НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 519.684

«До захисту допущено»
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Побудова системи захищеного відеоспостереження з забезпеченням цілісності і захищеності даних на базі мікроконтролера ESP32»**

Виконав:
студент IV курсу, групи ФІ-94
Музиченко Олександр Романович _____

Керівник:
с.н.с, д.т.н, професор
Кудін Антон Михайлович _____

Рецензент:
Заступник начальника управління безпеки інформації департаменту безпеки Національного банку України, к.т.н., доцент
Проскуровський Роман Васильович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Музиченко Олександр Романович

1. Тема роботи: «*Побудова системи захищеного відеоспостереження з забезпеченням цілісності і захищеності даних на базі мікроконтролера ESP32*», науковий керівник дисертації: с.н.с, д.т.н, професор Кудін Антон Михайлович,

затверджені наказом по університету №__ від «__» _____ 2023 р.

2. Термін подання студентом роботи: «__» _____ 2023 р.

3. Об'єкт дослідження: процеси передачі, збереження та пошуку даних відеоспостереження

4. Предмет дослідження: забезпечення конфіденційності та цілісності даних відеоспостереження, а також пошуку по зашифрованним даним відеоспостереження без їх розшифрування.

5. Перелік завдань:

1) Аналіз задач захисту інформації, що виникають при створенні систем захищеного спостереження

2) Аналіз та вибір криптографічних алгоритмів, які доцільно застосовувати для забезпечення конфіденційності та цілісності інформації в системі захищеного відеоспостереження.

3) Розробка криптопротоколу для забезпечення цілісності та конфіденційності даних відеоспостереження, що зберігаються.

4) Розробка алгоритму пошуку по зашифрованним даним без розшифрування із використанням гомоморфних криптоалгоритмів.

5) Реалізація програмно-апаратного макету системи захищеного відеоспостереження на базі мікроконтролеру ESP32.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: Презентація доповіді

7. Орієнтовний перелік публікацій: планується доповідь на всеукраїнській конференції

8. Дата видачі завдання: 10 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2022 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2022 р.	Виконано
3	Вибір апаратного забезпечення і реалізація простої передачі даних	жовтень-грудень 2022 р.	Виконано
4	Розробка криптографічної системи	січень 2023 р.	Виконано
5	Реалізація і дослідження різних криптографічних примітивів на мікропроцесорі ESP32	лютий-квітень 2023 р.	Виконано
6	Дослідження гомоморфних шфирів і реалізація системи пошуку по зашифрованим даним	квітень 2023 р.	Виконано
7	Реалізація фінальної програми	травень 2023 р.	Виконано
8	Оформлення результатів і підготовка до захисту	1 червня - 12 червня 2023 р.	Виконано

Студент _____ Музиченко О.Р.

Керівник _____ Кудін А.М.

РЕФЕРАТ

Кваліфікаційна робота містить: 49 стор., 4 рисунки, 3 таблиці, 10 джерел. В даній роботі демонструється побудова системи відеоспостереження з криптографічним захистом. Були розглянуті основні вимоги для даної системи і побудована відповідна схема для передачі і запису даних. Для цього були проаналізовані алгоритми симетричного шифрування і побудований специфічний алгоритм пошуку по зашифрованим даним за допомогою властивості гомоморфізму.

Також була наведе реалізація даної системи в Додатку і продемонстрований результат її роботи в розділі 3.3. Описані можливості подальшої оптимізації даної програми з додаванням нових функцій і покращенням вже існуючих.

ESP32, ВІДЕОСПОСТЕРЕЖЕННЯ, СИМЕТРИЧНА КРИПТОГРАФІЯ, ГОМОМОРФІЗМ

ABSTRACT

The qualification work contains: 49 pages, 4 figures, 3 tables, 10 sources.

This work demonstrates the construction of a video surveillance system with cryptographic protection. The main requirements for this system were considered and a corresponding scheme for data transmission and recording was built. For this purpose, symmetric encryption algorithms were analyzed and a specific search algorithm based on encrypted data was built.

The implementation of this system was also given in the Appendix and the result of its work was demonstrated in section 3.3. The possibilities of further optimization of this program with the addition of new functions and the improvement of existing ones are described.

ESP32, VIDEO SURVEILLANCE, SYMMETRIC CRYPTOGRAPHY, HOMOMORPHISM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

hash — хеш-функція

login — ідентифікатор програми, яка зашита на мікроконтролер

password — пароль для автентифікації

$\oplus$ — операція побітового додавання

$>>$ — побітовий зсув вправо

$<<$ — побітовий зсув вліво

ВСТУП

Актуальність дослідження. Захист даних є критично важливими аспектами в сфері відеоспостереження, особливо з урахуванням зростаючої кількості пристроїв IoT і потенційних загроз кібербезпеці. Одна з актуальних проблем при розробці криптографічного захисту для мікроконтролера це вибір оптимальних криптографічних алгоритмів по критеріям швидкості, пам'яті і криптостійкості. Оскільки дана система має працювати на мікроконтролері, то ми маємо обмежені ресурси в пам'яті і швидкості процесора і так як ми розробляємо криптографічну систему для відеоспостереження, то дуже важливим є фактор швидкості виконання даних алгоритмів. Дані фактори призводять до того, що потрібно провести аналіз по швидкості виконання криптографічних алгоритмів для вибору оптимальних алгоритмів для даної системи на мікроконтролері ESP32.

Так як існує проблема пошуку по зашифрованим даних без їх попереднього розшифрування на мікроконтролері з обмеженими ресурсами, то в моїй роботі я спробував реалізувати систему запису даних в форматі який дозволить виконати пошуку по зашифрованим даним і сам алгоритм пошуку. Так як, це дозволить використовувати мікроконтролер, як сховище не скомпроментованих даних.

Метою дослідження є створення оптимізованої криптографічної системою захисту при передачі даних відеоспостереження на сервер і забезпечення цілісності даних, при зберіганні на енергонезалежну пам'ять мікроконтролера. Для для досягнення цієї мети в моїй роботі потрібно було вирішити наступні задачі::

1) Реалізувати код для передачі відеоданих з мікроконтролера на сервер;

2) Розробити систему в якій відбувалася б аутентифікація мікроконтролера на сервері, передача зашифрованих даних і

розшифрування їх на сервері і зберігання кадрів на мікрокотролері з можливістю пошуку по ним і підтримкою цілісності цих даних;

3) Провести теоретичне дослідження існуючих алгоритмів, які можуть бути застосовані в даній системі, реалізувати їх і дослідити їх швидкодію;

4) Реалізувати розроблену криптографічну систему на кроці 2) з алгоритмами, які були досліджені і вибрані на кроці 3)

Об'єктом дослідження є процеси передачі, збереження та пошуку даних відеоспостереження.

Предметом дослідження є забезпечення конфіденційності та цілісності даних відеоспостереження, а також пошуку по зашифрованним даним відеоспостереження без їх розшифрування.

При розв'язанні поставлених завдань використовувались такі методи дослідження: методи лінійної та абстрактної алгебри, теорії кодування, теорії складності алгоритмів, методи комп'ютерного та статистичного моделювання

Наукова новизна полягає у створенні та оптимізації криптографічної системи для мікроконтролера ESP32. Дослідити можливості забезпечення цілісності та захисту даних у системах відеоспостереження з використанням обмежених ресурсів вказаного мікроконтролера.

Практичне значення полягає в забезпеченні безпечного використання передачі даних у пристроях, що побудовані на даному мікроконтролері. Система криптографічного захисту дозволить забезпечити конфіденційність даних. Шляхом використання різних алгоритмів шифрування буде захищена передача та збереження даних на мікроконтролері ESP32. Це гарантує, що лише авторизовані особи зможуть отримати доступ до цих даних, запобігаючи несанкціоновані перехоплення та доступ. Розроблена система дозволить виявляти будь-які зміни або спроби модифікації переданих даних.

1 ОПИС КРИПТОГРАФІЧНОЇ СИСТЕМИ І ХАРАКТЕРИСТИКИ МІКРОКОНТРОЛЕРА

Даний розділ об'єднує в собі теорію і теми, які мають бути описані перед початком реалізації. В підрозділі 1.1 буде описана основна ідея і вимоги для майбутньої програми і алгоритмів. В підрозділі 1.2 будуть розглянуті основні характеристики мікроконтролера, на якому буде реалізована дана система. В підрозділі 1.3 розглянемо схему криптографічного захисту даних при передачі відеодених з мікроконтролера на сервер і схему запису даних на енергонезалежну пам'ять мікроконтролера.

1.1 Загальний опис і вимоги для майбутньої системи

Системи відеоспостреження на мікроконтролерах досить ефективне рішення для спостереження за об'єктами, так як вони досить маленькі і не потребують багато електричної потужності. Але так як одним із можливих застосувань даних систем може бути забезпечення безпеки, то потрібен захист даної системи від втручань в її роботу. А саме, щоб було неможливо вивести з ладу, підмінити дані, або передавати дані з іншого пристрою, також важливим є те, щоб було неможливо отримувати та читати дані з даного пристрою. Дана задача звісно потребує комплексного підходу і потребує не тільки криптографічного захисту, однак в даній роботі буде розглянуто саме криптографічний захист і його реалізація.

В даному розділі розглянемо основні функціональні вимоги для майбутньої системи відоспостереження.

1) Можливість передавати кадри з мікроконтролера на сервер;

2) Зберігати дані на мікроконтролері – для можливості перевірити дані збережені на сервері, якщо доступ адміністратора сервера буде

скомпроментовано;

3) Зберігати дані на сервері – для можливості швидко отримати доступ до даних і переглянути з плином часу;

4) Перед початком передачі даних на сервер, має відбуватись автентифікація мікроконтролера, щоб була неможлива передача даних з іншого мікроконтролера;

5) Передача даних з мікроконтролера на сервер має відбуватись в зашифрованому вигляді(зберігання даних на сервері має бути також в зашифрованому вигляді) – для забезпечення захищеності даних під час передачі;

6) Зберігання даних на мікроконтролері має відбуватись з забезпеченням цілісності і можливістю відшукати кадр по мітці часу без попереднього розшифрування. Дана вимога потрібна, щоб дані створені на мікроконтролері були недоступні, для звичайних ситуацій, і могли використовуватись тільки у разі необхідності перевірити достовірність даних, якщо є підозра скомпроментованості прав адміністратора на сервері, або за потреби офіційних органів влади, отримати дані з даної системи. Тому цифровий ключ буде зберігатись на захищеній пам'яті мікроконтролера, щоб доступ до даних був можливий тільки, якщо напряму втрутитись у роботу мікроконтролера. Алгоритм пошуку по зашифрованим даним потрібен, так як у разі потреби конкретних даних, розшифрування може зайняти дуже довгий термін, тому пошук по розшифрованим даним не буде вихідом в даній ситуації;

Так як ми створюємо систему відеоспостереження, то потрібно забезпечити якнайбільшу кількість переданих кадрів в секунду, щоб мати більше інформації про об'єкт над яким відбувається спостереження. З попереднього досвіду, найблільша кількість кадрів які вдавалося передати по бездротовій мережі і відтворити - це 12 кадрів в секунду. Тому візьмемо це, як результат якого будемо намагатись наближатись і 1 кадр в секунду як найгірший можливий результат, при нижчому показнику потрібно буде вносити зміни в систему, або виконувати оптимізацію. Тому

нам потрібні будуть алгоритми шифрування, які будуть швидкими і не затримувати роботу відеопередачі. Також криптографічні алгоритми мають бути низькресурсними, так як система розроблюється для мікроконтролера, то існують обмеження по оперативній пам'яті і потужності процесора, які представлені в підрозділі 1.2

Попередній перелік вимог, окрім першого пункту, мають забезпечити мінімальний захист для системи відеоспостереження і вирішувати більшість задач покладені на дану систему. Однак, слід заначити, що кожен пункт буде сповільнювати виконання програми на мікроконтролері, тому потрібно буде по можливості оптимізувати роботу програмного забезпечення і системи.

1.2 Характеристики мікроконтролера ESP32

З розділу 1.1 маємо, що апаратні вимоги – це можливість отримувати дані з модуля камери, записувати дані на енергонезалежну пам'ять і передавати відеодані на сервер. Для цих цілей напрочуд чудово підходить модуль ESP32-CAM, який поєднує в собі мікроконтролер ESP32, модуль камери, модуль WI-FI для передачі даних на сервер і модуль для запису даних на micro-SD карту пам'яті. Розглянемо характеристики даного модуля, щоб надалі була можливість оцінювати потужність мікроконтролера і планувати майбутні покращення, які можливо буде зробити - Таблиця 1.1.[4]

В даному модулі не вбудований програматор, тому він буде підключений окремо. Для запису програми в пам'ять пристрою буде використовуватись модуль CP2102 (USB to TTL). Також даний модуль буде живити ESP32-CAM. Підключення програматора буде виконуватись через піни RX/TX - Таблиця 1.2.[6]

Реалізуємо прототип даної системи - Рисунок 1.1

Даний прототип має задовільняти всім вимогам апаратної частини, які були описані в підрозділі 1.1 та конкретизовані в підрозділі 1.2.

Таблиця 1.1 – Характеристики модуля

Контролер	ESP-32 32 біт
Частота процесора	160 МГц
Пам'ять	520 КБ SRAM
Напруга живлення	5В
Камера	OV2640
Роздільна здатність	2 Мп
Бездротовий зв'язок	WI-FI: 802.11 b/g/N
micro-SD карта	4 Гб
Інтерфейси які підтримує і використовуються	UART, SPI, I2C

Таблиця 1.2 – Підключення модуля CP2102

CP2102	ESP32-CAM
5V	5V
GND	GND
RX	TX
TX	RX

1.3 Опис схеми криптографічного захисту і елементи які мають бути використані

В розділі 1.1 були описані основні вимоги до криптографічного захисту. Можемо розділити цей захист на дві схеми – захист даних при передачі на сервер і захист даних при зберіганні на енергонезалежній пам'яті мікроконтролера.

Почнемо з схеми при передачі даних на сервер. Схема для передачі даних на сервер буде досить класична:

1) Автентифікація пристрою за допомогою передачі хеша логіна і пароля;

2) Передача зашифрованих даних;

Опишемо схему реалізовану в Рисунку 1.2 і наведену вище:

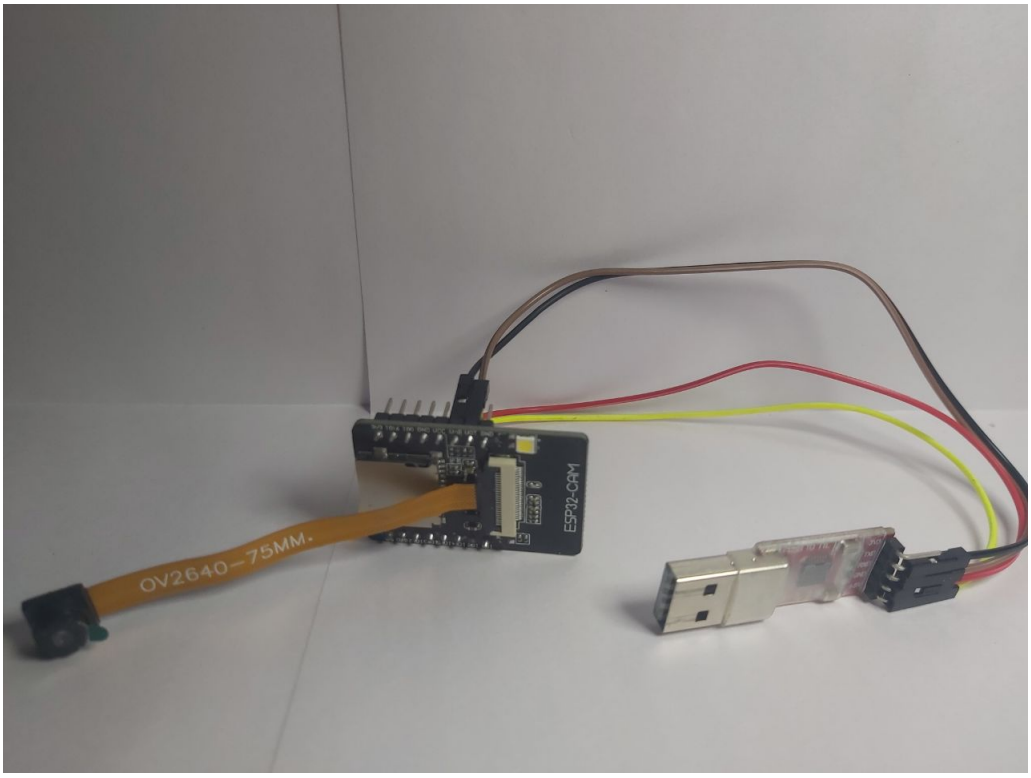

Рисунок 1.1 – Прототип апаратної частини системи відеоспостереження

Передача починається з того, що мікроконтролер передає хеш від login на сервер, сервер перевіряє чи є такий login в системі і у разі позитивного результату відправляє на мікроконтролер, що такий логін існує (якщо такого логіна не існує, то закінчується передача даних) (1). Після цього від мікроконтролера приходить зашифрований пароль(2), система розшифровує його і перевіряє, чи існує така зв'язка логіна і пароля(3). У разі позитивного результату, сервер відправляє підтвердження і мікроконтролер починає передачу даних(4). По завершеню, мікроконтролер відправляє повідомлення про закінчення передачі даних. На Рисунку 1.2 представлено, як виглядає передача даних і показані кроки.

Дана схема має запобігти можливості підробити потік даних від неавтентифікавоного пристрою. Також у разі автентифікації пристрою, але відсутності коректного ключа для шифрування, також буде помітно втручання зловмисника.

В блоці (4) має також відбуватись запис даних на енергонезалежну

пам'ять мікроконтролера. При чому запис має відбуватись, щоб дані неможливо було підробити, навіть адміністратору системи і був можливий пошук конкретних кадрів по мітці часу. Вказана проблема має бути вирішено в розділі 2, з представленим конкретним алгоритмом зберігання даних, забезпеченням їх цілісності і можливістю виконувати пошук по ним.

Окрім запису на мікроконтролері, повинні бути записані дані на самому сервері, щоб була можливість їх прочитати, маючи відповідний доступ. В блоці (5) має відбуватись запис даних на сервері в зашифрованому вигляді.

Висновки до розділу 1

В даному розділі були описані мінімальні вимоги для системи, яку потрібно реалізувати. Створений прототип, на якому будуть тестуватись всі подальші алгоритми. Також була описана криптографічна система, для подальшої розробки.

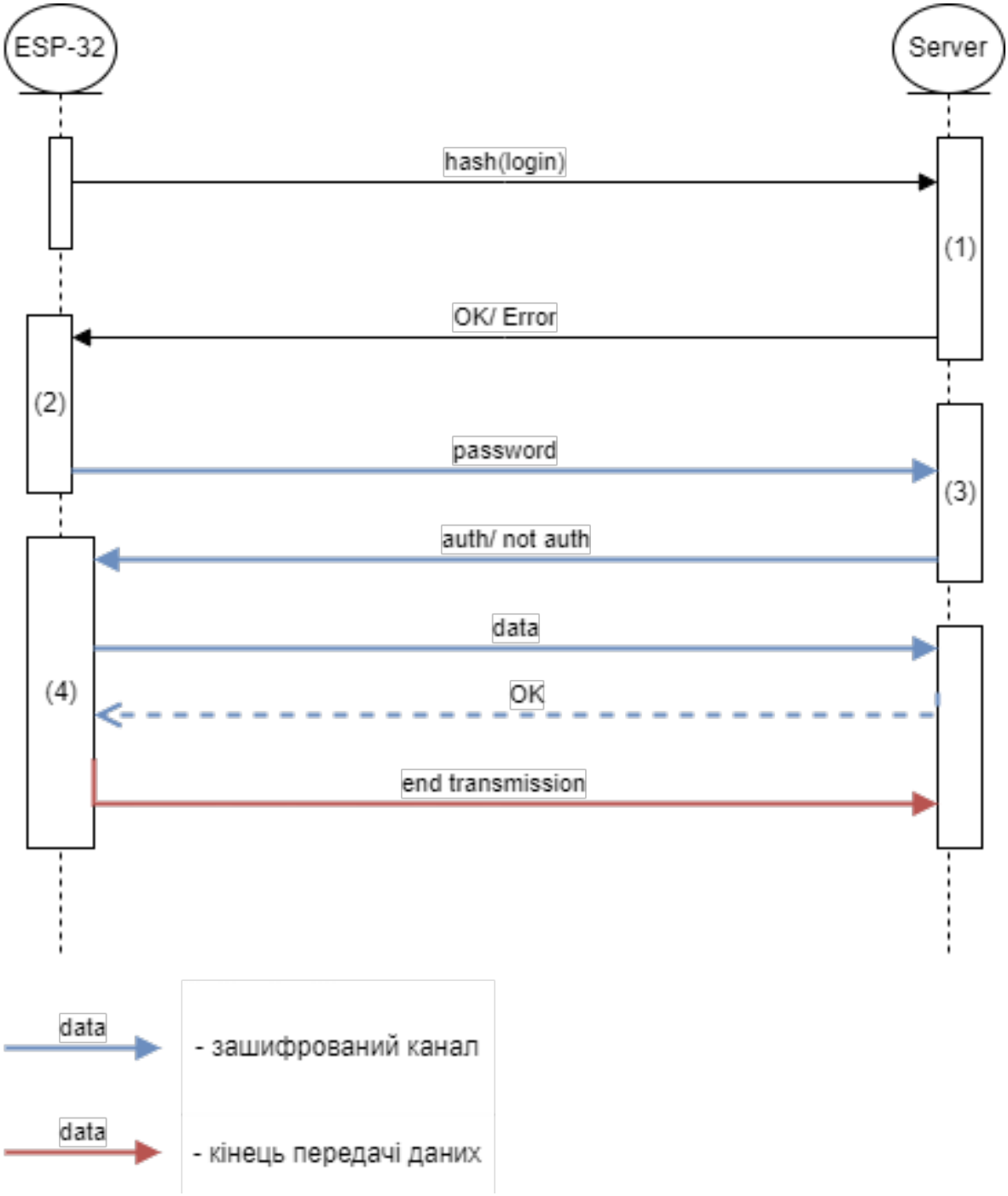

Рисунок 1.2 – Схема передачі даних

2 АНАЛІЗ ЕЛЕМЕНТІВ СИСТЕМИ

В другому розділі будуть розглянуті алгоритми роботи сервера і мікроконтролера в системі відеоспостереження. Також будуть вказані, які саме алгоритми шифрування мають використовуватись в даній системі та теоретичні відомості про побудову алгоритму пошуку.

2.1 Визначення алгоритму передачі даних на сервер і алгоритм роботи серверу

Опишемо більш формально розглянуту система в розділі 1.3, для того щоб ми могли дослідити швидкодію алгоритмів для даної системи. Почнемо з схеми для передачі даних з мікроконтролера на сервер: На мікроконтролері при прошивці програмного забезпечення також буде задавитись значення: login, password, key. Алгоритм який має виконуватись на мікроконтролері для передачі даних - Алгоритм 2.1

Алгоритм 2.1 Передача даних на сервер

1: $h = hash(login)$
2: якщо відповідь сервера == OK, то
    продовжити
3: інакше закінчити передачу
4: $p = Encrypt(key, password|h)$
5: відправити p на сервер
6: якщо відповідь сервера == OK, то
    продовжити
7: інакше закінчити передачу
8: поки не відбудеться збою в роботі циклу, або відповідь сервера == OK
    ph_i = зчитати дані з камери
    t_i = отримати мітку часу
    виконати запис в пам'ять phi і ti
    $m_i = Encrypt(key, ph_i|t_i)|h$
    відправити mi на сервер
    якщо відповідь сервера == OK, то продовжити
    якщо відповідь сервера == END, то закінчити передачу
    закінчити виконання

Алгоритм, що виконується на сервері для отримання даних з мікрокотролера - Алгоритм 2.2

Алгоритм 2.2 Роботи сервер

1: отримати h від мікроконтролера
2: якщо існує заданий логін в БД, то
створити нове підключення з вказаним h, password і key для нього
3: інакше повернути ERROR
4: отримати p від мікроконтролера
5: відправити p на сервер
6: з p отримати h
7: якщо існує з'єднання з вказаним h, то
зробити запис даних на сервері p
8: $ph_i|t_i = Decrypt(key, Encrypt(key, ph_i|t_i))$
9: вивести ph_i
10: інакше повернути мікороконтролеру ERROR

Як видно з вище наведеного алгоритму, потрібно описати функції hash() і Encrypt(). І якщо для функції hash() час виконання не є критичним, так як дана функція виконується тільки один раз за весь час, то час виконання функції Encrypt є досить критичним, як і алгоритм запису даних в пам'ять.

2.2 Алгоритми шифрування

Для виконання функції шифрування Encrypt(), що використовується в алгоритмах, які описані в розділі 2.1 потрібно використовуватись досить швидкі алгоритми на обмежених ресурсах. Для цього може використовуватись легковагова криптографія або алгоритми шифрування для яких є апартне прискорення на мікроконтролері. Так як ми можемо записати ключ шифрування на мікроконтролері і на сервері, то будемо використати алгоритми симетричного шифрування.

В легковаговій криптографії є ARX алгоритми, тобто алгоритми які виконуються за рахунок використання операцій: додавання по модулю, побітового зсуву і XOR. Так як, вказані операції є дуже швидкими, то саме такі алгоритми будуть використовуватись в наступних розділах при

аналізі швидкості алгоритмів шифрування.[5]

Алгоритм AES128 має апаратне прискорення в мікроконтролері ESP-32, тому він також може бути проаналізований на швидкість шифрування в наступних розділах.

2.3 Аналіз побудови алгоритму пошуку по зашифрованим даним

В даному подрозділі буде розглянуті можливі властивості для алгоритму шифрування для того щоб виконувались вимоги 6) з підрозідіу 1.1. Тобто, має функціонувати алгоритм, який буде зберігати дані на мікроконтролері, до яких буде доступ, тільки у разі фізичного втручання в роботу системи і можна буде отримати доступ до конретного кадру, за допомогою алгоритму пошуку по зашифрованим даним. Для цього потрібно буде реалізувати цифровий підпис відеоданих і ключ для підпису цих даних має зберігатись в захищеній пам'яті мікроконтролера і доступ до нього не буде ні в кого.

Для побудови алгоритму пошуку по зашифрованим даним, нам потрібна можливість виконувати математичні дій над даними без їх попереднього розшифрування. Для цього може бути використане гомоморфне шифрування.

Означення 2.1 Алгоритм шифрування $E()$ є гомоморфним, якщо для $E(x)$ і $E(y)$ ми можемо обчислити $E(x \bullet y)$, не маючи значень x та y для деякого оператора $\bullet$. [8]

Гомоморфні алгоритми поділяються на:

- часткове гомоморфне шифрування
- повністю гомоморфне шифрування[10]

Часткове гомоморфне шифрування – шифрування яке гомоморфне відносно тільки однієї операції – додавання або множення. Повністю гомоморфне шифрування – шифрування яке гомоморфне відносно

операції і додавання, і множення.[10]

Так як нам не потрібно робити складні обчислення над зашифрованими даними, то ми можемо реалізувати пошук тільки за рахунок операції множення. Для цього будемо використовувати алгоритм шифрування і цифрового підпису RSA. Цей алгоритм є частково гомоморфним відносно операції множення і має апаратне прискорення на мікроконтролері ESP32, що дає можливість нам використовувати його для шифрування і пошуку по зашифрованим даним.[7] - Алгоритм 2.3

Алгоритм 2.3 RSA

1: p, q – обрати два простих числа
2: $n = pq$
3: $d = e^{-1} mod \varphi(n)$
4: (e, n) – відкритий ключ
5: (d, n) – секретний ключ

Гомоморфічні властивості RSA: Нехай m_1 і m_2 – відкритий текст, s_1 і s_2 – шифротекст, тоді

$s_1 s_2 = (m_1^e mod n)(m_2^e mod n) = (m_1 m_2)^e mod n$

Отже, ми можемо використовувати даний алгоритм для пошуку даних, конкретний алгоритм запису даних і пошуку по ним описаний в розділі 3.2.

Висновки до розділу 2

В даному розділі були описані властивості алгоритмів і їх теоретичні означення, які мають використовуватись при реалізації алгоритмів шифрування, побудові алгоритму запису даних і пошуку по зашифрованим даним.

3 ОПИС РІШЕНЬ

В даному розділі будуть розглянуті конкретні алгоритми шифрування і їх аналіз. Також буде описаний алгоритм запису даних на енергонезалежну пам'ять і алгоритм пошуку по зашифрованим даним. Окрмі цього продемонстрована реалізація системи, код якої наведений в Додатку А і майбутні можливі покращення даної системи.

3.1 Аналіз швидкості виконання алгоритмів симетричного шифрування

Для створення шифрованого каналу, будуть використовуватись симетричні шифри. Для аналізу вибрав легковагові шифри: TEA, Speck, LEA. Розглянемо алгоритми даних шифрів і напишемо програми для їх еталонних реалізацій (алгоритми які були описані в статтях їх винахідників).

Почнемо з алгоритма TEA(Tiny Encryption Algorithm) – блочний симетричний алгоритм шифрування.[9] - Алгоритм 3.1

Алгоритм Speck – блочний симетричний алгоритм шифрування. Так як даний алгоритм підтримує різні розміри блока і ключа, то в даній реалізації я використав блок довжиною 128 біт і довжина ключа 128 біт.[2] - Алгоритм 3.2

Алгоритм LEA - Lightweight Encryption Algorithm – блочний симетричний алгоритм штфрування. З довжиною блоку 128 біт і довжиною ключа 192 біт.[3] - Алгоритм 3.3

Так як для AES-128 в мікроконтролері ESP32 передбачене апаратне прискорення, то проаналізуємо і його. Ключ повинен буде мати довжину 128 біт(16 байт). Відкрите повідомлення має бути розбите на блоки по 128 біт (16 байт).[1] - Алгоритм 3.4 Тепер проаналізуємо дані алгоритми по

Алгоритм 3.1 TEA шифрування

Вхід:
K — $k[0], k[1], k[2], k[3]$ - ключ
V — $v[0], v[1]$ – відкритий текст
Вихід:

V – $v[0], v[1]$ – шифротекст
$y = V[0]$
$x = V[1]$
$delta = 0x9e3779b9$
$sum = 0$
$i = 0$
Поки $i! = 32$ виконувати
    $sum+ = delta$
    $y+ = ((x << 4) + k[0]) \oplus (x + sum) \oplus ((x >> 5) + k[1])$
    $x+ = ((y << 4) + k[2]) \oplus (y + sum) \oplus ((y >> 5) + k[2])$
    $i+ = 1$
$v[0] = y$
$v[1] = z$

Алгоритм 3.2 Speck шифрування

Вхід:
K — $k[0], k[1], k[2], k[3]$ - ключ кожен елемент 64 бітне слово
V — $v[0], v[1]$ – відкритий текст, де кожен елемент предсталяє собою слово довжиною 64 біта
Вихід:
V – $v[0], v[1]$ – шифротекст

$y = V[0]$
$x = V[1]$
функція $RR(x, y)$:
    $((x >> r)OR(x << (64 - y))$
функція RL(x, y):
    $((x << r)OR(x >> (64 - y))$
$x = (RR(x, 8) + y) \oplus k[1]$
$y = (RL(y, 3)) \oplus x$
Поки $i! = 32$ виконувати
    $k[0] = (RR(k[0], 8) + k[1]) \oplus i$
    $k[1] = (RL(k[1], 3)) \oplus k[0]$
    $x = (RR(x, 8) + y) \oplus k[1]$
    $y = (RL(y, 3)) \oplus x$
    $i+ = 1$
$v[0] = y$
$v[1] = x$

Алгоритм 3.3 LEA шифрування

1: **Вхід:**
2: K — $k[0], k[1], k[2], k[3], k[4], k[5]$ - ключ
3: V — $v[0], v[1], v[2], v[3]$ – відкритий текст, де довжина кожного елементу 32 біта
4: **Вихід:**
5: C – $C[0], C[1], C[2], C[3]$ – шифротекст
6:
7: $x[0][0], x[0][1], x[0][2], x[0][3] = v[0], v[1], v[2], v[3]$
8: $i = 0$
9: Поки $i! = 28$ виконувати
$x[i+1][0] = ((x[i][0] \oplus k[i][0]) + (x[i][1] \oplus k[i][1])) <<< 9$
$x[i+1][1] = ((x[i][1] \oplus x[i][2]) + (x[i][2] \oplus k[i][3])) >>> 5$
$x[i+1][2] = ((x[i][2] \oplus x[i][4]) + (x[i][3] \oplus k[i][5])) >>> 3$
$x[i+1][3] = x[i][0]$
$i+ = 1$
10: $c[0], c[1], c[2], c[3] = x[28][0], x[28][1], x[28][2], x[28][3]$

Алгоритм 3.4 AES-128

1: M - відкрите повідомлення(16 байт) представлене у вигляді 32 16-розрядних чисел – (m_1, m_2, . . . , m_{32}).
2: Виконується розширення ключа
3: Додавання ключа до відкритого повідомлення
4: Перетворення в матрицю 4 на 4, де кожен елемент це 2 16-розрядних числа:

$$\begin{pmatrix} m_1m_2 & m_3m_4 & m_5m_6 & m_7m_8 \\ m_9m_{10} & m_{11}m_{12} & m_{13}m_{14} & m_{15}m_{16} \\ m_{17}m_{18} & m_{19}m_{20} & m_{21}m_{22} & m_{23}m_{24} \\ m_{25}m_{26} & m_{27}m_{28} & m_{29}m_{30} & m_{31}m_{32} \end{pmatrix},$$

5: Для і від 0 до 9 виконуються наступні функції:
SubBytes
ShiftRows
MixColumns
AddRoundKey
6: SubBytes
7: ShiftRows
8: AddRoundKey

часу виконання їх для шифрування одного кадру і оберемо один з них.

Таблиця 3.1 – Час виконання алгоритмів

Алгоритм	Час виконання (мкс)
TEA	924
Speck	1386
LEA	3696
AES	8470

Як видно з Таблиці 3.1, то час виконання даних алгоритмів варіюється від 0.9 мс до 8.5 мс. Так як в нашому випадку ми передаємо 2464 байтів на сервер, то ми можемо обрати найстійкіший алгоритм криптографічного захисту з розглянутих і це не дуже вплине на швидкість виконання нашої програми. Однак варто врахувати, що при передачі більших об'ємів відеоданих, потрібно буде змінити алгоритм, для того щоб збільшити кількість кадрів, які можуть бути передані на сервер, в секудну.

3.2 Алгоритм запису даних на мікроконтролері і пошуку по ним

В даному розділі буде побудований конкретний алгоритм, теоретичні відомості якого описані в розділі 2.3

Дані, які будуть зберігатись на мікроконтролері: m – дані з камери. t – мітка часу

Створюємо відкритий ключ(e_1, n_1)

Сторюємо відкритий ключ (e_2, n_2)

Створюємо секретний ключ(d_1, n_1)

Створюємо секретний ключ(d_2, n_2) - саме цей ключ буде зберігатись в захищеній пам'яті мікроконтролера.

$s_1 = t^{e_1} mod n_1$

$s_2 = m^{d_2} mod n_2$

$cadr = s_1 | s_2$

Файл буде складатись з: $cadr_1$

$cadr_2$

...

$cadr_n$

Далі наведемо алгоритм пошуку по зашифрованим даним - Алгоритм 3.5, який має знаходити конкретний $cadr$ по мітці часу t.

Алгоритм 3.5 Алгоритм пошуку

Вхід:
t — мітка часу яку ми маємо знайти
Вихід:
s_{2i} — кадр якому відповідає конкретна мітка часу

Знайдемо t^{-1}
$s_i = (t^{-1})^e mod n_1$
Для і від 1 до n:
    Якщо $s_{1i} s_i = 1$, то закінчити цикл і повернути s_{2i} як шукане

Для числа t завжди буде існувати обернений, якщо воно є взаємнопросте з n. Так як, $n = pq$, то якщо t = p, або t = q, не буде існувати оберненого, але дані проблеми можна обійти при реалізації, наприклад збільшенням t на одиницю.

3.3 Прототип програмного забезпечення

Так як, були розглянуті і проаналізовані всі алгоритми і були повністю розроблені всі схеми, то можемо розглянути прототип програмного забезпечення. В даному розділі буде розглянута демонстрація, програми яка реалізована в Додатку А.1 – програма, яка має бути зашита на мікрокотролер. В Додатку А.2 – представлена програма для серверу. Дане програмне забезпечення включає в себе

алгоритми, які були описані в попередніх розділах. Нажаль, реалізація, може бути розглянена тільки у вигляді рисунків.

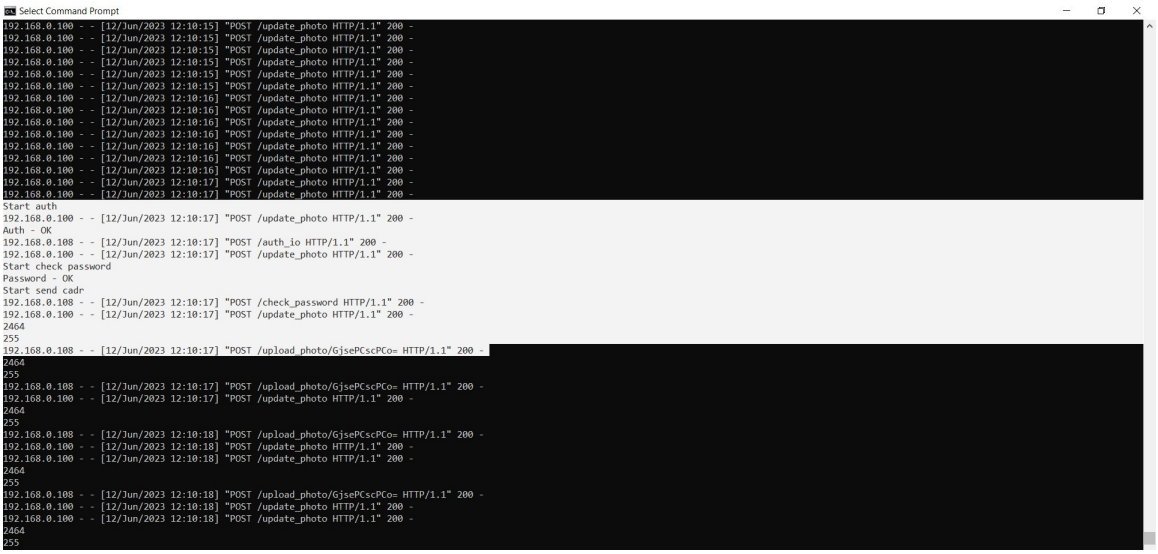

Рисунок 3.1 – Запуск сервера

На Рисунку 3.1 показаний запуск сервера, автентифікація і передача даних. З даного Рисунку можемо побачити, що дані приходять на сервер 1–3 рази в секунду.

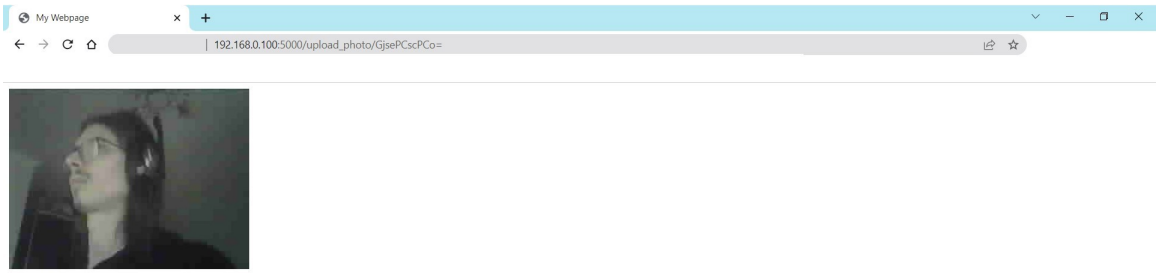

Рисунок 3.2 – Графічне зображення інформації

Для виводу графічних даних з серверу, використовується браузер.

Для цього був створена сторінка, яка оновлює її кожні 100 мс.

Даний прототип демонструє роботу схеми, яка описана була в попередніх підрозділах.

3.4 Можливі покращення

В даному розділі розглянемо, як можна покращити реалізацію в майбутньому і які оптимізації можна провести.

Один із напрямів, які слід покращити, це можливість отримувати більше кадрів в секунду і обробляти дані більшого розміру, тобто щоб зображення було кращої якості. Для цього потрібно зменшити час виконання алгоритму запису даних на мікроконтролер і час шифрування даних при передачі. Для цього, можна використати ще один мікроконтролер: один мікроконтролер буде займатись зчитуванням даних і передачею їх на сервер, а інший буде отримувати ці дані по інтерфейсу I2C і займатись зберіганням цих даних на енергонезалежну пам'ять. Звісно така система збільшить споживання електроенергії, якщо ми захочимо живити її від акамулятора, але збільшить продуктивність.

Також один з можливих напрямів, який можна оптимізувати - це швидкість передачі по WI-FI від мікроконтролера до сервера. Для цього можна використати, кращу антену, або зробити передачу даних на сервер по дроту.

Для покращення криптографічного захисту, потрібно розробити захист від атак по стороньому каналу, захист від атаки «людина посередині» , які наразі в даній системі одні із самих небезпечних. Однак, дані покращення можуть бути виконані в наступних версіях даної системи.

Висновки до розділу 3

В даному розділі були представлені алгоритми для передачі даних на сервер, проаналізовані алгоритми симетричного шифрування і представлений алгоритм пошуку по зашифрованим даним. Також була продемонстрований прототип данох системи в підрозділі 3.3 і розглянута подальша робота над данною системою.

ВИСНОВКИ

В даній роботі я мав виконати наступні задачі: 1) Аналіз задач захисту інформації, що виникають при створенні систем захищеного спостереження. 2) Аналіз та вибір криптографічних алгоритмів, які доцільно застосовувати для забезпечення конфіденційності та цілісності інформації в системі захищеного відеоспостереження. 3) Розробка криптопротоколу для забезпечення цілісності та конфіденційності даних відеоспостереження, що зберігаються. 4) Розробка алгоритму пошуку по зашифрованним даним без розшифрування із використанням гомоморфних криптоалгоритмів. 5) Реалізація програмно-апаратного макету системи захищеного відеоспостереження на базі мікроконтролеру ESP32. Був проведений стислий аналіз до захисту інформації в системах відеоспостереження на мікрокотролерах і сформовані основні вимоги до системи в підрозділі 1.1. Була розроблена система в якій відбувається аутентифікація мікроконтролера на сервері, передача зашифрованих даних і розшифрування їх на сервері і зберігання кадрів на мікрокотролері з можливістю пошуку по ним і підтримкою цілісності цих даних. Огляд даної системи представлений в підрозділі 1.3 і повністю описаний в підрозділі 2.1. Проведене дослідження існуючих алгоритмів шифрування, які можуть бути застосовані в даній системі в підрозділі 2.2 і проаналізована швидкість виконання в підрозділі 3.1. Побудований алгоритм пошуку по зашифрованим даним, за допомогою гомоморфних властивостей шифрів в підрозіділі 2.3 і повністю описаний в підрозділі 3.2. Реалізацію даної системи для передачі відеоданих з мікроконтролера на сервер можна побачити в Додатку А.1 і Додатку А.2. Також демонстрація даної програми можна побачити в розділі 3.3.

Дану систему можна використовувати, як базовий захист для системи відеоспостереження з розглянутим криптографічним захистом і пошуком по зашифрованим даним за допомогою гомоморфних властивостей шифру.

ПЕРЕЛІК ПОСИЛАНЬ

[1] FIPS 197. *Advanced Encryption Standard*. Англ. 2001. URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf.

[2] Voronkov A. Bjørner N. Virbitskaite I. *Perspectives of System Informatics 12th International*. Англ. 2019.

[3] Dong-Geon Lee Daesung Kwon Kwon Ho Ryu. *LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors*. Англ. 2014.

[4] EspressifSystems. *ESP32 Datasheet*. Англ. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[5] Daniel J. Bernstein Jean-Philippe Aumasson. *SipHash: a fast short-input PRF*. Англ. 2012. URL: https://web.archive.org/web/20200312053222/.

[6] Silicon Laboratories. *CP2102 Datasheet*. Англ. 2007. URL: https://pdf1.alldatasheet.com/datasheet-pdf/view/201067/SILABS/CP2102.html.

[7] Vanstone S. A. Menezes A. J. van Oorschot P. C. *Handbook of Applied Cryptography*. Англ., с. 434.

[8] Ron Rivest. *Lecture Notes15: Voting, Homomorphic Encryption*. Англ. 2002. URL: http://web.mit.edu/6.857/OldStuff/Fall02/handouts/L15-voting.pdf.

[9] David J. Wheeler Roger M. Needham. **?**TEA, a Tiny Encryption Algorithm**?** Англ. В: (1994). URL: https://www.cix.co.uk/~klockstone/tea.pdf.

[10] Анна Ільєнко. *СУЧАСНІ МЕТОДИ ГОМОМОРФНОГО ШИФРУВАННЯ ІНФОРМАЦІЙНИХ РЕСУРСІВ*. 2015. URL: https://web.archive.org/web/20171215225641/.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 Програма 1

Текст програми, яка зашивається на мікроконтролера.

```
#pragma GCC target ("aes")
#include <mbedtls/aes.h>
#include <esp_timer.h>
#include <stdlib.h>
#include <stdio.h>
#include "mbedtls/md.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include <FS.h>
#include <SPIFFS.h>
#include <SD.h>
#include <mbedtls/pk.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <CRC32.h>
#include <esp_timer.h>
#include <mbedtls/sha256.h>
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

#include <stdlib.h>
#include <stdio.h>
#include "esp_camera.h"
#include "esp_wifi.h"
```

```
#include "esp_system.h"
#include "esp_event_loop.h"
#include "esp_log.h"

#include <esp_timer.h>

#define PRIVATE_KEY_PEM "-----BEGIN PRIVATE KEY-----\n"\
                        "-----END PRIVATE KEY-----\n"

CRC32 crc;

void calculateSHA256(const uint8_t* data, size_t dataSize,
   uint8_t* sha256Hash) {
  mbedtls_sha256_context ctx;
  mbedtls_sha256_init(&ctx);
  mbedtls_sha256_starts(&ctx, 0);
  mbedtls_sha256_update(&ctx, data, dataSize);
  mbedtls_sha256_finish(&ctx, sha256Hash);
  mbedtls_sha256_free(&ctx);
}

void generate_signature(const unsigned char* message, size_t
   message_len, unsigned char* signature, size_t* signature_len
   ) {
    mbedtls_pk_context pk;
    mbedtls_pk_init(&pk);

    mbedtls_pk_parse_key(&pk, (const unsigned char*)
   PRIVATE_KEY_PEM, strlen(PRIVATE_KEY_PEM) + 1, NULL, 0);
```

```
    mbedtls_pk_sign(&pk, MBEDTLS_MD_NONE, message, message_len,
    signature, signature_len, NULL, NULL);

    mbedtls_pk_free(&pk);
}

void getEsp32Id(uint8_t* macAddress) {
    esp_efuse_mac_get_default(macAddress);
}

const char* ssid = "WI-FI_NAME";
const char* password = "PASSWORD";
const char* serverUrl = "http://192.168.0.100:5000/auth_io";
const char* url_send_pwd = "http://192.168.0.100:5000/
   check_password";
const char* url_send_video = "http://192.168.0.100:5000/
   upload_photo/GjsePCscPCo=";
String hashString;


void initCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
```

```
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_QVGA;
  config.jpeg_quality = 40;
  config.fb_count = 1;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
}




void setup() {
  Serial.begin(115200);
  delay(3000);
  WiFi.begin(ssid, password);
```

```
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Start connect to WiFi...");
}
Serial.println("Connect to WiFi");
char *payload = "GjsePCscPCo=";
byte shaResult[32];

mbedtls_md_context_t ctx;
mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;

const size_t payloadLength = strlen(payload);

mbedtls_md_init(&ctx);
mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 0)
 ;
mbedtls_md_starts(&ctx);
mbedtls_md_update(&ctx, (const unsigned char *) payload,
 payloadLength);
mbedtls_md_finish(&ctx, shaResult);
mbedtls_md_free(&ctx);
for(int i= 0; i< sizeof(shaResult); i++)
{
char str[3];
sprintf(str, "%02x", (int)shaResult[i]);
Serial.print(str);
hashString += str;
}

initCamera();
```

```
  esp_timer_create_args_t timer_args;
  timer_args.callback = NULL;
  esp_timer_handle_t timer;
  esp_timer_create(&timer_args, &timer);

   if (!SPIFFS.begin(true)) {
    Serial.println("Error with SPIFFS");
    return;
  }

  Serial.println("SPIFFS and SD card initialized");
  createFile();

}

File file;
String current_name_file = "/data.txt";

void writeData(uint8_t *data_){
  file = SPIFFS.open(current_name_file, FILE_APPEND);
  if (!file) {
    Serial.println("Failed to open file for writing");
    return;
  }


  uint8_t signature[MBEDTLS_MPI_MAX_SIZE];
  size_t signature_len;
```

```
  generate_signature((uint8_t*)data_, sizeof(data_), signature,
    &signature_len);
  file.write(signature, sizeof(signature));
  file.print('\n');
  file.close();
  Serial.println(current_name_file);
  Serial.println("Data written to file");
}


void createFile(){
  file = SPIFFS.open(current_name_file, FILE_WRITE);
  if (!file) {
    Serial.println("Failed to create file");
    return;
  }
  uint8_t esp32Id[6];
  getEsp32Id(esp32Id);
  file.write(esp32Id, sizeof(esp32Id));
  file.print('\n');
  file.close();

  Serial.println("Write to file");
}

void readData() {
  file = SPIFFS.open(current_name_file);
  if (!file) {
    Serial.println("Failed to open file for reading");
    return;
  }
```

```
  while (file.available()) {
    Serial.print((uint8_t)file.read());
  }
  Serial.println();

  file.close();
}

void closeFile(){
  file = SPIFFS.open(current_name_file);
  if (!file) {
    Serial.println("Failed to close file");
    return;
  }
  while (file.available()) {
    uint8_t byte_ = (uint8_t)file.read();
    crc.update(byte_);

  }
  uint32_t crc_ = crc.finalize();
  uint8_t crc_8[4];

  crc_8[0] = (crc_ >> 24) & 0xFF;
  crc_8[1] = (crc_ >> 16) & 0xFF;
  crc_8[2] = (crc_ >> 8) & 0xFF;
  crc_8[3] = crc_ & 0xFF;
  writeData(crc_8);
  file.close();
}
int a = 0;
```

```
bool start_send = false;
WiFiClient client;
HTTPClient http;
HTTPClient http2;

uint8_t key[16] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0
   xEF, 0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10};



void loop(){
  a+=1;
  int32_t delta = 0x9e3779b9;
  uint32_t sum_[32] = {};
  uint32_t delta_ = 0x9e3779b9;
  uint32_t start_count = 0;

  mbedtls_aes_context aes;
   mbedtls_aes_init(&aes);
   mbedtls_aes_setkey_enc(&aes, key, 128);

  for(int i = 0; i < 32; i++){
    start_count += delta_;
    sum_[i] = start_count;
  }

    uint8_t password[8] = {0x4567, 0xaabb, 0xdd22, 0xff11, 0
   x4567, 0xaabb, 0xdd22, 0xff11};
    uint8_t login_[8] = {0x1A, 0x3B, 0x1E, 0x3C, 0x2B, 0x1C, 0
   x3C, 0x2A};
```

```
  uint32_t key[4] = {0x01020304, 0x05060708, 0x090A0B0C, 0
x0D0E0F00};
  http.begin(client, serverUrl);
  http.addHeader("Content-Type", "image/jpeg");
  int httpResponseCode = http.POST(hashString);
  String response_ = http.getString();
  Serial.println(response_);
  if (httpResponseCode == HTTP_CODE_OK) {
  if(response_ == "auth"){
      http2.begin(client, url_send_pwd);
      http2.addHeader("Content-Type", "image/jpeg");
      encrypt(password, key, 8);
      int httpResponseCode = http2.POST((uint8_t *)password,
8);
      String response_2 = http2.getString();
      Serial.println(response_2);
      if(response_2 == "start_send"){
          start_send = true;
      }
  }

  Serial.print("Server: ");
  Serial.println(response_);
  http2.end();
  http.end();
  }else {
  Serial.print("Error auth");
}
  http.begin(client, url_send_video);
  while(start_send){
    camera_fb_t * fb = esp_camera_fb_get();
```

```
while(!fb) {
  fb = esp_camera_fb_get();
  if (fb) {
    break;
  }
}
```

```
 size_t originalLength = fb->len;
 size_t bufSize = originalLength + (16 - originalLength%16) +
  48;
 uint8_t* extendedBuffer = new uint8_t[bufSize];
 uint8_t* ciphertext = new uint8_t[bufSize];
 memcpy(extendedBuffer, fb->buf, originalLength);
 memset(extendedBuffer + originalLength, 0, bufSize -
originalLength - 48);

 unsigned long current_time = millis();

uint8_t time_bytes[10];

time_bytes[0] = (current_time >> 24) & 0xFF;
time_bytes[1] = (current_time >> 16) & 0xFF;
time_bytes[2] = (current_time >> 8) & 0xFF;
time_bytes[3] = current_time & 0xFF;

uint8_t esp32Id[6];
getEsp32Id(esp32Id);
for(int i = 4; i < 10; i++){
```

```
  time_bytes[i] = esp32Id[i-4];
}
uint8_t res_sha256[32];
calculateSHA256(time_bytes, 10, res_sha256);

for(int count = 0; count < 32; count++){
  extendedBuffer[bufSize-40+count] = res_sha256[count];
}



 for(int i =0; i < 8; i++){
 extendedBuffer[bufSize - 8 + i] = login_[i];
 }
 Serial.println(extendedBuffer[2], HEX);
 Serial.println(extendedBuffer[bufSize-2], HEX);

for(int N = 0; N < bufSize; N+=16){
    mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT,
 extendedBuffer + N, ciphertext + N);
 }

http.addHeader("Content-Type", "image/jpeg");
int httpResponseCode = http.POST((uint8_t*)ciphertext,
 bufSize);
Serial.println(httpResponseCode);
writeData(ciphertext);
 if(a%10 == 0){
  closeFile();
  current_name_file = "/data" + String(a/10) + ".txt";
  readData();
  createFile();
```

```
  }

  esp_camera_fb_return(fb);
  delete[] extendedBuffer;

  }
    http.end();
}
```

A.2 Програма 2

Текст програми для серверу

```
from flask import Flask, render_template, Response, request,
   jsonify
import base64
import socket
import hashlib
import time
import random
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
logins = {b'GjsePCscPCo=': 'GjsePCscPCo='}
data = {'GjsePCscPCo=':'Z7siEWe7IhE='}
data_logins = {'GjsePCscPCo=': 'Z7siEWe7IhE='}
data_keys = {'Z7siEWe7IhE=': [0x01020304, 0x05060708, 0
   x090A0B0C, 0x0D0E0F00]}
sessions = []

def decrypt_aes(ciphertext, key):
    cipher = AES.new(key, AES.MODE_ECB)
```

```
    decrypted_data = cipher.decrypt(ciphertext)
    print(decrypted_data[0])
    #original_data = unpad(decrypted_data, AES.block_size)
    #print(original_data[0])
    return decrypted_data

app = Flask(__name__, template_folder = "templates")

encode_photo = None

@app.route('/')
def index():
    global encode_photo
    return 'Oks'

@app.route('/upload_photo/<lgn>', methods=['GET'])
def upload_photo_2(lgn):
    if lgn in sessions:
        global encode_photo
        return render_template('index.html', photo=encode_photo
  )
    else:
        print('Not auth')
        return "Error: not auth"

@app.route('/upload_photo/<lgn>', methods=['POST'])
def upload_photo(lgn):
    if lgn in sessions:
        global encode_photo
        photo_data = request.get_data()
        session_id = []
```

```
            print(len(photo_data))
            key = bytes([0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD,
    0xEF, 0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10])
            decrypted_data = decrypt_aes(photo_data, key)

            login_message = decrypted_data[len(decrypted_data)-8:]
            encode_login = base64.b64encode(login_message).decode('
    utf-8')
            if encode_login == lgn:
                encode_photo = base64.b64encode(decrypted_data
    [:-48]).decode('utf-8')
            else:
                return "Error: not auth"
            return "Ok"
        else:
            print('Not auth')
            return "Error: not auth"

@app.route('/auth_io', methods = ['POST'])
def auth():
    print("Start auth")
    global sessions
    auth_code = request.get_data()
    #print(base64.b64encode(auth_code).decode('utf-8'))
    for lg in logins.keys():
        m = hashlib.sha256(lg).hexdigest()
        #print(m)
        #print(str(auth_code)[2:].replace("'", ""))
        if str(auth_code)[2:].replace("'", "") == str(m):
            print('Auth - OK')
            sessions.append(logins[lg])
```

```
            return 'auth'
        else:
            print('error')
            return 'error'

@app.route('/check_password', methods = ['POST'])
def check_pwd():
    print("Start check password")
    pwd_data = request.get_data()
    login = sessions[-1]
    password = data_logins[login]
    key = data_keys[password]

    decrypt_pwd = decrypt(pwd_data, key)
    encode_pwd = base64.b64encode(decrypt_pwd).decode('utf-8')

    if password == str(encode_pwd):
        print("Password - OK")
        print("Start send cadr")
        return 'start_send'
    else:
        return 'error_send'

@app.route('/update_photo', methods=['POST'])
def update_photo():
    global encode_photo
    return jsonify({"photoESP":encode_photo})

if __name__ == '__main__':

    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
    s.connect(('8.8.8.8', 80))
    ip_address = s.getsockname()[0]
    s.close()
    app.run(host=ip_address, port=5000, debug=True)
```

A.3 Програма 3

Текст програми для відображення графічних даних

```
<!DOCTYPE html>
<html>
<head>
  <title>My Webpage</title>
</head>
<body>
  <img src="{{photo}}" id = "photoESP"/>
</body>
<script>

  function getData(){
  var xhttp = new XMLHttpRequest();

  xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var response = JSON.parse(xhttp.responseText);
    console.log(response);
    document.getElementById("photoESP").src = "data:image/jpeg;
   base64,"+response.photoESP;
    }
  };
```

```
  xhttp.open("POST", "http://192.168.0.100:5000/update_photo",
   true);
  xhttp.send();
  }

  setInterval(getData, 150);

</script>
</html>
```