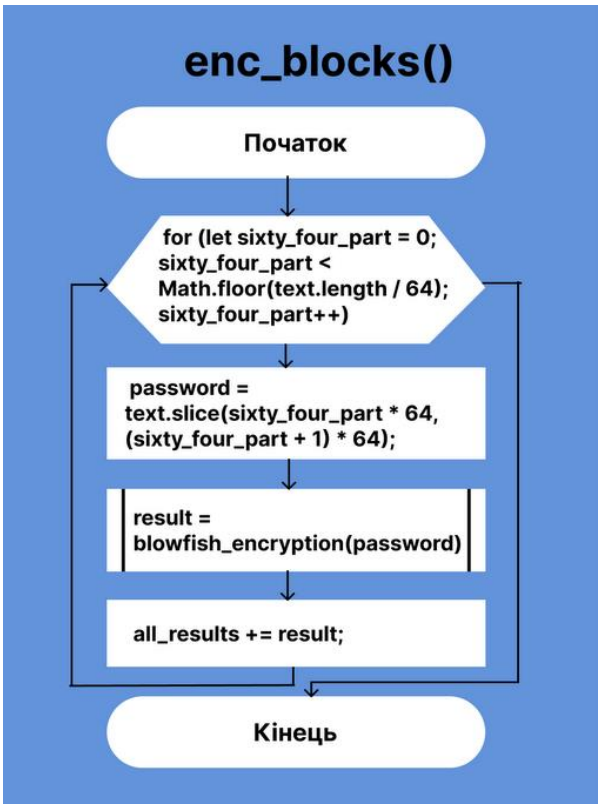

Рисунок 3.14 – Блок-схема функції, де пароль поділяється на блоки по 64 біта

Блок-схему функції *blowfish_encryption()* можна побачити на рис. 3.15. В другому розділі було описано, що перше, що відбувається при шифруванні це розділення блоку на дві частини, за це відповідають такі рядки коду:

```
left_side = password.slice(0, 32);
right_side = password.slice(32, 64);
```

Тобто ліва частина це перші 32 біта паролю, а права частина це наступні 32 біта паролю. Далі відбувається 16 раундів шифрування:

```
for (let r = 0; r < 16; r++) {
    pr = p[r].toString(2);
    pr = add_zeros(pr);
    left_side = xor(left_side, pr);
    left_side_after = main_function(left_side);
```

right_side = xor(right_side, left_side_after);

left_side, right_side = change(left_side, right_side);

};

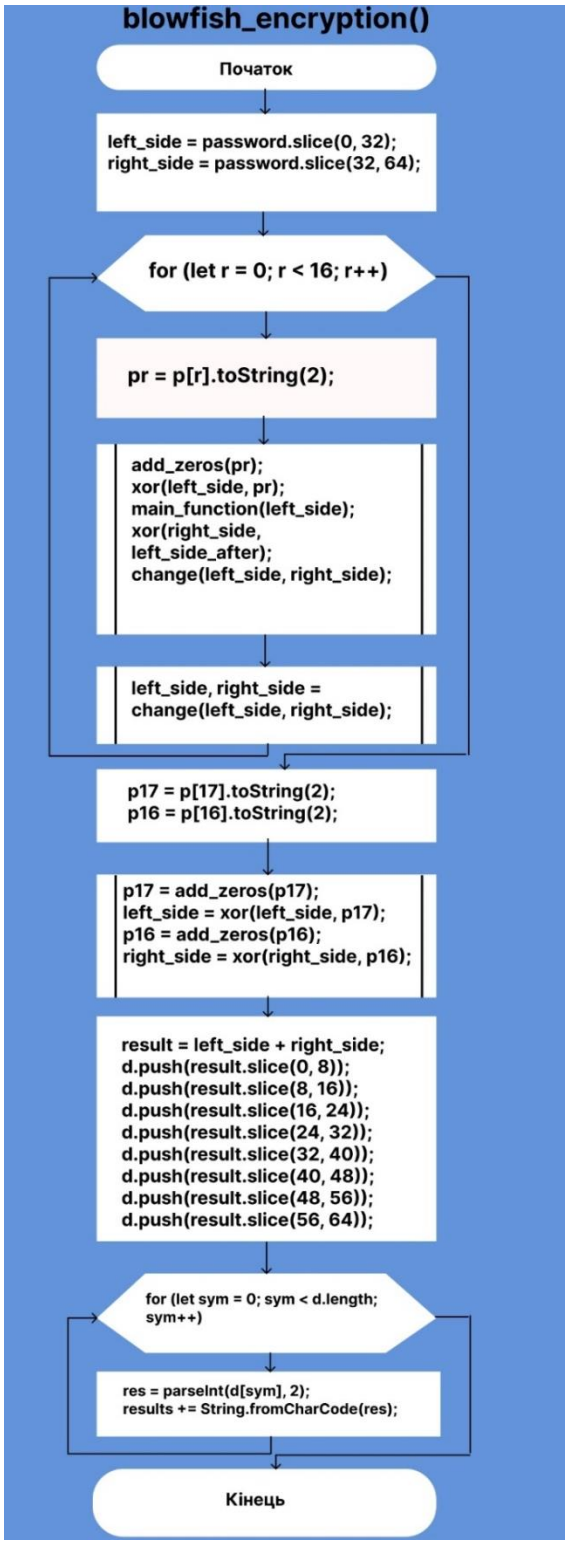

Рисунок 3.15 – Блок-схема функції, де дані шифруються за алгоритмом Blowfish

На початку кожного раунду, береться відповідне до раунду значення підключа p[i], перетворюється в двійкову систему числення (метод toString(2)), додаються нулі для того, щоб при наступній операції XOR з лівою частиною, вони мали однаковий розмір – 32 біта. Після цього, ліва частина передається на вхід в раундову функцію, яку розберемо далі. Результат, отриманий після раундової функції виконує операцію XOR із правою частиною, після цього права і ліва частина міняються місцями за допомогою функції *change(left_side, right_side)*.

На цьому 16 раундів закінчуються, і відбуваються наступні операції:

```
left_side, right_side = change(left_side, right_side);
p17 = p[17].toString(2);
p17 = add_zeros(p17);
left_side = xor(left_side, p17);
p16 = p[16].toString(2);
p16 = add_zeros(p16);
right_side = xor(right_side, p16);
result = left_side + right_side;
```

Після закінчення 16 раунду, ліва та права частина знову міняються місцями. Підключ p[17] переводиться в двійкову систему числення, до нього додаються нулі, якщо їх не вистачає до 32 бітів, і після цього даний підключ виконує операцію XOR із лівою частиною. Така сама послідовність дій відбувається із підключем p[16], проте він виконує операцію XOR із правою частиною. Після виконання цих операцій, результати конкатентуються (*result = left_side + right_side*).

Так як паролі в базах даних не зберігаються у вигляді нулів та одиниць, тому результат отриманий при шифруванні блока даних, потрібно перевести в символи, в десяткову систему числення. Для цього виконуються такі операції:

```
let d = [];
d.push(result.slice(0, 8));
d.push(result.slice(8, 16));
d.push(result.slice(16, 24));
d.push(result.slice(24, 32));
```

```
d.push(result.slice(32, 40));
d.push(result.slice(40, 48));
d.push(result.slice(48, 56));
d.push(result.slice(56, 64));
results = '';
 for (let sym = 0; sym < d.length; sym++) {
        res = parseInt(d[sym], 2);
        if (res <= 160 && res >= 127) {
         res = res - 40;
} else if (res <= 32 && res >= 0) {
        res = res + 40;
} else if (res >= 161) {
        res = 100;
}
results += String.fromCharCode(res);
}
```

На початку, весь результат, розміром 64 біта, поділяється на частини по 8 біт і записується в масив d за допомогою методу push(). Після цього, кожен елемент масиву перебирається, за допомогою функції parseInt() знаходиться код даного символа, перевіряється чи код відноситься до символів алфавіту, цифр або символів математичних операцій, якщо не відноситься, то відповідно зменшується код, щоб він входив до дозволених символів. Після того, як код перевірено, він перетворюється в символ, тобто за допомогою функції String.fromCharCode() знаходиться відповідний символ до визначеного коду, і результат записується в змінну results.

Тепер розглянемо раундову функцію main_function(). Блок-схему даної функції можна побачити на рис. 3.16. Вона складається із декількох частин. Перша частина це визначення блоку заміни для перших восьми біт лівої частини блоку, яка подається на вхід в дану функцію. Це відбувається за допомогою такого рядка коду:

```
result = s[0][parseInt(left_side.slice(0, 8), 2)]
```

Рисунок 3.16 – Блок-схема раундової функції алгоритму Blowfish

Тобто перші 8 біт лівої частини перетворюються в десяткову систему числення і це результатом буде індекс першого S блоку, який буде заміною для цих

8 біт. Результат (*result_bit1*) переводиться в двійкову систему числення, і до нього додається префікс ‘0b’, для того, щоб можна було далі виконати операцію додавання. Друга частина - наступні 8 біт перетворюються в десяткову систему числення, і стають індексом для другого S блоку. Так само, як і в попередній заміні, результат (*result_bit2*) перетворюється в двійкову систему числення і до нього додається префікс ‘0b’.

Оскільки відомі уже результати перших двох замін, то відбувається операція додавання за модулем 2^{32}:

*result_bit1and2 = (+result_bit1 + +result_bit2) % 2 ** 32;*

result_bit1and2 = result_bit1and2.toString(2);

result_bit1and2 = add_zeros(result_bit1and2);

Спочатку відбувається додавання результатів перших двох замін (*+result_bit1 + +result_bit2*), і результат береться за модулем 2^{32}.

Третя частина – це заміна наступних 8 біт лівої частини блоку. Процедура така сама, як і заміна перших двох блоків:

result = s[2][parseInt(left_side.slice(16, 24), 2)];

Результат заміни даного блоку, переведений в двійкову систему числення (*result_bit3*) виконує операцію XOR із результатом додавання за модулем 2^{32} перших двох блоків:

result_bit1and2and3 = xor(result_bit1and2, result_bit3);

result_bit1and2and3 = "0b" + result_bit1and2and3;

І також додається до результату префікс ‘0b’.

Остання частина раундової функції – це заміна четвертого восьмибітного блоку лівої частини. Процедура заміни аналогічна з замінами попередніх блоків. Результат заміни записується в змінну *result_bit4* та до неї додається префікс ‘0b’. Після цього виконується операція додавання за модулем 2^{32} результату заміни 4 блока та результату виконання операції XOR попередніх двох блоків:

*result_s = (+result_bit1and2and3 + +result_bit4) % 2 ** 32;*

result_s = result_s.toString(2);

result_s = add_zeros(result_s)

Результат переводиться в двійкову систему числення. Ось така раундова функція виконується протягом 16 раундів.

Це було розглянуто файл script_reg.js, який відповідає за функціонал форми реєстрації.

Файл script_log.js відповідає за функціонування форми входу в інтернет-магазин. Шифрування відбувається за таким самим принципом, як і в файлі script_reg.js. Відрізняється лише те, що при залогіненні в систему перевіряються дані імені користувача та пароль із тими, що є в базі даних.

На кнопку «LOGIN» поставлено обробник подій на натискання кнопки. Після натискання відбуваються такі дії:

```
username_log = input_name_log.value;
passw_log = input_password_log.value;
allMyData.transaction(function(regData) {
    regData.executeSql("SELECT * FROM users_data", [], function(regData, data) {
    for (let i = 0; i < data.rows.length; i++) {
        if (data.rows.item(i)['name'] == username_log) {
            num = data.rows.item(i)['id'];
            passw_encr = encr(passw_log, num);
}}});
regData.executeSql("SELECT * FROM users_data", [], function(regData, data) {
    for (let i = 0; i < data.rows.length; i++) {
        if (username_log == data.rows.item(i)['name'] && passw_encr == data.rows.item(i)['password']) {
        location.href = "2.html";
};}});});
```

Змінним username_log та passw_log присвоюється значення, які введені користувачем, тобто логін і пароль. Після чого відкривається база даних, в якій шукаються записи, в яких значення в полі ‘name’ буде співпадати із введеним логіном користувача (d*ata.rows.item(i)['name'] == username_log*). Після того, як

відповідний запис знаходиться, ідентифікатор користувача записується в змінну num, яка відправляється як параметр, при виклику функції шифрування паролю.

Якщо зашифрований пароль співпадає із паролем, який є в базі даних, то відбувається перехід на сторінку інтернет-магазину.

Ось таким чином працює створений сайт із реєстрацією та входом користувачів в інтернет-магазин.

Висновки за розділом 3

В третьому розділі було розглянуто існуючі реалізації алгоритму Blowfish, описано як ними користуватись, та в яких мовах програмування їх можна використовувати. Створено програмну реалізацію, яка забезпечує безпечне зберігання паролів користувачів із використанням алгоритму шифрування Blowfish. Був написаний сайт із формами входу та реєстрації. Паролі користувачів зберігаються зашифрованими у WEB SQL. Було розглянуто особливості реалізації алгоритму Blowfish мовою програмування JavaScript.

ВИСНОВКИ

Під час написання дипломної роботи було досліджено проблематику безпечного зберігання паролів користувачів. Вияснено, що зберігати можна паролі в вигляді хешу або зашифрованої послідовності. Для покращення надійності зберігання паролів потрібно створювати сильні паролі, бажано для кожного нового облікового запису створювати нові паролі.

Було проаналізовано безпеку блокових алгоритмів шифрування. Було вияснено, що оптимальним алгоритмом для шифрування паролів є алгоритм Blowfish. На інші алгоритми, такі як 3DES та Twofish існують атаки та дані алгоритми не бажано використовувати для шифрування

Для реалізації механізму безпечного зберігання паролів було створено веб-сайт, який містить форму реєстрації та входу користувачів в інтернет-магазин. Шифрування паролів відбувається на основі алгоритму Blowfish.

Отже, було досягнуто основної мети дипломної роботи – розроблено механізм безпечного зберігання паролів із використанням алгоритму блокового шифрування Blowfish.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Які вимоги до складності паролю для входу в Особистий кабінет? [Електронний ресурс] Режим доступу - https://dila.ua/instruction/trebovanija-k-slozhnosti-parolja.html

2. Шифрування, хешування, засолювання – у чому різниця? [Електронний ресурс] Режим доступу - https://instagalleryapp.com/informacijna-bezpeka/shifruvannja-heshuvannja-zasoljuvannja-u-chomu/

3. Blowfish [Електронний ресурс] Режим доступу - https://www.techtarget.com/searchsecurity/definition/Blowfish

4. Криптографічна хеш-функція [Електронний ресурс] Режим доступу - https://tebapit.com/%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%87%D0%BD%D0%B0-%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F/

5. Passwords Management using Blowfish Algorithm [Електронний ресурс] Режим доступу - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.300.7607

6. Passwords Management System using Blowfish Cryptographic Algorithm with Cipher Block Chaining Mode [Електронний ресурс] Режим доступу - https://www.researchgate.net/publication/267784265_Passwords_Management_System_using_Blowfish_Cryptographic_Algorithm_with_Cipher_Block_Chaining_Mode

7. A Survey on Network Security and SecurityAuthentication using Biometrics [Електронний ресурс] Режим доступу - https://www.academia.edu/20055659/A_Survey_on_Network_Security_and_SecurityAuthentication_using_Biometrics

8. Методи аутентифікації [Електронний ресурс] Режим доступу - https://sites.google.com/site/identifikaciataautentifikacia/ponatta-pro-autentifikaciu/metodi-autentifikaciie

9. Як створити сильний пароль (і запам'ятати його) [Електронний ресурс] Режим доступу - https://ua.phhsnews.com/articles/howto/how-to-create-a-strong-password-and-remember-it.html

10. Bruteforcing blowfish [Електронний ресурс] Режим доступу - https://security.stackexchange.com/questions/68818/bruteforcing-blowfish

11. Що таке двофакторна аутентифікація і навіщо її використовувати? Давайте розберемось. [Електронний ресурс] Режим доступу - https://c.mi.com/forum.php?mod=viewthread&tid=3987800&extra=page%3D1/

12. Bruce Schneier Biography, Life, Interesting Facts [Електронний ресурс] Режим доступу - https://www.sunsigns.org/famousbirthdays/d/profile/bruce-schneier/

13. The Blowfish Encryption Algorithm—One Year Later [Електронний ресурс] Режим доступу – https://www.schneier.com/academic/archives/1995/09/the_blowfish_encrypt.html

14. Фото Брюса Шнайєра [Електронний ресурс] Режим доступу – https://docplayer.com/docs-images/74/70844291/images/20-0.jpg

15. The Blowfish Encryption Algorithm [Електронний ресурс] Режим доступу – https://www.schneier.com/academic/blowfish/

16. Blowfish Algorithm: An Interesting Overview In 7 Points [Електронний ресурс] Режим доступу – https://www.jigsawacademy.com/blogs/cyber-security/blowfish-algorithm/#Blowfish-Algorithm-Applications

17. Blowfish [Електронний ресурс] Режим доступу – https://www.wiki.uk-ua.nina.az/Blowfish.html

18. Blowfish vs. AES: What Is the Difference? [Електронний ресурс] Режим доступу – https://www.securitybind.com/blowfish-vs-aes-what-is-the-difference/

19. Introduction to Blowfish [Електронний ресурс] Режим доступу – https://www.splashdata.com/splashid/blowfish.html

20. Importance of S-Blocks in Modern Block Ciphers [Електронний ресурс] Режим доступу – https://www.researchgate.net/publication/276230393_Importance_of_S-Blocks_in_Modern_Block_Ciphers

21. Алгоритм Фейстеля [Електронний ресурс] Режим доступу – https://sites.google.com/site/sashadkeem1502/algoritmi-sifruvanna/algoritm-fejstela

22. What is the Blowfish encryption algorithm? [Електронний ресурс] Режим доступу – https://www.educative.io/edpresso/what-is-the-blowfish-encryption-algorithm

23. How Does XOR Cipher Work? — XOR Chipher Encryption [Електронний ресурс] Режим доступу – https://medium.com/@logsign/how-does-xor-cipher-work-xor-chipher-encryption-b7ad316209ca

24. The Blowfish Cipher [Електронний ресурс] Режим доступу - https://learn.saylor.org/mod/book/view.php?id=29703&forceview=1

25. What is Blowfish in security? Who uses Blowfish? [Електронний ресурс] Режим доступу – https://www.encryptionconsulting.com/education-center/what-is-blowfish/

26. Twofish vs Blowfish | Encryption Differences [Електронний ресурс] Режим доступу – https://gigmocha.com/twofish-vs-blowfish-encryption-differences/

27. Структура алгоритму des. Його переваги та недоліки. [Електронний ресурс] Режим доступу – https://studfile.net/preview/8914423/page:8/

28. Основні модифікації DES (3DES, DESX) [Електронний ресурс] Режим доступу – https://studfile.net/preview/5993348/page:9/

29. Шифрування: типи і алгоритми. Що це, чим відрізняються і де використовуються? [Електронний ресурс] Режим доступу – https://hostpro.ua/wiki/ua/security/encryption-types-algorithms

30. Advantages of AES | disadvantages of AES [Електронний ресурс] Режим доступу - https://www.rfwireless-world.com/Terminology/Advantages-and-disadvantages-of-AES.html

31. Products that Use Blowfish [Електронний ресурс] Режим доступу – https://www.schneier.com/academic/blowfish/products/

32. Libgcrypt [Електронний ресурс] Режим доступу – https://gnupg.org/software/libgcrypt/index.html

33. The Libgcrypt Reference Manual [Електронний ресурс] Режим доступу - https://gnupg.org/documentation/manuals/gcrypt.pdf

34. MCrypt [Електронний ресурс] Режим доступу - http://mcrypt.sourceforge.net/

35. PHP: Better Password Encryption using Blowfish [Електронний ресурс] Режим доступу - https://www.the-art-of-web.com/php/blowfish-crypt/

36. javascript-blowfish [Електронний ресурс] Режим доступу - https://github.com/agorlov/javascript-blowfish

37. node.bcrypt.js [Електронний ресурс] Режим доступу - https://github.com/kelektiv/node.bcrypt.js#readme

38. bcrypt vs bcryptjs vs blowfish vs crypt vs crypto js vs password hash [Електронний ресурс] Режим доступу - https://www.libtrends.info/npm-compare/bcrypt-vs-bcryptjs-vs-blowfish-vs-crypt-vs-crypto-js-vs-password-hash

39. crypt — Function to check Unix passwords [Електронний ресурс] Режим доступу - https://docs.python.org/3/library/crypt.html

40. The crypt Module [Електронний ресурс] Режим доступу - https://www.oreilly.com/library/view/python-standard-library/0596000960/ch02s21.html

ДОДАТОК А
ОПУБЛІКОВАНА ПРАЦЯ ЗА ТЕМОЮ ДИПЛОМУ

Тези на конференцію APIT 2021

«РОЗРОБКА МЕХАНІЗМУ БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ ЗА ДОПОМОГОЮ ВИКОРИСТАННЯ АЛГОРИТМУ BLOWFISH»

В сучасному світі ми щодня стикаємось з веб-ресурсами, мобільними та десктопними додатками. Зазвичай, для того, щоб працювати з ними, необхідно зареєструватись (наприклад, електронна пошта, інтернет-магазин, інтернет-банкінг, соціальні мережі, месенджери). Під час процедури реєстрації користувач повинен ввести логін, пароль, а також як додаткові атрибути – електронну пошту та інші дані.

Для того, щоб забезпечити безпечне зберігання паролю користувача використовують найчастіше такі симетричні алгоритми шифрування як DES(Data Encryption Standard), Triple DES, AES (Advanced Encryption Standard, Blowfish) та асиметричне шифрування за допомогою хеш-функцій (MD5, SHA). Проте для нашої роботи ми обрали Blowfish, тому що він є одним з найшвидших алгоритмів для забезпечення безпеки паролів.

Проблему безпечного зберігання паролів вивчали багато вчених, серед яких Savita Devidas Patil, Ashish.T. Bhole, Chirag Singh Sisodia та Aparajiit Shrivastava, Raaj Ahuja та Mukesh Ramrakhyani та Bunty Manchundiya та Sonal Shroff. Вчені звертають увагу на те, що якщо зловмисники викрадуть паролі користувачів, то з їх допомогою можна легко отримати приватні дані і потім використовувати ці дані в своїх цілях. А так як, користувачі зазвичай не бажають запам’ятовувати різні паролі для всіх своїх акаунтів, то дізнавшись один пароль можна отримати доступ до акаунтів інших ресурсів в Інтернеті, таких як пошта, соціальні мережі та інше.

Для того, щоб пом’якшити наслідки даної проблеми або повністю усунути проблему можна використовувати менеджери паролів або зберігати паролі в базах

даних в зашифрованому вигляді. Якщо розглянути менеджери паролів, то він містить в собі логіни та паролі від акаунтів користувача, для того щоб отримати до цих даних доступ, необхідно запам’ятати один лише майстер-пароль, який повинен бути дуже сильний, тобто містити в собі більше ніж 8 літер та цифр, символи, літери верхнього та нижнього регістру. Якщо розглянути збереження паролів в базах даних в зашифрованому вигляді, то для цього найчастіше використовуються хеш-функції, проте алгоритм шифрування Blowfish набагато краще шифрує паролі. Після перетворення паролю в хеш, пароль відображається у вигляді символів, літер та цифр, його по суті неможливо перетворити в початкову форму, його можна тільки підібрати перебором через атаку brute force, це займає не так багато часу. Якщо шифрувати пароль через алгоритм Blowfish, то виходить також набір літер, символів та цифр, проте для того, щоб перетворити пароль в початкову форму, необхідно знати ключ, який міститься в частині самого паролю, тому якщо підбирати перебором через атаку brute force, то це може зайняти близько 7 років.

На сьогоднішній день, в мові програмування PHP існує функція crypt(), яка дозволяє шифрувати текст алгоритмом Blowfish.Також існують схожі функції у різних мовах програмування, тому якщо використовувати їх для перетворення паролів в зашифрований текст, то це забезпечить безпечне зберігання паролю. Не завжди для хешування та зашифрування використовують унікальні солі (частини тексту, які додаються до паролю перед хешуванням та шифруванням), що робить перетворений пароль легшим для підбору. Чим більші солі додаються, тим стійкіший стає хеш до злому.

Таким чином було запропоновано такі дії, щоб зловмисники не скомпроментували приватні дані користувачів: необхідно шифрувати паролі за допомогою алгоритму Blowfish, або також використовувати менеджери паролів, проте, якщо зловмисник дізнається майстер-пароль, то усі паролі, які записані в менеджері будуть скомпроментовані. Метод шифрування паролів за допомогою алгоритму Blowfish можна використовувати для реєстрації та автентифікації на сайті, а менеджери паролів краще використовувати кожному користувачу для того, щоб безпечно зберігати усі свої паролі від різних акаунтів. Подальше дослідження

будемо проводити у можливості використання алгоритму Blowfish для забезпечення якісного та безпечного зберігання паролів.

ДОДАТОК Б
КОД ФАЙЛУ SCRIPT_REG.JS

```
input_name_reg = document.querySelector('input[name="name_reg"]');
input_password_reg = document.querySelector('input[name="passw_reg"]');
button_reg = document.querySelector('input[type="submit"]');
modal = document.querySelector('.modal');
cross = document.querySelector('.cross');

button_reg.addEventListener('click', function(e) {
e.preventDefault();
username = input_name_reg.value;
passw = input_password_reg.value;
allMyData.transaction(function(regData) {
	regData.executeSql("SELECT * FROM users_data", [], function(regData, data) {
	num = data.rows.length;
	num++;
	passw_encr = encr(passw, num);
	});
});
allMyData.transaction(function(regData) {
	regData.executeSql('INSERT INTO users_data (id, name, password) VALUES (?,
?, ?)', [num, username, passw_encr]);
	input_name_reg.value = "";
	input_password_reg.value = "";
	modal.style.display = "flex";
});
});
cross.addEventListener('click', function() {
```

```
        modal.style.display = "none";
        location.href = "1.html";
});

function ext_keys() {
         for (let p_elem = 0; p_elem < 18; p_elem++) {
                bit_p_elem = p[p_elem].toString(2);
                bit_p_elem = add_zeros(bit_p_elem);
                bit_key_elem = key[p_elem % 14].toString(2);
                bit_key_elem = add_zeros(bit_key_elem);
                p[p_elem] = xor(bit_p_elem, bit_key_elem);
        }
        let ind = 0;
        let information = "00000000";
        for (let i = 0; i < 9; i++) {
                cod_info = encr(information)
                p[ind] = cod_info.slice(0, 32);
                ind = ind + 1;
                p[ind] = cod_info.slice(32, 64);
                ind = ind + 1
                information = cod_info
        }
        let si = 0
        for (let s; s < 4; s++) {
                while (si < 256) {
                        cod_info = encr(information)
                        s[s][si] = cod_info.slice(0, 32);
                        si = si + 1;
                        s[s][si] = cod_info.slice(32, 64);
                        si = si + 1;
```

```
                information = cod_info;
            }
        }
        };

function add_zeros(result_bit) {
        len_result = result_bit.length;
        if (len_result < 32) {
            differ = 32 - len_result;
            zeros = "";
            for (let i = 0; i < differ; i++) {
                zeros += '0';
            }
            result_bit = zeros + result_bit;
        }
        return result_bit;
        };

function xor(side, p) {
        res = "";
        for (let i = 0; i < p.length; i++) {
            if (p[i] == side[i]) {
                res += '0';
            } else {
                res += '1';
            }
        }
         return res;
        };
```

```
function main_function(left_side) {
        result = s[0][parseInt(left_side.slice(0, 8), 2)];
        result_bit1 = result.toString(2);
        result_bit1 = add_zeros(result_bit1);
        result_bit1 = "0b" + result_bit1;

        result = s[1][parseInt(left_side.slice(8, 16), 2)];
        result_bit2 = result.toString(2);
        result_bit2 = add_zeros(result_bit2);
        result_bit2 = "0b" + result_bit2;

        result_bit1and2 = (+result_bit1 + +result_bit2) % 2 ** 32;
        result_bit1and2 = result_bit1and2.toString(2);
        result_bit1and2 = add_zeros(result_bit1and2);

        result = s[2][parseInt(left_side.slice(16, 24), 2)];
        result_bit3 = result.toString(2);
        result_bit3 = add_zeros(result_bit3);
        result_bit1and2and3 = xor(result_bit1and2, result_bit3);
        result_bit1and2and3 = "0b" + result_bit1and2and3;

        result = s[3][parseInt(left_side.slice(24, 32), 2)];
        result_bit4 = result.toString(2);
        result_bit4 = add_zeros(result_bit4);
        result_bit4 = "0b" + result_bit4;
        result_s = (+result_bit1and2and3 + +result_bit4) % 2 ** 32;
        result_s = result_s.toString(2);
        result_s = add_zeros(result_s)
        return result_s
        };
```

```
function change(left_side, right_side) {
        new_right = left_side;
        new_left = right_side;
        return new_right, new_left;
        };

function blowfish_encryption(password) {
        left_side = password.slice(0, 32);
        right_side = password.slice(32, 64);
        for (let r = 0; r < 16; r++) {
                pr = p[r].toString(2);
                pr = add_zeros(pr);
                left_side = xor(left_side, pr);
                left_side_after = main_function(left_side);
                right_side = xor(right_side, left_side_after);
                left_side, right_side = change(left_side, right_side);
        };
        left_side, right_side = change(left_side, right_side);
        p17 = p[17].toString(2);
        p17 = add_zeros(p17);
        left_side = xor(left_side, p17);
        p16 = p[16].toString(2);
        p16 = add_zeros(p16);
        right_side = xor(right_side, p16);
        result = left_side + right_side;
        let d = [];
        d.push(result.slice(0, 8));
        d.push(result.slice(8, 16));
        d.push(result.slice(16, 24));
```

```
        d.push(result.slice(24, 32));
        d.push(result.slice(32, 40));
        d.push(result.slice(40, 48));
        d.push(result.slice(48, 56));
        d.push(result.slice(56, 64));
        results = '';
        for (let sym = 0; sym < d.length; sym++) {
                res = parseInt(d[sym], 2);
                if (res <= 160 && res >= 127) {
                        res = res - 40;
                } else if (res <= 32 && res >= 0) {
                        res = res + 40;
                } else if (res >= 161) {
                        res = 100;
                }
                results += String.fromCharCode(res);
        }
        return results;
        };

function enc_blocks(text) {
        let all_results = "";
        for (let sixty_four_part = 0; sixty_four_part < Math.floor(text.length / 64);
sixty_four_part++) {
                password = text.slice(sixty_four_part * 64, (sixty_four_part + 1) * 64);
                result = blowfish_encryption(password);
                all_results += result;
        }
        return all_results;
        };
```

```
function encr(password, num) {
	let text = "";
	let sol =
"HnqvKxQXUWFI1BZ9fsjORa4wptue263CPADrYVM0cd5NzLbhSyo8m7TgGEilkJ";
	num = +num;
	if (num < 40 && num > 20) {
		num = num + 21
	} else if (num < 20 && num > 10) {
		num = num + 31
	} else if (num < 10 && num >= 0) {
		num = num + 41
	}
	let s = 1
	let r = ""
	while (s < 41) {
		r += sol[num % s];
		s = s + 1;
	}
	password = r.slice(0, 20) + password + r.slice(20, 40);
	let zeros;
	if (password.length < 56) {
		differ = 56 - password.length;
		zeros = ""
		for (let i = 0; i < differ.length; i++) {
			zeros = zeros + '0';
		}
		password += zeros
	}
	for (let i = 0; i < password.length; i++) {
```

```
            let zeros = "";
            bin_text = password.charCodeAt(i).toString(2);
            if (bin_text.length < 8) {
                differ = 8 - bin_text.length;
                for (let d = 0; d < differ; d++) {
                zeros = zeros + '0';
                }
                bin_text = zeros + bin_text;
            }
            text += bin_text;
        }
        result = enc_blocks(text);
        return result;
};
```