Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод та програмно-технічний засіб генерування ключів аутентифікації»

КвРКІП. 013042.17.01.01 ПЗ

Виконав: студент 2 курсу, група КІ2м-21-1 _____ Томусяк А.В
Підпис Ініціали, прізвище

Керівник кандидат техн. наук, доцент _____ Іванов О..В.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
10 05 2023 р.

Хмельницький, 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Томусяку Андрію Валерійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод та програмно-технічний засіб генерування ключів аутентифікації

Керівник проекту (роботи) Іванов О.В., к.т.н., доцент
Прізвище, ім`я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 09.01.2023 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих методів виявлення уразливості пристроїв при генеруванні ключів аутентифікації

Модель процесу генерування ключів аутентифікації та проектування апаратної частини

Процес розробки архітектури і сценарію метода та програмно-технічного засобу

Результати та аналіз розроблених методу та програмно-технічного засобу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання
Нормоконтроль	Лисенко С.М., професор кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання « 06 » 09 2022р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Пр
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	05.09.2022	ви
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2022	ви
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2022	ви
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2022	ви
5	Робота над науковою статтею	05.01.2023	ви
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2022	ви
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2023	ви
8	Оформлення пояснювальної записки згідно вимог	15.04.2023	ви
9	Попередній захист ДРМ	18.04.2023	ви
10	Захист ДРМ на засіданні ЕК	До 10.05.2023	

Студент _____ А.В. Томусяк
Підпис Ініціали, прізви

Керівник роботи _____ О.В. Іванов
Підпис Ініціали, прізви

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Метод та програмно-технічний засіб генерування ключів аутентифікації.

Автор роботи: Томусяк А.В.

Керівник роботи: Іванов О.В.

Пояснювальна записка: 85 с., 22 рис., 7 табл., 3 дод., 50 джерел.

АУТЕНТИФІКАЦІЯ, БЕЗПЕКА, ГЕНЕРАЦІЯ КЛЮЧІВ, КОМП'ЮТЕРНІ СИСТЕМИ, КРИПТОГРАФІЯ, ШИФРУВАННЯ, ЗАХИСТ ДАНИХ,ЗАХИЩЕНІ СИСТЕМИ.

Об'єктом дослідження є процес створення методу та програмно-технічного засобу генерування ключів аутентифікації.

Предметом дослідження є модель, метод та програмно - технічний засіб генерування ключів аутентифікації.

Метою кваліфікаційної роботи магістра є процес створення методу та програмно – технічного засобу генерування ключів аутентифікації.

Для розв'язання поставлених задач використовувалися методи :

1. розробка протоколу зв'язку для безпечного та швидкого з'єднання між парою пристроїв(контролер із пристроєм Інтернету Речей);
2. включення схеми шифрування на основі протоколу Deffie – Hellman;
3. проектування програмно – технічного засобу та його використання в дослідженні процесу взаємодії з усіма пристроями Інтернету Речей і використання інфрачервоної технології, для забезпечення захисту при першому налаштуванні пристрою;
4. програмування програмно – технічного засобу на мовах програмування C та Python.

Наукова новизна отриманих результатів:

1) вперше одержано генерування ключів аутентифікації під час налаштування нового пристрою Інтернету Речей, методом інфрачервоної технології та застосуванням QR-коду;

2) удосконалено систему Інтернету Речей, завдяки приєднанню розробленого програмно – технічного засобу, що використовується як пристрій налаштування та Brandmauer в системі пристроїв Інтернету Речей.

− набув подальшого розвитку метод інфрачервонної технології;

− набула подальшого розвитку інформаційна технологія безпечної генерації ключів аутентифікації.

На основі проведених досліджень розроблена архітектура і компоненти програмного забезпечення, що використовується як пристрій налаштування та Brandmauer в системі пристроїв Інтернету Речей.

Практична значимість отриманих результатів. В результаті виконаного дослідження та застосуванні на практиці методу та програмно – технічного засобу вдалось реалізувати процес безпечного та зручного налаштування пристроїв Інтернету Речей

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС -Операційна система

WPS - Wi-Fi protected setup

WPA - Wi-Fi Protected Access

KRACK (абревіатура англ. Key Reinstallation Attack) — повторне використання ключа

TOR - The Onion Router

ІЧ- інфрочервоний

IDE (англ. Integrated development environment або англ. IDE)- комплексне програмне рішення для розробки програмного забезпечення

SSH (англ. Secure SHell — «безпечна оболонка») — мережевий протокол рівня застосунків, що дозволяє проводити віддалене управління комп'ютером і тунелювання TCP-з'єднань (наприклад, для передачі файлів)

EEPROM (англ. Electrically Erasable Programmable Read-Only Memory) — постійний запам'ятовувач, що програмується та очищується за допомогою електрики, один з видів енергонезалежної пам'яті

ACK (анг. Acknowledge)- символ управління передачею

SSID (англ. Service Set Identifier) - унікальне найменування бездротової мережі, що відрізняє одну мережу Wi-Fi від іншої

PSK (англ. Phase-shift keying (PSK)) — один з видів фазової модуляції, при якій фаза частоти-носія змінюється стрибкоподібно залежно від інформаційного повідомлення

OLED (англ. organic light-emitting diode (OLED)) — світлодіод, у якому електролюмінесценція відбувається в тонкому пласті органічного напівпровідника, розташованому між двома електродами

ВСТУП

Аналіз існуючих джерел як у вітчизняних, так і в зарубіжних дослідженнях науковців та інженерів в галузі програмного забезпечення та кібербезпеки по темі даної роботи, показав, що задачі по усуненню проблем з безпекою та конфіденційністю у локальних мережах користувачів частково вирішено. Серед них виділяються:

1) антивірусні програми;
2) криптографічні засоби;
3) засоби аутентифікації користувачів;
4) засоби архівації даних;
5) засоби керування доступом;
6) протоколювання та аудит;
7) вбудовані засоби захисту інформації в мережевих системах ОС;
8) спеціалізовані програми захисту інформації[1].

Однак, разом з цим, не вирішеними, залишаються проблеми з появою нових методів атак на пристрої користувачів, з цілю викрадення конфіденційної інформації [2]. Особливо, це пов’язано з уразливістю комерційних налаштувань пристроїв Інтернету Речей.

Актуальність даної роботи на сьогодні полягає в тому що:

1) за останні роки зросла велика кількість пристроїв з підключенням до Інтернету під назвою Інтернету Речей [3]. Вони здатні в автономному режимі під’єднуватися до серверів, обробляти інформацію та приймати рішення;

2) більшість з них не реалізують жорстких принципів безпеки, що призводить до порушення безпеки та конфіденційності користувача.

Наступна робота пропонує альтернативу технології WPS на початку налаштування пристроїв Інтернету Речей, на якому пристрій буде згенеровано ключами аутентифікації Wi-Fi, для підключення до мережі.

Використання інфрачервоної технології, який реалізує протокол обміну ключами Diffie-Hellman для генерування ключа, робить процес набагато безпечнішим без шкоди для пристрою чи досвіду користувача [4].

Виконана дипломна робота має взаємозв'язок з такими напрямами як, комп'ютерна інженерія, програмування та кібербезпека.

Метою дипломної роботи є процес створення методу та програмно – технічного засобу генерування ключів аутентифікації.

Поставлена мета досягається розв'язання таких основних завдань:

– необхідно розробити метод, який полягає у вирішенні проблем з безпекою та складністю під час генерування ключів аутентифікації для налаштування нових пристроїв Інтернету Речей користувачем, що запобігатиме втручанню у процес третіх осіб(атаки Brute Force, Men in the middle, тощо);

– метод повинен бути розроблений з урахуванням аналізу вже існуючих методів та пов'язаних з ними проблем;

– необхідно спроектувати програмно – технічний засіб для генерування ключів аутентифікації, що використовує інфрачервоні технології та дасть змогу безпечного шифрування ключів аутентифікації.

Об'єктом дослідження є процес розробки методу та програмно – технічного засобу генерування ключів аутентифікації.

Предметом дослідження є властивості діяльності методу та програмно – технічного засобу генерування ключів аутентифікації, які можуть бути використанні у подальшому застосуванні в розробці пристроїв Інтернету Речей, таких як маршрутизатори та мережеві шлюзи і забезпечують вирішення задачі пов'язаною з безпекою та конфіденційністю під час налаштування нового пристрою Інтернету Речей.

Для досягнення мети використані наступні методи:

1) розробка протоколу зв'язку для безпечного та швидкого з'єднання між парою пристроїв(контролер із пристроєм Інтернету Речей);

2) включення схеми шифрування на основі протоколу Deffie – Hellman;

3) проектування програмно – технічного засобу та його використання в дослідженні процесу взаємодії з усіма пристроями Інтернету Речей і використання інфрачервоної технології, для забезпечення захисту при першому налаштуванні пристрою;

4) програмування програмно – технічного засобу на мовах програмування C та Python.

Наукова новизна отриманих результатів:

1) перше одержано генерування ключів аутентифікації під час налаштування нового пристрою Інтернету Речей, методом інфрачервоної технології;

2) удосконалено систему Інтернету Речей, завдяки приєднанню розробленого програмно – технічного засобу, що використовується як пристрій налаштування та Brandmauer в системі пристроїв Інтернету Речей.

Практична цінність отриманих результатів – в результаті виконаного дослідження та застосуванні на практиці методу та програмно – технічного засобу вдалось реалізувати процес безпечного та зручного налаштування пристроїв Інтернету Речей.

Отриманні результати можуть бути використанні для майбутньої розробки повноцінного пристрою маршрутизації з використанням інфрачервоної технології, що зробить налаштування та керування системою пристроїв Інтернету Речей набагато зручнішою, для користувачів, навіть тих, що не мають технологічних знань в даному процесі.

Також, безпека та конфіденційність будуть на більш вищому рівні, ніж в існуючих, на сьогодні, пристроях.

За темою дипломної роботи опублікована одна стаття у фахофому науковому видані «Вісник Хмельницького національного університету. Технічні науки»(Додаток А).

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ ВИЯВЛЕННЯ УРАЗЛИВОСТІ ПРИСТРОЇВ ПРИ ГЕНЕРУВАННІ КЛЮЧІВ АУТЕНТИФІКАЦІЇ

1.1 Розподіл ключів аутентифікації в системі пристроїв IP

Більшість існуючих пристроїв включають приватний ключ за замовчуванням, який впроваджується виробником під час виробничого процесу[5]. Для надання захисту ключам та пристроям, мікропрограма проходить етап шифрування за допомогою безпечного завантажувача. Однак, це не гарантує безпеку для пристрою, так як, замість того, щоб скомпрометувати зашифровану мікропрограму, завантажувач сам може стати мішенню. Скільки б рівнів безпеки не надавалося, завжди залишається відкритий код, запрограмований сценарій для завантаження та мікропрограма з пов’язаним з нею ключем[6].

Таким чином, заводський ключ вразливий під час етапу налаштування. Також, слід зауважити, що більшість ключів, розроблені, як тимчасові. Виробники рекомендують змінювати їх під час налаштування. Разом з тим, багато користувачів ігнорують дану рекомендацію, вважаючи, що попередньо визначений пароль повністю надає захист пристрою та конфіденційність інформації.

В даний час, у внутрішній системі Інтернету Речей, більшість виробників використовують технології Wi-Fi, або Bluetooth для розповсюдження нового згенерованного симетричного приватного між контролером домашнього концентратора Інтернету Речей (комп'ютер, мобільний телефон, або певний пристрій IP) та різними локальними вузлами Інтернету Речей (музичні колонки, телевізори та ін.), які також можуть до маршрутизатора через Wi-Fi[7].

Крім того, багато пристроїв Інтернету Речей, не включають порт дротового з’єднання(Ethernet або USB), тобто їх конфігурація повинна виконуватися бездротовим способом.

Початкове налаштування передбачає:

1) Мережа Wi-Fi або Bluetooth-з’єднання є повністю безпечними, тобто жодна третя сторона не має доступу для прослуховування трафіку, під час налаштування вузлів Інтернету Речей або введення ключа аутентифікації.

2) Центральний вузол Інтернету Речей здатний генерувати випадкові числа як основу для створення закритого ключа, який буде встановлено в різні клієнтські вузли[8].

З часом було доведено, що технологія Wi-Fi, не є такою безпечною, як про неї стверджують[9]. За останні роки було виявлено велику кількість вразливостей, таких як використання WPS або атака перевстановлення ключа WPA, що порушують безпеку WPA2. Крім цього, завжди будуть існувати перебої в системі, людські помилки під час налаштування мереж та пов'язаних з ними ключів.

Якщо в момент введення ключа аутентифікації, до мережі Wi-Fi підключився зловмисник, він зможе перехопити та розшифрувати його, а також отримати повний доступ над усіма пристроями в системі Інтернету Речей [10]. Це призведе до повної втрати контролю та безпеки пристроїв та захисту конфіденційної інформації.

При розгляді цього аспекту, виникає можливе рішення: використання протоколу обміну ключами Diffie-Hellman [4]. Даний метод був розроблений, для того щоб, при обміні ключами між пристроями, ніхто не зміг заволодіти інформацією, яка відбувається під час цього процесу та забезпечує конфіденційність приватного ключа.

Про те, метод протоколу Diffie-Hellman не надає захист від комбінованої атаки перехоплення, переривання та перевстановлення, про що далі буде наголошуватись в даній роботі.

Таким чином, можна зробити висновок, що сучасне керування та впровадження приватних ключів аутентифікації у внутрішню систему пристроїв Інтернету Речей, не є безпечною, оскільки не завжди виконуються дотримання безпечного генерування, під час виробництва[11].

Приклад протоколу Diffie-Hellman бачемо на рисунку 1.1.

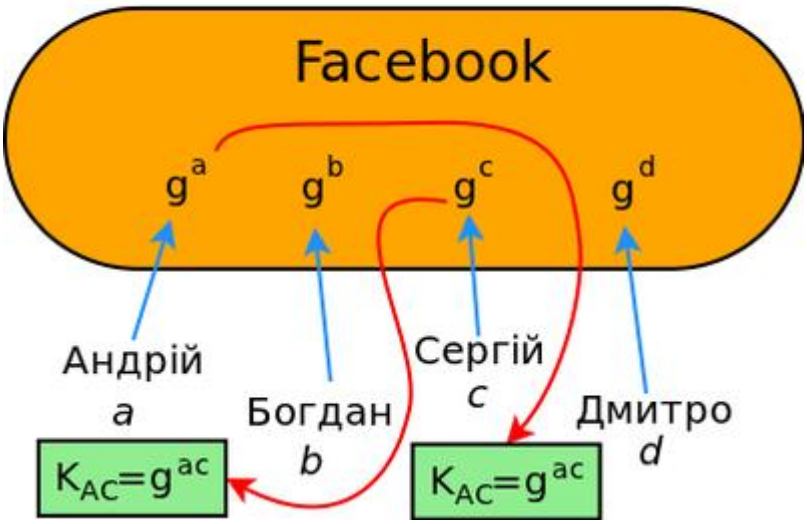

Рисунок 1.1 – Приклад протоколу Diffie-Hellman

1.2 Вразливості Wi-Fi

Система безпеки WPA2 не є надійною. Існує декілька атак, які можуть заволодіти інформацією через перехоплення трафіку зашифрованої мережі Wi-Fi. Дані атаки компрометують ін’єкцію приватного ключа Інтернету Речей або WPA [12].

1.2.1 Атаки перевстановлення ключа або атаки Krack

В 2017 році було оприлюднено вразливість, щодо експлойту, який має вплив на процес чотирьохстороннього рукостискання, що використовується у WPA2. Ця атака, вперше була описана в дослідницькій статті під назвою «Атака перевстановлення ключа» (Krack) [13].

Дана загроза дозволяє примусово скинути nonce ключа шифрування, що використовується користувачем для аутентифікації маршрутизатора. Це дозволяє зловмиснику розшифрувати багато пакетів шляхом скидання деяких ключових

параметрів в алгоритмі рандомізації, що використовується окрім ключа шифрування. В цій рандомізації використовуються два загальні параметри:

1) Nonce, який представляє кількість переданих пакетів;
2) Лічильник повторів, який відображає кількість отриманих пакетів.

Експлойт дозволяє розшифрувати пакети, які надходять від клієнта до маршрутизатора. Критичною проблемою є те, що кожен пристрій має бути виправлено окремо за допомогою оновлення пристрою, й навпаки. Це означає, що оновлення роутера не запобігає атаці підключення до нього, а навпаки, створює небезпеку для всіх даних клієнта.

Незважаючи на те, що сам ключ WPA2 не розкривається, ця атака дасть можливість зловмиснику перехопити згенерований ключ, надісланий користувачем на пристрій Інтернету Речей у процесі налаштування. Ризик піддатися атаці врази збільшується, якщо врахувати той факт, що багато користувачів не оновлюють свої пристрої, а це загроза конфіденційності й викриття інформації, що передається через з'єднання Wi-Fi [14].

1.2.2 Атака на захищене налаштування Wi-Fi (WPS)

WPS – це механізм, сертифікований Wi-Fi Alliance і реалізований у більшості сучасних маршрутизаторів, що дозволяє швидкий і простий спосіб підключення пристрою до маршрутизатора, як альтернативу класичному, вручну. Більшість користувачів використовують саме цей метод, через його простоту.

Однак, дослідження показали – WPS не є повністю безпечним для протидії атакам [15].

WPS можна реалізувати, як кнопку, щоб увімкнути сполучення між пристроєм та маршрутизатором або, як восьмизначний Pin-код. Головна проблема цієї атаки, втому, що не потрібно вгадувати усі вісім цифр разом.

При виявлені перших чотирьох цифр, маршрутизатор поверне контрольну точку, підтверджуючи їх правильність, повідомляючи зловмиснику, коли потрібно

перейти до останніх чотирьох цифр. Це полегшує атаку, так як перебір по чотири цифри, простіше за процес з вгадуванням всіх восьми за один раз. Крім того, восьма цифра Pin-коду є контрольною сумою, тому припущення скорочується до трьох цифр. Якщо знайти першу групу з чотирьох, другу з трьох, обчислити контрольну суму, яка представляє восьму цифру, то атака буде проведена у дуже короткий термін часу.

В мережі Інтернет доступно багато різних інструментів, що автоматизують дану атаку для проникнення у систему користувача. Одним з найвідоміших є Reaver. Існує багато наукових робіт [16], які дають опис для проведення можливих атак Brute Force.

Існує два способи захисту від цієї атаки:

– вимкнути опцію WPS у програмному забезпеченні налаштування маршрутизатора в системі Інтернету Речей;

використовувати маршрутизатор без підтримки WPS.

Окрім цього, більшість сучасних маршрутизаторів містять програми блокування зловмисника, після кількох невдалих спроб підбору коду. Тим не менш, загроза лишається та знаходяться нові методи для проведення атак.

Після того, як Pin-код було зламано, атакуючому надсилається ключ шифрування(WPA, WPA2), який використовує маршрутизатор, що означає приєднання пристрою до системи зловмисника. Таким чином, це дозволить проводити різні атаки, включаючи отримання конфіденційної інформації, наприклад приватного ключа Інтернету Речей, якщо власник мережі налаштовує свої пристрої за допомогою Wi-Fi[17].

1.3 Вразливість Bluetooth

Bluetooth є більш безпечнішим протоколом для з’єднання пристроїв, ніж Wi-Fi. Однак, також має місце вразливостей у безпеці [18]. Частина з них, стосується конкретних пристроїв, інша, пов’язана з протоколом з’єднання

1.3.1 Аналіз генерування ключами аутентифікації в Bluetooth

В 2018 році, була виявлена вразливість VU#304725 [19]. У ній зазначається, що частина останньої мікропрограми Bluetooth, відповідає за перевірку параметрів еліптичної кривої під час обміну ключами протоколу Diffie-Hellman.

Проблемою є те, що кіберзлочинець зможе провести атаку «людина по середині» та визначити криптографічний ключ, під час обміну ключами [20].

Усунення даної проблеми полягає в оновленні мікропрограми кожного пристрою та заміні скомпрометованих мікропрограм виправленими. Не виправлення, призведе до втручання третіх осіб, що призведе до втрати контролю над пристроями та конфіденційними даними користувача[21].

1.3.2 Аналіз вразливостей BlueBorne

BlueBorne – загальна назва для восьми вразливостей пристроїв, що працюють з різними імплементаціями Bluetooth в Windows, Linux, Android, IOS [22]. Набір вразливостей, виявлено в 2017 році, не можливо виявити і виправити стандартними методами, а для експлуатації шкідливих програм, зловмиснику не потрібні:

– взаємодія з користувачем;
– пара з цільовим пристроєм.

Кожна з цих вразливостей впливає на певну функцію протоколу в конкретних пристроях. Так, зловмисник при проведені атаки «людина по середині», повністю контролює пристрій, зчитуючи та записуючі конфіденційні дані клієнта[23].

У результаті атаки, кіберзлочинець отримує конфіденційну інформацію, включаючи приватний ключ Інтернету Речейё, особливо, при генерування ключів аутентифікації під час налаштування пристрою Інтернету Речей за допомогою Bluetooth [24]. Також, можливе віддалене захоплення та введення приватного коду кіберзлочинцем на пристрій клієнта.

На сьогодні, рекомендуються такі заходи безпеки по усуненню даної проблеми [25], які включають:

1. Оновлення, для пристроїв, для яких були випущені патчі.
2. Повне відключення Bluetooth до тих пір, поки не буде встановлений патч для відповідного пристрою.

Використання Bluetooth тільки при необхідності дозволить істотно звузити можливості зловмисників для злому.

1.4 Конфіденційність в системі пристроїв Інтернету Речей

Безпека та конфіденційність Інтернету Речей є концепцією, яка застосовується до конкретної програми Інтернету Речей, що становить проблему одного домену [26]. Однак, цього слід уникати, так як, внутрішні проблеми та цілі Інтернету Речей часто відрізняються від загального підходу. Потреби критичної широкомасштабної інфраструктури чи комерційної діяльності відрізняються від тих, які можуть мати побутові пристрої. Фінансові та людські ресурси, що використовуються для їх експлуатації та управління, також відрізняються [27]. Людській фактор найбільш впливає на критичну внутрішню систему пристроїв Інтернету Речей.

Окрім безпеки, конфіденційність є ще однією проблемою в системах пристроїв Інтернету Речей. Незважаючи на його важливість, даний фактор не завжди розглядається виробниками та розробниками, що призводить до конкретних проблем втрати конфіденційності в пристроях для їх власників [28]. За останні роки проведено велику кількість дослідів, у яких аналізуються різні сценарії та ризики конфіденційності, а також пропонуються рішення для них.

Для виконання своїх функцій, пристрої Інтернету Речей потребують підключення до мережі Інтернет [29]. Деякі з пристроїв дозволяють користувачам віддалено керувати та взаємодіяти з ними, в той час як інші, потребують надавати запити на зовнішні сервери для отримання певної інформації. Як правило, сервери

збирають конфіденційні та приватні дані з клієнтських пристроїв, а також, мають можливість бути використаними зловмисниками для надсилання шкідливої інформації та заволодіння ними [30].

Залежно від пристрою сервери, з яким він взаємодіє, отримують різні типи важливої приватної інформації [31]. Так, інтелектуальна лампочка, надсилає статистичні дані, що може призводити до розкриття розкладу процесів користувача у системі Інтернету Речей. Отримуючи щоденну інформацію про періоди включення та вимкнення лампочки, можна зробити точну оцінку перебування користувача у домашній системі пристроїв Інтернету Речей [32].

Іншою важливою проблемою є також те, що користувач може не знати, яку інформацію збирає пристрій. Прикладом для даного випадку є приховані мікрофони в деяких пристроях, що не використовуються для своїх функцій. Так, декілька років тому, в розумному роботі–міксері, був виявлений пристрій для прослуховування приватних розмов користувачів. Цей факт викликав серйозні суперечки, оскільки компанія-виробник не анонсувала даний компонент, як специфікація в наявному продукті [33].

Також, має місце те, що при підключенні до інтернет-сервісу, отримується ір-адреса пристрою [34]. В майбутньому, великі інформаційні сервери виростуть, зі зростанням попиту, щоб задовільнити потребу інформації для датчиків пристроїв Інтернету Речей. Вони діятимуть, як оновлена бібліотека, що надаватиме відповідь на різні запити оновлення інформації для пристроїв Інтернету Речей. Однак відмінність полягає в тому, що замість зосередження на конкретній сфері чи службі, їх дія буде спрямована на охоплення різних послуг та джерел інформації. Зрештою, це дасть змогу компаніям, через такі сервери, повністю отримувати інформацію про приватне життя та використовувати інструменти для вилучення інформації з великих даних[35].

Не зважаючи на те, що сервери будуть корисними для користувачів, вони збиратимуть мільйони гігабайтів інформацію про все приватне життя клієнтів. Також це призведе до з’єднання кожної ір-адреси зі схемою пристроїв Інтернету

Речей підключених до мережі [36]. Як наслідок, вся відокремлена інформація від кожного датчика та пристрою буде об’єднана спільною ір-адресою. Повна рентгенографія даної системи буде виставлена та комерціалізована у вигляді корисних даних для рекламодавців та третіх сторін, які можуть вважати це досить прибутковим для себе[37].

В даній роботі пропонується, для запобігання проблем з конфіденційністю в тісному підключенні до Інтернету Речей, кілька методів для анонімності трафіку, створеного з різнихпристроїв Інтернету Речей, щоб унеможливити групування за їхньою ір-адресою.

Метод полягає у використанні внутрішнього маршрутизатора у приватній системі пристроїв Інтернету Речей. Даний маршрутизатор матиме розширені функції брандмауера, щоб запобігти підключення пристроїв до небажаних серверів. Крім того, увесь трафік, що надходить з цього маршрутизатора, буде переправлений основним приватним маршрутизатором до мережі TOR, забезпечуючи його анонімність та конфіденційність .

Мережа Tor – це розподілена комунікаційна інтернет-мережа, в якій трафік передається через випадкову кількість вузлів, доки не буде перенаправлено на сервер, з яким мало бути початкове з’єднання [38]. Недоліком є те, що з’єднання неефективне з точки зору продуктивності (швидкість та затримка), проте це робить вузол запитувача неможливим до розкриття, забезпечуючи його анонімність для сервера. Кожний клієнт, що використовує мережу TOR, сам стає проміжним вузлом, що перенаправляє трафік інших користувачів до випадкових вузлів [39].

Оскільки, переважна більшість пристроїв Інтернету Речей, не потребують швидкого з’єднання або підключення з низькою затримкою, ця анонімна мережа може бути рішенням деяких проблем з конфіденційністю. Якщо сервер, котрий надає послугу, не може визначити місце, звідки надходить запит, дана інформація втрачає цінність як комерційний елемент [40].

Дослідження цієї проблеми, також показує, що деякі пристрої виконуючі запити, можуть розкривати важливу інформацію [41]. Так, наприклад, пристрій,

який робить запит про погоду, запитує сервер для свого регіону, що дає інформацію про місцезнаходження користувача. Одним зі способів зробити неочевидним це для сервера, надіслати запити на інформацію про різні регіони. Таким чином сервер не має можливості визначити знаходження користувача [42].

1.4.1 Контроль конфіденційності в мережі TOR

TOR –це браузер, що створено для забезпечення анонімності в Інтернеті. Завдяки добровільно встановленим серверам, якими може бути користувачі, маршрутизація інтернет-трафіку в мережі повністю приховує розташування клієнта. Так, Tor, ускладнює для третьої сторони скомпрометувати не лише цілісність конфіденційної інформації, а також справжню особу відправника та одержувача інформації, що надсилається.

Дана мережа є посередником, через яку проходить уся інформація. Спосіб обробки даних у TOR не дає виявити, що саме через цю мережу надіслано пакет даних, таким чином надаючи захист конфіденційності [43].

Встановлення сеансу даного браузера, для відправки інформації, відбувається, через встановлення програмного забезпечення на комунікаційному пристрої клієнта. Це програмне забезпечення включає засіб збору каталогу TOR, який необхідний для отримання списку доступних вузлів в усьому світі.

При отримані інформації про доступні вузли, користувач налаштовує деякі параметри підключення, а також кількість проміжних вузлів, що оброблятимуть трафік. При більшій кількості вузлів, з’єднання стає безпечніше, але затримка передачі даних зростає [44].

Пакети в мережі TOR називаються осередками і мають в довжину 498 байт. Вони надсилаються у зашифрованому форматі з фіксованою довжиною. Факт використання унікальної довжини, для будь-якого заданого типу протоколу зв’язку, унеможливлює отримання типу зв’язку, що має місце.

Процес використання браузера TOR полягає:

1. Користувач підключається до мережі TOR до мережі Інтернет, за допомогою програмного забезпечення даного браузера та встановлює спеціальне рукостискання з мостом TOR. Це перший вузол мережі.

2. Клієнт робить запит до каталогу керування TOR для доступних вузлів. Зробивши вибір вузла, користувач створює унікальну комірку схеми, сформовану іншими вузлами по всьому світі, яка буде маршрутизувати зв'язок. Кожен вузол Tor знає лише два своїх сусідніх вузла в ланцюзі [45] [46] [47].

3. Пакети фіксованої довжини надсилаються на будь-який внутрішній вузол TOR у зашифрованому вигляді.

4. Вихідний вузол перетворює пакети TOR в стандартні, для того, щоб мати можливість спілкуватися з сервером за допомогою інтернет-протоколів. Отримавши відповідь, він направляє її назад до першого вузла в ланцюзі, використовуючи ту саму механіку[48] [49]. Приклад цього зображено на рисунку1.2.

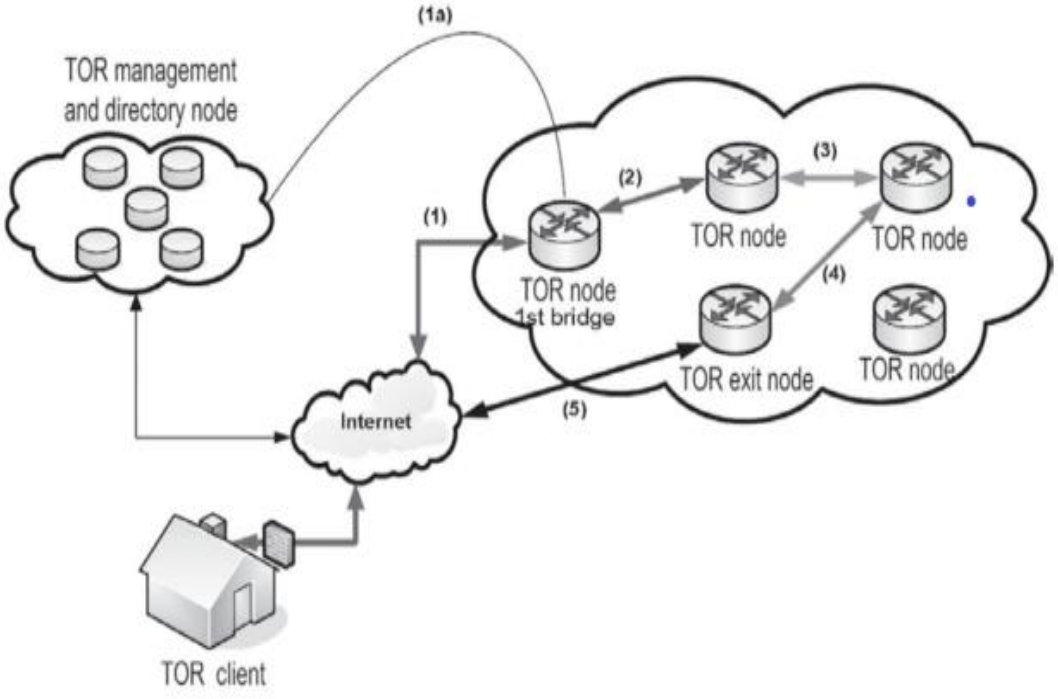

Рисунок 1.2 – Підключення користувачів до мережі TOR

1.4.2 Аналіз виявлення вразливостей пристроїв Інтернету Речей за допомогою пошукової системи Shodan

Shodan – це пошукова система розроблена програмістом Джоном Мезерлі та запущена у 2009 році [50] [51] [52]. Дана система є інструментом, що найбільше використовується під час пошуку пристроїв Інтернету Речей(маршрутизатори, вебкамери, сервери тощо), підключених до Інтернету за допомогою фільтрів.

Подібно до того, як працюють відомі пошукові системи, Shodan сканує Інтернет у пошуках онлайн-серверів та пристроїв та їхні запущені порти.[53] [54] [55] Для кожного результату створюється окремий запис в базу даних, що складається з ір-адреси, порту та банера служби.

Використання веб-сайта Shodan, дає клієнту здійснити пошук, під'єднуючись до бази даних цього ресурсу, яка постійно оновлюється та знаходиться у публічному доступі. Отримання доступу відбувається за допомогою веб-сайту Shodan або отримання Shodan API [56] [57].

Дана система є потужним інструментом для виявлення служб потенційних вразливостей. Однак, розширений пошук за допомогою Shodan, може бути складним і в більшості використовується спеціалістами в галузі кібербезпеки.

Окрім цього, на базі Shodan, був розроблений проект під назвою ShoVAT[58] [59] [60], який був спеціально розроблений для підвищення ефективності дослідження вразливостей та створення звіту про них, що містить список потенційно вразливих пристроїв [61] [62] [63]. Також, звіт включає запис стану пристроїв під час кожного оновлення та версії програмного забезпечення [64] [65].

ShoVAT здатний, крім виявлення вразливого пристрою, також отримати інформацію, коли відбулося ураження та скільки часу пристрій перебував у цьому стані до виправлення [65] [66] [67] [68].

Проект ShoVAT містить архітектуру інструменту оцінки вразливостей, яка спирається на веб-сервери Shodan, для створення модульної архітектури з

додатковими функціями, що допомагають спеціалістам у галузі безпеки, як показано на рисунку 1.3, взятому з документації ShoVAT:

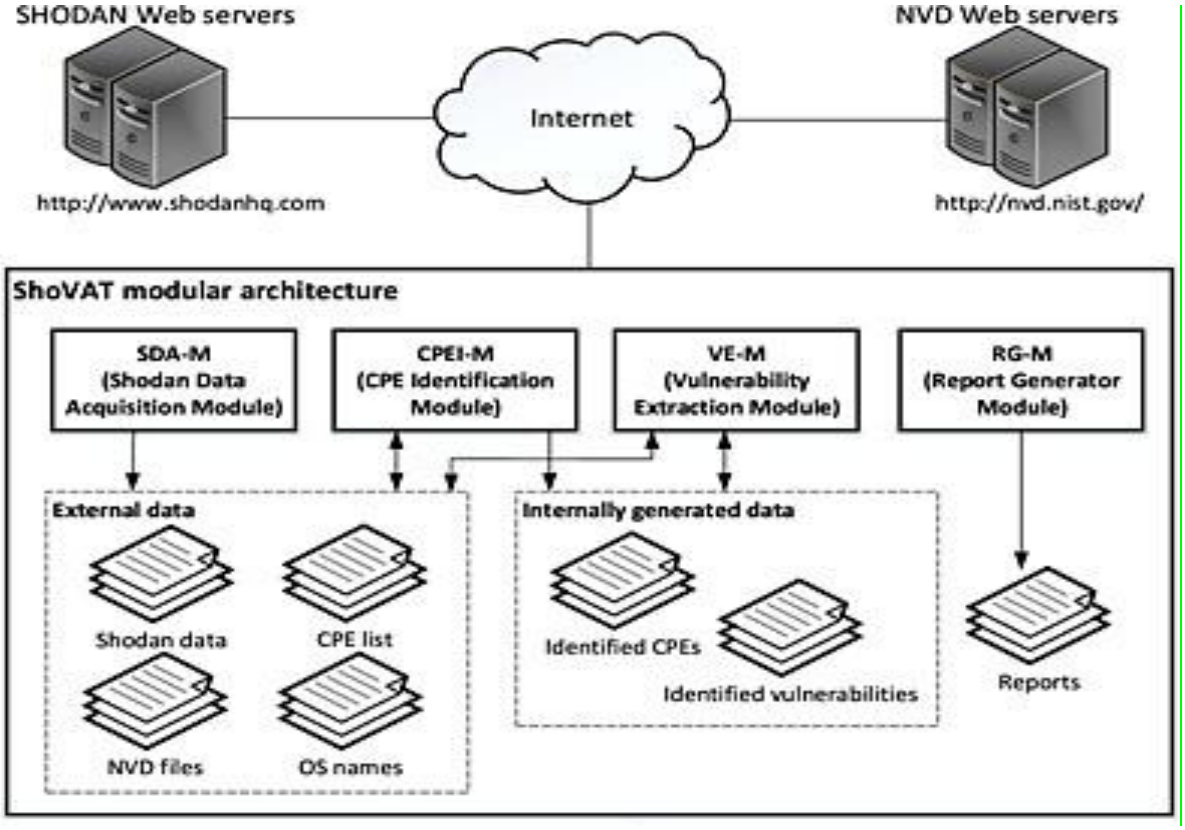

Рисунок 1.3 – Модульна архітектура Shodan на основі ShoVAT

1.5Постановка задачі

Дослідження теми генерування ключів аутентифікації показало, що проблема з шифруванням ключів, особливо на початковому налаштуванні, є досить актуальною [69] [70] [71]. Використання методу генерування ключів аутентифікації з застосуванням інфрачервоної технології, може бути одним із найбільш ефективним способом для безпеки пристроїв і захисту конфіденційності інформації користувача.

За для вирішення цієї проблеми, потрібно виконати наступні задачі:

1. Дослідити існуючі методи генерування ключів аутентифікації та визначити їх недоліки та шляхи вирішення;

2. Розробити модель процесу генерування ключів аутентифікації;

3. Розробити метод генерування ключів аутентифікації;

4. Розробити програмно-технічний засіб генерування ключів аутентифікації з застосуванням нового методу;

5. Підвести підсумки про необхідність розробки системи у майбутньому;

6. Провести експериментальне дослідження функціонування розробленого програмно – технічного засобу генерування ключів аутентифікації та оцінити його ефективність.

1.6 Висновки

Аналіз дослідження методів для захисту конфіденційності показав, що вони дієві та використовуються для забезпечення безпеки пристроїв та конфіденційності користувачів. З кожним роком удосконалюються інструменти та способи по боротьбі з ураженнями пристроїв і різними видами атак.

Разом з тим є недоліки, так як завжди з’являються нові види атак та вдосконалюються методи кіберзлочинців.

Переглянуті методи в Розділі 1, показали ефективність у виявлені загроз, особливо при генеруванні ключа аутентифікації під час першого налаштування та були вдосконалені при проектуванні програмно-технічного засобу, про що буде описано нижче.

2 МОДЕЛЬ ПРОЦЕСУ ГЕНЕРУВАННЯ КЛЮЧІВ АУТЕНТИФІКАЦІЇ ТА ПРОЕКТУВАННЯ АПАРАТНОЇ ЧАСТИНИ

2.1 Конфігурація сценарію

Основою даного проекту є програмна та апаратна частини, що пропонують новий метод у викладеній проблемі. Тому, в цьому розділі описуються запропоновані методи в реалізації програмно-технічного засобу генерування ключів [71] [72].

Програмна частина включає розробку надійної системи передач на основі інфрачервоного зв'язку. ІЧ-бібліотека використовується в цій роботі, як вихідна точка та буде налаштована, щоб підвищити її ефективність та дозволити двох направлену передачу великих обсягів даних. Існує декілька функціональних можливостей, включених у код:

1) керування пам'яттю;

2) використання EEPROM, для зберігання облікових даних, у разі відключення електроенергії [73] [74] ;

3) керування світлодіодним у корпусі контролера, або автоматичне з'єднання з точкою доступу, що дасть можливість користувачеві підключитися згодом до нього всередині внутрішньої мережі.[75] [76] [77]

Апаратна частина буде об'єднувати кілька пристроїв. Більшість з них взято вже з існуючих, що облегшило роботу по розробці програмно-технічного засобу.

2.2 Модель процесу створення програмної складової при генерування ключів аутентифікації

Основною частиною програмного забезпечення є розробка протоколу зв'язку для безпечного та швидкого з'єднання між парою пристроїв, таких як:

– контролер;

– пристрій Інтернету Речей.

Аналізуючи існуючі методи, що використовують виробники при генеруванні ключів аутентифікації для налаштуванні пристроїв маршрутизації, виявлено їх слабкі сторони [78] [79][80].

Проведено застосування такого методу як Brute Force до сучасних моделей роутерів. В експерименті було взято модель TP-Link.

Комерційний Pin-код, здебільшого стандартний для всіх пристроїв і включає вісім цифр, що показано на рисунку 2.1.

Рисунок 2.1 – Стандартний пароль маршрутизатора TP-Lin

Це дає можливість зловмиснику підібрати код, що було і доведено. Експериментальний Brute force проведено за допомогою застосування мови програмування Python, зображеного на рисунку 2.2.

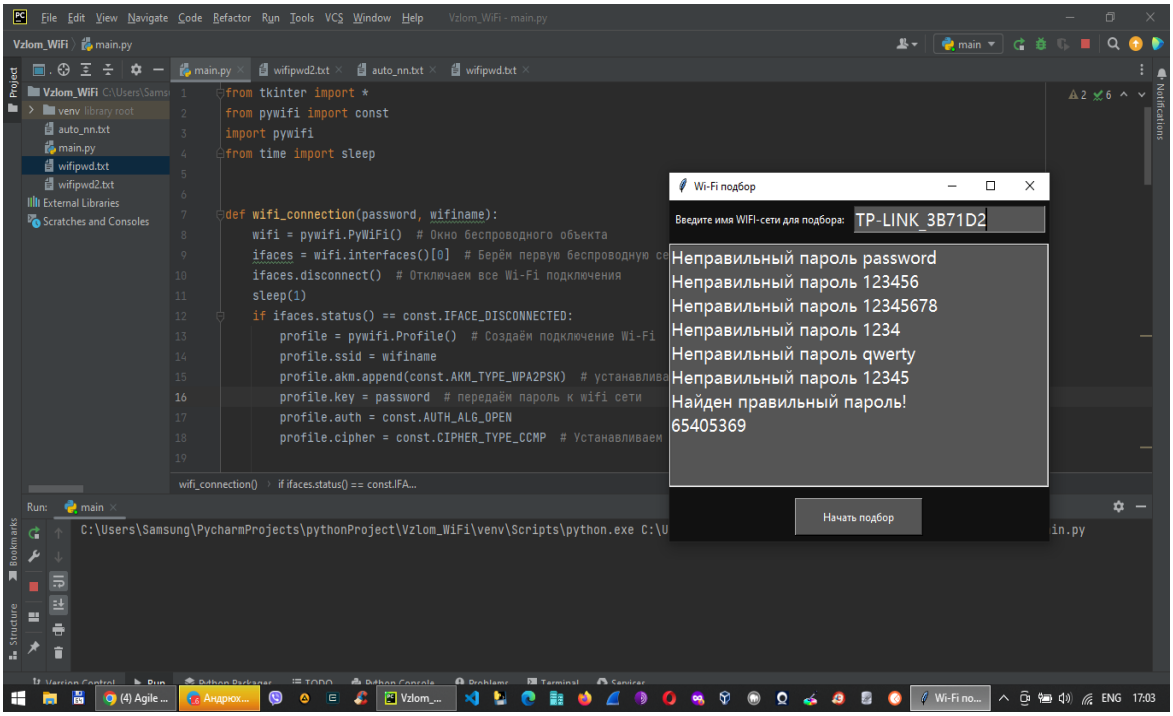

Рисунок 2.2 – Застосування атаки Brute Force на маршрутизатор TP-Link

Крім цього, на Рисунку 2.1, можна побачити стандартний пароль для налаштування внутрішнього пристрою, що складається з двох однакових слів: admin admin. Зміна налаштування пароля виконується через веб-ресурс, зв'язаний з роутером, що дає змогу перехопити генерування ключів аутентифікації при початковому налаштуванні.

Також однією з проблем не захищеності пристроїв є відсутність технологічних знань користувачів і складність застосування налаштування для звичайних клієнтів.

В цьому проекті пропонується метод інфрачервонної технології, що не застосовується на сьогодні у виробництві комерційних пристроїв.

Застосування даного методу показало, що майже не можливо отримати дані, особливо протокол генерування ключа аутентифікації, через змогу налаштовувати пристрої Інтернет Речей без виходу в мережу Інтернет .

При цьому, застосування методу буде повністю пов'язана з апаратною частиною дипломної роботи.

2.3 Метод інфрачервонної технології

Генерування ключів аутентифікації за допомогою інфрачервоного зв’язку є основною новизною та функціональністю даної роботи. ІЧ-зв’язок не розроблявся для надійної двонаправленої передачі інформації. Тому було вирішено самостійно розробляти нові функції, які забезпечували ці вимоги.

Це передбачає асинхронні виклики, а також постійну взаємодію з апаратним забезпеченням(ІЧ-приймачами та відправниками), що потребує тестування та ітерацію кожного нового дизайну для остаточної версії, котру можна б було вважати повністю функціональною, надійною та безпечною.

Оскільки, було запропоновано новий метод, вдосконалення процесу ІЧ-технології, було проведено глибоке тестування, що включало в себе:

1. Односпрямоване тестування, від контролера до приймача і навпаки. Надсилання деяких тестових даних, щоб побачити поведінку.
2. Двонаправлене тестування. Контролер починає зв’язок, надсилаючи пакет, що представляє число, одержувач назад той самий номер плюс один і так по циклу.
3. Тестування протоколів IR Library. Аналіз специфікацій кожного протоколу та поведінки на практиці до вибору найкращого: RC6(симетричний блоковий криптографічний алгоритм).
4. Визначення кожного необхідного типу пакету.
5. Оформлення каркасів. Кожен кадр містить, не лише дані як корисне навантаження, але й іншу корисну інформацію, таку як, адресат або тип пакету.
6. Проектування фаз зв’язку:
   – рукостискання;
   – надсилання даних;
   – закрите з’єднання.
7. Перша робоча версія. Інформація надсилалася за протоколом Stop&Go, без механізму виправлення помилок. Передача без шифрування зайняла 8 секунд.

Кожен пакет надсилався періодично до отримання свого ACK та переходу до наступного.

Недоліки цієї версії:

– не економічність часу;

– помилка при використанні двох послідовних пакетів з ідентичним корисним навантаженням.

8. Оптимізація встановлення зв’язку. У випадку паралельності, коли два пристрої одночасно прослуховують контролер, метод початкового встановлення зв’язку викликав зіткнення пристроїв один з одним під час спроби відповісти на пакет «Привіт».

Рішення: послідовність пакетів була розширена, включаючи виклик випадкового числа, тому лише отримувач, який згенерував те саме випадкове число, міг продовжити зв’язок з контролером.

9. Оптимізація протоколу часу. Було змінено бібліотеку для скорочення часу, необхідного для надсилання пакету.

10. Другий робочий варіант. Протокол Stop&Go замінено двома пакетними послідовностями(для SSID та PSK) з кінцевим ACK у кожній. Окрім цього, контрольна сума для виявлення помилок була включена в останній пакет. Завдяки виправленням, було вирішено проблеми, що були в попередній версії, а також деякі помилки апаратного забезпечення. Також було скорочено час, який потрібний для надсилання всієї інформації. Для чіткої передачі знадобилося близько 5,5 секунд.

11. Проектування шифрування. Включено новий етап зв'язку, щоб забезпечити підтримку спільного використання відкритих ключів за допомогою методу обміну ключами Diffie-Hellman.

12. Подальша оптимізація. Обмін ключами значно збільшив загальний час передачі. Для його зменшення було внесено деякі коригування та незначні зміни.

13. Третій робочий варіант. Додано деякі умови збою, для запобігання безкінечним циклам у разі переривання зв’язку, а також проведено часткову

оптимізацію. Це зменшило затримку передачі: 2,2 секунди в чистому вигляді та 3,7 секунди в зашифрованому.

14. Четверта робоча актуальна версія.

Дана версія скорочує час передачі приблизно на 15% порівняно з попередньою. Зменшується кількість загальних переданих пакетів з 3-х до 2-х, оскільки це включає відкритий ключ контролера в пакет даних, який надсилається. Отримані часові затримки становлять 2,0 секунди в чистому вигляді та 3,7 в зашифрованому вигляді.

Кожна версія адаптувалася до обставин вимог поточної проблеми. Це показує, що протокол передачі еволюціонував від занадто загального, здатного передавати різну інформацію, але не ефективним способом, до поточного, за яким передається втричі більше інформації за втричі менший час. Код було оптимізовано для цього конкретного випадку.

Незважаючи на те, що використання інфрачервоного зв’язку не є широко поширеним, через втрату універсальності, даний метод задовольняє вимоги безпеки і вдосконалює пристрої, що вже існують, більш надійним захистом генерування ключа аутентифікації на початковому етапі налаштування.

2.3.1 Модель процесу розробки диференційованих ІЧ-пакетів

Під часу дослідження методу застосування ІЧ-технології для безпеки генерування ключів аутентифікації, постало питання зробити зв’язок можливим. Для цього потрібна розробка диференційних пакетів, що дозволять одержувачу заздалегідь знати тип вмісту, який міститься в кадрі.

Даний процес по визначенню пакетів розробляється на початку розробки протоколу.

Остаточна версія реалізовує два різних за розмірами кадри:

1. 16-бітні пакети.

Використовуються в усіх кадрах ACK. Це було зроблено, щоб зменшити час передачі для підтвердження, що дає підвищення загальної ефективності протоколу. Приклад на таблиці 2.1:

Таблиця 2.1– Структура кадру ACK(16-біт)

Структура кадру ACK(16-біт)															
Тип пакету			Контролер	Корисне навантаження(12-біт)											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1) Три старші біти використовуються для ідентифікації пакета.

2) 12 біт – це позначка, що використовується для розрізнення пакетів, які надходять від контролера(якщо встановлено значення1).

3) Дванадцять молодших біт зарезервовано для корисного навантаження ACK, що надсилаються після отримання останнього пакету даних, що ідентифікує кінець пакету інформаційних пакетів. Корисне навантаження заповнюється інформацією про їх кількість.

2. 36 –бітні пакети.

Даний вид пакету використовується для всіх інших етапів у зв’язку. Тип пакетів обмежено до восьми, оскільки використовується 3-бітове поле, як ідентифікатор у кадрі.

36-бітні пакети включають:

– «Привіт Новий»: 0b010. Відповідає за встановлення зв’язку з новим вузлом. Даний пакет постійно транслюється з контролера на всі пристрої прослуховування, включаючи в його корисне навантаження новий ідентифікатор

вузла, що приймає вузол-відповідач. На нього відповідає вузол Інтернету Речей того самого типу;

– «Привіт»: 0b001. Цей пакет надсилається після «Привіт Новий», у разі зв’язку з новим вузлом. При під’єднані до відомого вузла, це може бути перший пакет. Спочатку він надсилається контролером з корисним навантаженням, що дорівнює вибраній випадковій послідовності, отриманій вузлом, у випадку з «Привіт Новий», або заповнюється ідентифікатором існуючого вузла, при зв’язку з ним;

– «Привіт Шифрований»: 0b000. Має такі ж функції як пакет «Привіт». Однак, оскільки немає місце для додаткового біта, на що вказує прапор шифрування, для контролера був встановлений новий тип пакету, щоб вказати вузол, інформація з якого буде надіслана в зашифрованому вигляді, тому він повинен підготуватися до процедури обміну відкритим ключем;

– дані: 0b011. Тип пакету, корисним навантаженням якого є фактична інформація. Він надійде в пакет із змінною кількості пакетів даних;

– кінець даних: 0b101. Вказує на завершення серії пакетів даних. Корисне навантаження, також містить інформацію. Після включення контрольної суми 32 біти корисного навантаження в пакет, «Кінець даних» завжди є контрольною сумою, що відповідає корисному навантаженню усього пакету;

– шифрування кінця даних: 0b110. Також вказує на кінець пакету та на те, що інформація була надіслана в зашифрованому вигляді і її потрібно розшифрувати перш, ніж її можна буде прочитати.

Роль кожного з пакетів детальніше буду розглянуто в 3 Розділі, під час перегляду фактичного процесу зв'язку інфрачервоного протоколу.

Структура 36-бітного кадру залежить від типу. Цей метод дає оптимізацію корисного навантаження у кадрах даних, де інформація про ідентифікатор призначення було видалено, щоб додати додатковий октет корисного навантаження.

Структура кадру будь-якої варіації пакету «Привіт» така в Таблиці 2.2:

Таблиця 2.2 – Структура кадру «Привіт»1.

<table>
<tr><td colspan="18">Структура кадру «Привіт»(36 біт)</td></tr>
<tr><td colspan="3">Пакет
Тип 3 біти</td><td colspan="8">destID
8-біт</td><td>Контро
лер</td><td colspan="6">Корисне навантаження
24-біти</td></tr>
<tr><td>35</td><td>34</td><td>33</td><td>32</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>18</td><td>17</td></tr>
<tr><td colspan="18"></td></tr>
<tr><td colspan="18">Корисне навантаження 24-біти</td></tr>
<tr><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

У порівнянні з кадром ACK, додаткові 20 біт дають більше місця для розширення корисного навантаження на 12 біт (з 12 до 24) та включення октету для ідентифікації адресата пакета.

Однак, як тільки послідовність рукостискання завершено, є гарантоване спілкування з вузлом, оскільки всі інші мовчатимуть до наступної послідовності пакету «Привіт». Після цього, немає необхідності включати заголовок ідентифікатора призначення. Признаючи цей простір для корисного навантаження, підвищується ефективність пакетної інформації на 50% (з 24 до 36 біт).

Ця реалізація для двох різних структур для однакової довжини пакети можлива, оскільки ідентифікатор пакета, поле типу пакета, залишається в тому самому місці в обох випадках. У програмній логіці, після отримання кадру перше, що робить код, шукає тип пакета, для визначення його структури.

Це рішення було прийнято для оптимізації протоколу. Дозвіл для більшої корисної інформації в пакеті, означає менше пакетів для передачі, що згодом скорочує її глобальний час.

Приклад наведено в Таблиці 2.3:

Таблиця 2.3– Структура кадру даних (36 біт)

<table>
<tr><td colspan="3">Пакет Тип 3 біти</td><td>Контролер</td><td colspan="14">Корисне навантаження (22 біти)</td></tr>
<tr><td>35</td><td>34</td><td>33</td><td>32</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td></tr>
<tr><td colspan="18"></td></tr>
<tr><td colspan="18">Корисне навантаження (22 біти)</td></tr>
<tr><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

2.4 Застосування шифрування даних за допомогою методу генерування ключів аутентифікації

Головна мета даного проекту – надати користувачеві безпечну альтернативу для завантаження облікових даних Wi-Fi на пристрої Інтернету Речей під час генерування ключів аутентифікації, без шкоди для них та для простішого використання. Застосування методу інфрачервоного зв’язку для досягнення поставленої мети є безпечнішим процесом, ніж використання інших відомих технологій, такі як WPS. Однак, це не причина, для того щоб уникати застосування додаткових заходів безпеки.

До даної роботи включено схему шифрування на основі протоколу Diffie–Hellman. Завдяки цьому, підвищується рівень захисту до того, що не буде можливості скомпрометувати дані, шляхом перехоплення зв'язку.

Обчислення в протоколі Diffie-Hellman, має такий вигляд(формули 2.1, 2.2).

Є дві людини, які таємно обирають два випадкових числа та таємно обчислюють їх й обмінюються, через незахищений канал передачі даних:

$$aA = gsA \quad (2.1)$$

$$aB = gsB \quad (2.2)$$

sA та sB – два випадкових цілих числа в інтервалі [0, |G| -1]

Після цього остаточне обчислення (формули.2.3б 2.4).

$$aBA = aBsA = gsBsA \quad (2.3)$$

$$aAB = aAsB = gsAsB \quad (2.4)$$

Слід зазначити, що $a_{AB} = a_{BA}$ і тому це число може служити таємним ключем K для двох користувачів.

Обидві частини зв'язку реалізують у коді однакову бібліотеку для генерування ключів аутентифікації, які в кінцевому підсумку застосовуватимуться для шифрування та дешифрування. Ця бібліотека має назву Curve25519-donna.

Дана крива, широко використовується в криптографії, завдяки її 128 біт безпеки та швидкості обчислення, під час отримання нових ключів. Крім того, він знаходиться у вільному доступі, так як не захищений патентом.

У коді повний процес, за допомогою якого, спільний ключ остаточно генерується шляхом комбінування відкритих ключів від контролера та вузла, які водночас є похідними від різних закритих ключів, є таким:

1. Генерування закритого ключа.

32-байтний ключ генерується випадковим чином. Це робиться поза бібліотекою Curve25519. В даному проекті використовується та сама бібліотека, яка застосовується для випадкової генерації облікових даних точки доступу.

2. Генерування відкритого ключа.

32-байтний відкритий ключ обчислюється з закритого ключа та базової точки, яка зазвичай є константою (у коді uint8_t basepoint[32] = {9}). Після створення закритого ключа та базової точки за допомогою функції Curve25519-donna, повертає відкритий ключ, обчислений за допомогою еліптичної кривої.

3. Обмін відкритими ключами.

Після того, як обидва пристрої згенерували свою пару відкритих приватних ключів, вони надсилають відкритий ключ один одному. Це відбувається за допомогою розробленого протоколу інфрачервоного зв’язку. Оскільки використовується інфрачервоне випромінювання, шанс бути перехопленим, малоймовірний. Навіть, якщо його сканували, схема Diffie-Hellman не дозволяє кіберзлочинцю розшифрувати зв’язок, так як йому знадобиться приватний ключ, що ніколи не передається.

4. Генерування спільного ключа.

На цьому етапі, кожна частина має відкритий ключ іншої і завдяки цьому можуть генерувати спільний ключ, що остаточно застосовується для шифрування та дешифрування інформації. За допомогою бібліотечної функції Curve25519-donna, що викликається з аргументами приватного та відкритого ключа, генерується 32-байтний спільний ключ.

Обидва ключі мають бути однаковими, через математику, яка лежить в основі генерування ключів аутентифікації.

5. Шифрування даних.

В цьому випадку шифрування обмежене лише закритим ключем точки доступу розміром 32 байти. Існує декілька способів шифрування даних, а також різні доступні бібліотеки на основі векторів ініціалізації та одноразовості, щоб уникнути витоку ключа шифрування, через його надмірне використання. В даній роботі ці бібліотеки не використовуються, через особливість технології, в якій довжина даних для шифрування збігається з довжиною спільного ключа: обидва мають 32 байти. Тому для шифрування та дешифрування використовується проста функція XOR (eXclusive OR) на битовому рівні:

– дані зашифровані = Спільний ключ очищення даних XOR;
– очищення даних = Спільний ключ XOR, з шифруванням даних.

Логічна таблиця XOR зображена в таблиці 2.4:

Таблиця 2.4 – Логічна таблиця XOR

Логічна таблиця XOR		
Введення		Виведення
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

В даному прикладі, якщо спостерігається застосування операції XOR до будь-якого значення проти самого себе, результат завжди буде 0.

Наприклад, [xXOR A] XOR A завжди дорівнює x як: 0 XOR 0 == 1 та 1 XOR1 == 0.

Це робить операцію XOR простим та швидким способом шифрування та дешифрування інформації, у випадку, якщо інформація для шифрування відповідає ключу шифрування за довжиною.

Важливо зауважити, що кожна нова комунікація передбачає повторення цього процесу та буде згенеровано новий закритий ключ.

Так як, спільний ключ використовується лише один раз для шифрування інформації, немає потреби застосовувати випадкові числа одноразового використання, що змінюють ключ підчас кожного завантаження та вектори ініціалізації, складність шифрування значно зменшується.

Якщо довжина ключа була більшою, наприклад 64 байти, використання одного ключа для шифрування двох блоків по 32 байти знизило б безпеку схеми. Чим частіше ключ використовується повторно, тим слабшим він є.

2.5 Процес застосування апаратної частини

Емуляція точки доступу Інтернету Речей, здійснюється через пристрій Raspberry Pi 3b+. Уся розробка виконана з використанням мови програмування Python 3.7.

Так як, даний пристрій поставляється без будь-якої операційної системи, в цьому проекті вирішено використовувати Операційну Систему на основі Linux Debian, розробленого спеціально для Raspberry.

Raspberry Pi 3b+ схожий на інші ОС на базі Linux, вся конфігурація виконується через команди терміналу. Невелика відмінність є в деяких конфігураціях та в керуванні політикою безпеки, які потрібно вимикати локально. Наприклад, щоб запустити модуль камери Raspberry Pi або дозволити віддалене підключення SSH, потрібно змінити політику безпеки.

Після налаштування політики безпеки важливо встановити статичну IP-адресу у домашній мережі, щоб згодом отримати віддалений доступ з інших комп'ютерів та полегшити налаштування таблиці маршрутизації після запуску точки доступу.

Це робиться шляхом зміни файлу /etc/dhcpcd.conf та додаванням наступної інформації:

```
interface eth0
        static IP-address=192.168.1.11/24
        static routers=192.168.1.1
        static domain_name_servers=192.168.1.1
interface wlan0
        static IP-address=192.168.100.1/24
        nohook wpa_supplicant
```

В даному випадку eth0 відповідає дротовому інтерфейсу, що з'єднує пристрій з домашнім маршрутизатором, який фактично буде використовуватись під час віддаленого налаштування пристрою з комп'ютера.

Interface wlan0 пізніше буде виділено, як точка доступу. Тому, для нього призначається інший статичний IP-адрес, який закінчується на 192.168.100.1, оскільки він діятиме як шлюз для пристроїв Інтернету Речей, підключених до цієї бездротової мережі. Однак, конфігурацію wlan0, не потрібно додавати вручну, так як сценарій налаштування точки доступу робить це автоматично.

В даній роботі програмування, виконання та налагодження коду Raspberry Pi повністю виконується віддалено. Комп'ютер використовується під керуванням PyCharm Python IDE на ОС Windows 10. Ця IDE підтримує віддалені інтерпретатори Python та автоматичне віддалене завантаження файлів, що значно полегшує програмування пристрою.

Створення точки доступу легко виконується за допомогою терміналу. Однак, для цього потрібно виконати декілька кроків, так як для проекту важливо забезпечити простий спосіб автоматичного налаштування та перезапису попередньо запущеної точки доступу.

Налаштування виконується за допомогою Python 3. Використовуються, вже існуючи локальні бібліотеки Python:

1) os;
2) sys;
3) subpocess;
4) re.

Окрім даних локальних бібліотек, було встановлено деякі зовнішні:

– iptables_persistent;
– dnsmask;
– hostapd.

Метод даного сценарію названо wifi_setup_test.py. Він приймає два параметри як аргументи: SSID та PSK, які матиме налаштована точка доступу. Для запуску, викликаємо його за допомогою команди в терміналі:

sudo python3 wifi_setup_test.py <SSID><Key>

Якщо права адміністратора sudo не надано, сценарій спробує відкликати себе, за допомогою sudo, якщо це можливо.

Сценарій виконуватиме такі дії:

– встановлення всіх необхідних пакетів для коректної роботи скрипта;

– знищення попередньої конфігурації точки доступу, якщо вона існує;

– конфігурація нової точки доступу з наданими SSID паролем;

– конфігурація маршрутизації між інтерфейсами eth0 та wlan0. Це дозволяє підключати до інтернету пристрої, підключені до wlan0, за замовчуванням, через NAT Masquerading;

– конфігурація постійності. Конфігурацію розроблено для забезпечення постійності, у разі перезавантаження пристрою. Точка доступу повинна запускатися автоматично щоразу, коли пристрій увімкнено.

Для полегшення процесу передачі облікових даних AP від контролера Інтернету Речей до Raspberry Pi застосовано метод програмування другого сценарію, що автоматизує дію попереднього, коли доводиться вводити дані в ручному режимі.

Завдяки цьому, достатньо просканувати QR-код із SSID та PSK через камеру. Крім того, цей код логічно пов’язаний зі сценарієм створення AP.

Програма використовує модуль OpenCV 3.4.4 для розпізнавання та обробки пікселів в зображенні, щоб шукати QR-код. Окрім неї використовується ще одна зовнішня бібліотека Python: zbar. Даний модуль відповідатиме за аналіз з зображення, отриманого з OpenCV, для пошуку дійсного QR-коду.

Реалізація в Raspberry Pi змушує сканер періодично робити робити та аналізувати зображення, поки не знайде дійсний QR.

QR-код вважається дійсним, якщо його можно правильно прочитати, а також, якщо розшифрована інформація відповідає формату, який використовується для кодування SSID та PSK. При виконані цих умов, програма припиняє сканування та викликає сценарій wifi_setup_test.py з SSID та PSK, як аргументами, що дозволяє налаштувати точку доступу.

Сканер камери можна запустити вручну за допомогою Python2 camera_scan.py. Однак Raspberry Pi налаштовано на автоматичний запуск цього сценарію під час кожного завантаження, якщо немає існуючої точки доступу. У цьому випадку застосовується запуск вручну.

Процес виконання даного сценарію виглядає так (рисунок 2.3).

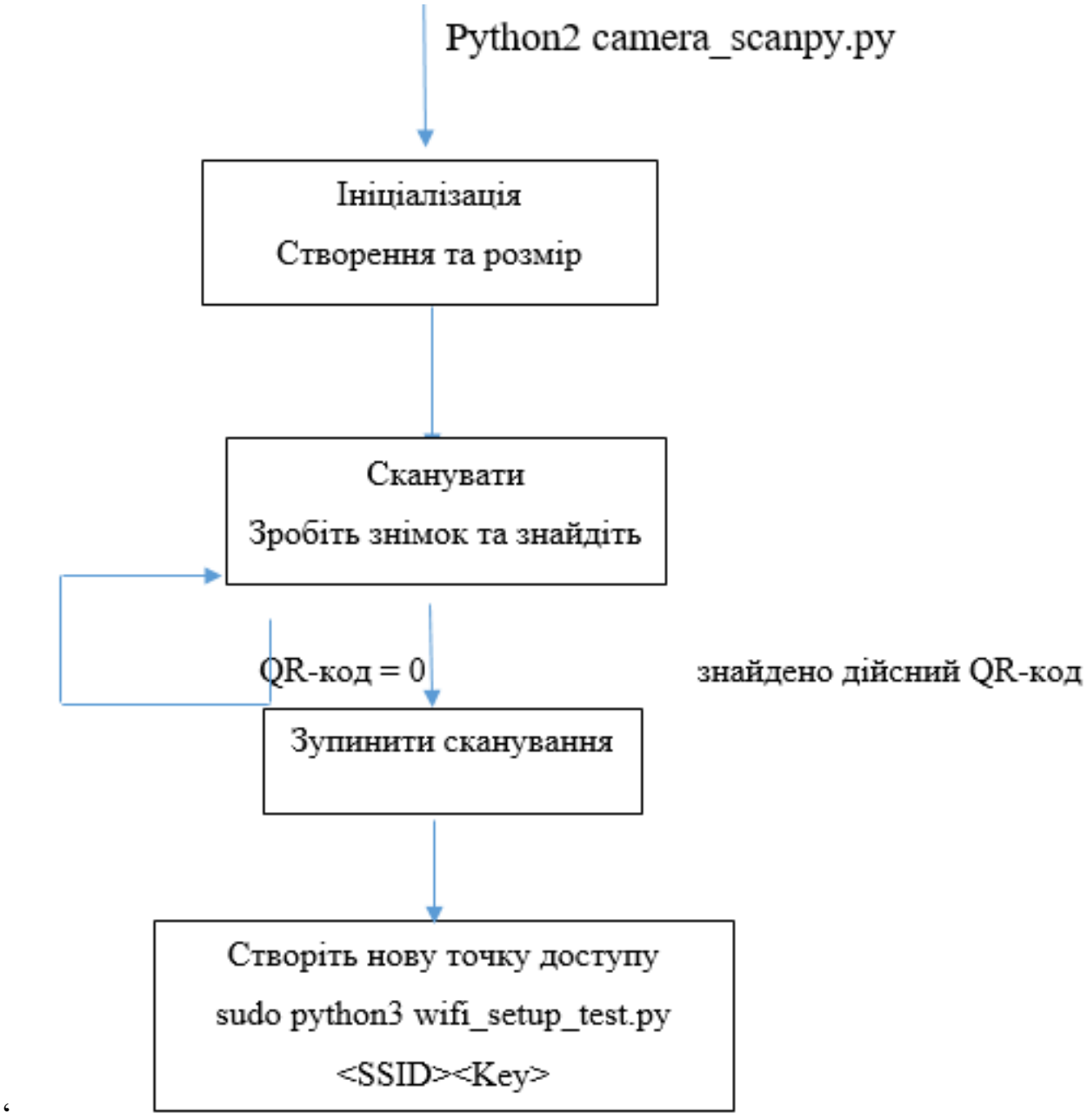

Рисунок 2.3 – Діаграма стану коду QR-сканеру

Під час виклику вручну або автоматично, після завантаження, сценарій ініціалізує камеру Raspb Pi, вказуючі деякі параметри конфігурації зйомки. Після встановлення даних параметрів, застосовується сканування.

Етап сканування – це нескінчений цикл, у якому він періодично оптимізує знімки та аналізує їх, поки не знайде QR-код. Коли запис не містить дійсний QR-код, сценарій проігнорує його та створить новий. При надходженні дійсного – сценарій припиняється та створюється точка доступу sudo python3 wifi_setup_test.py <SSID><Key> , з параметрами декодованого QR-коду, що був присутній під час останнього захоплення.

Як додаткову альтернативу, було реалізовано скрипт, який може автоматично зчитувати дані авторизації з послідовного порту на Raspberry Pi.

Цей скрипт був розроблений для простоти використання: спочатку він запускається з терміналу, а потім IoT контролер підключається до Rasp Pi через USB. Скрипт починає зчитувати дані безпосередньо з послідовного порту та шукає вихідні дані для Wi-Fi: ім'я мережі (SSID) та пароль (PSK). Як тільки дані авторизації зчитано, скрипт завершує роботу та викликає програму wifi_setup.py для створення нової точки доступу з термінальною командою:

```
python3 serial_reader.py
```

Цей скрипт працює з обома версіями Python, але був ретельно протестований лише на версії Python 3. Важливо також зазначити, що для правильної роботи IoT контролера, йому потрібно було призначити порт ttyUSB0. Якщо до Rasp Pi було підключено будь-який інший пристрій, його потрібно було відключити перед підключенням контролера.

Складність цього скрипту дуже низька, оскільки для комунікації з послідовним портом використовується заздалегідь встановлена бібліотека для Python, яка називається serial. Однак для цього застосування не потрібно

відправляти жодну інформацію на пристрій контролера. Повний код цього скрипту доступний для читання в Додатку V.

Як вказано на Рисунку 7, наразі існує три різні альтернативи для налаштування точки доступу. Якщо перша полягає вручну введенні, то дві інші спрощують процес для користувача. Ці скрипти з вилучення інформації також можуть бути налаштовані на автоматичний запуск під час завантаження Rasp Pi, оскільки в звичайній ситуації користувач може не мати знань для доступу до нього за допомогою віддаленої сесії.

Як ми вже згадували раніше, існує четверта альтернатива, яка не зображена на діаграмі, і полягає у передачі ключа до Rasp Pi за допомогою інфрачервоних променів. Оскільки вона наразі не працює, то була просто пропущена (рисунок 2.4).

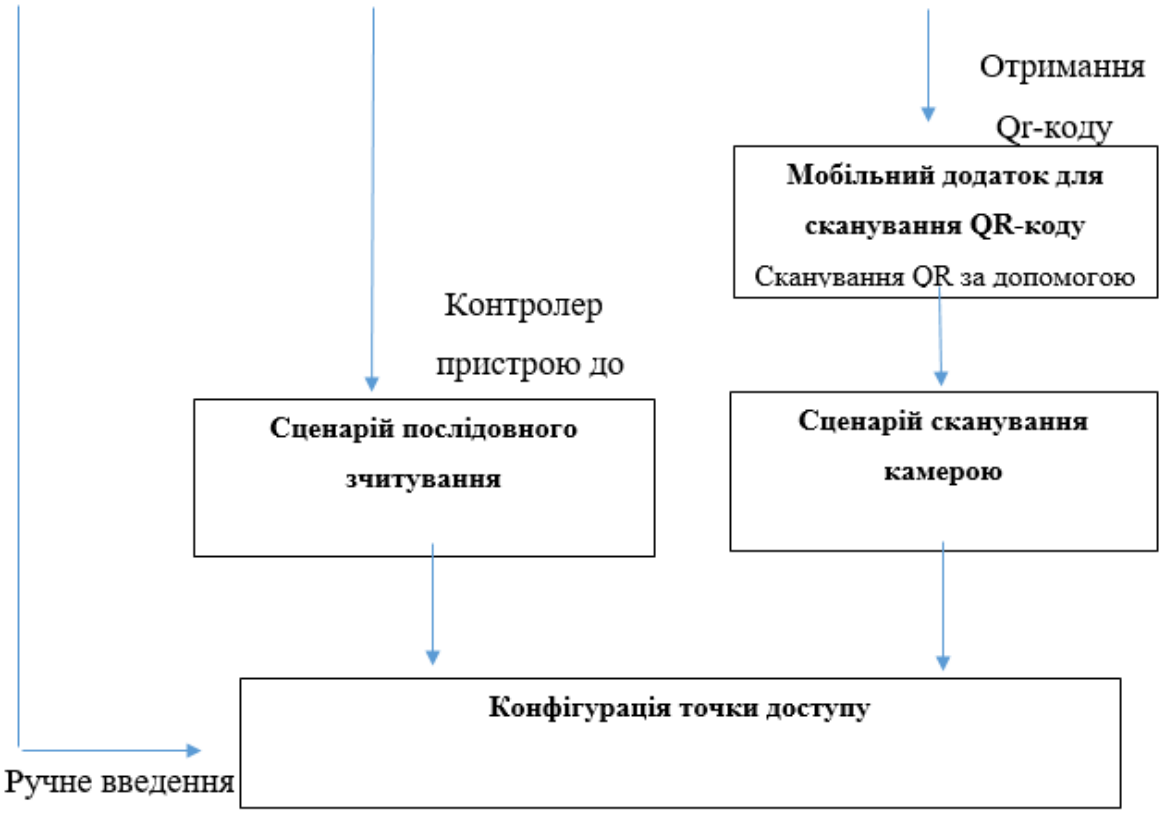

Рисунок 2.4 – Альтернативи налаштування точки доступу в Rasp PI

2.6 Висновки

Проведено аналіз існуючих методів генерування ключів аутентифікації та виявлено їх недоліки. Також розроблено новий метод, що базується на інфрачервоній технології та забезпечує більший захист конфіденційної інформації.

У роботі було використано Raspberry Pi та камеру Rasp Pi, що є доступними та дешевими пристроями, доступними для більшості користувачів. Використання мови програмування Python дозволило швидко та ефективно реалізувати розроблені методи.

Одним із найважливіших відкриттів роботи є новий метод генерації ключів аутентифікації на основі інфрачервоної технології. Цей метод дозволяє забезпечити більш високий рівень безпеки, оскільки він зменшує можливість перехоплення ключа аутентифікації з відстані.

Застосування програмно-технічних засобів, таких як Raspberry Pi та Python, показало їхню ефективність та доступність для широкого кола користувачів. Результати дослідження можуть бути корисні для розробки безпечних систем Інтернету Речей та забезпечення захисту від несанкціонованого доступу.

k3 ПРОЦЕС РОЗРОБКИ АРХІТЕКТУРИ І СЦЕНАРІЮ МЕТОДА ТА ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Архітектура та конфігурації сценарію

Дизайн внутрішньої мережі пристроїв Інтернету Речей в даній роботі відповідає двом основним вимогам: безпека та конфіденційність. Ці два фактори є ключивими, так як з кожним роком все більше приладів підключаються до Інтернету, а вони зможуть припинити доступ до пристроїв користувачів різного роду зловмисників (рисунок 3.1).

Рисунок 3.1 – Діаграма пропозиції топології пристроїв Інтернету Речей

Мережа матиме два маршрутизатори:

1. Основний маршрутизатор: той, що надається Інтернет-провайдером. Він відповідає за надання Інтернет-підключення. Усі не-IoT пристрої, такі як телефони, ПК або ноутбуки, будуть підключені безпосередньо до нього (дротовим

або бездротовим зв'язком). Це дозволить їм мати повний доступ до Інтернету зі стандартними обмеженнями, які налагає Інтернет-провайдер на домашні мережі. Представлене рішення сумісне з будь-якою поточною домашньою мережею, оскільки воно не передбачає жодних змін у цій стандартній мережі, хоча для внесення деяких змін до їх параметрів можуть знадобитися розширені настройки.

2. IoT-маршрутизатор: точка доступу Інтернету Речей з можливостями брандмауера. Цей маршрутизатор з'єднаний дротово з основним маршрутизатором. Він дозволить підключати до нього бездротово тільки пристрої Інтернету Речей. Він також реалізує додаткові функції для забезпечення безпеки та приватності.

Пропозиція додає до існуючої мережі цілком нову мережу, створену IoT-маршрутизатором. У практичному сценарії цей IoT-маршрутизатор - це Raspberry Pi Model 3b+, основні технічні характеристики якого будуть описані пізніше. Він реалізує операційну систему Raspbian, яка є операційною системою на основі Debian, і працює як точка доступу для всіх пристроїв Інтернету Речей, які було для цього налаштовано.

3.2 Апаратне забезпечення

Апаратне забезпечення, що використовується у цій пропозиції, складається з кількох пристроїв та деяких схем, що використовуються як підсилювачі для інфрачервоних компонентів. Багато з пристроїв були придбані у форматі дошки розробки, оскільки це значно зменшує складність необхідності зварювати та проектувати PCB для об'єднання всіх менших деталей.

У комерційному сценарії було б рекомендовано купувати компоненти окремо за більш доступною ціною та проектувати власні плати або PCB, щоб максимально зменшити витрати.

3.2.1. ESP8266 NodeMCU як пристрій Інтернету речей

Для імітації поведінки пристроїв Інтернету речей використовується мікроконтролер ESP8266, вбудований у дошки розробки NodeMCU[11b]. Ці мікросхеми є низькобюджетними, а їх основні особливості для NodeMCU devkit наступні:

– Мікропроцесор ESP8266, RISC 32-бітний процесор з частотою 80 МГц
– 32 кіБ інструкційної пам'яті RAM в залежності від моделі
– 96 кіБ користувацької та системної оперативної пам'яті RAM
– 4 Мб флеш-пам'яті
– IEEE 802.11 b/g/n Wi-Fi
– 10 GPIO-пінів (припаяно)
– Регульований USB-напруга для живлення

Плата зображена на рисунку 3.2

Рисунок 3.2 – Плата ESP8266 NodeMCU

3.2.1.1. Проектування схеми ІЧ світлодіодного передавача

У всіх пристроях симуляції Інтернету Речей використовується схема для підсилення потужності вихідного ІЧ світлодіодного передавача. Незважаючи на те, що він вже працює при підключенні безпосередньо до PIN-кодів, після тестування

було виявлено, що дальність передачі зменшується, а відношення помилок бітів значно зростає. Тому було вирішено вкласти трохи більше зусиль в проектування підсилювальної схеми для подачі більшого струму на LED.

Розроблена схема передачі має наступну конфігурацію(рисунок 3.3).

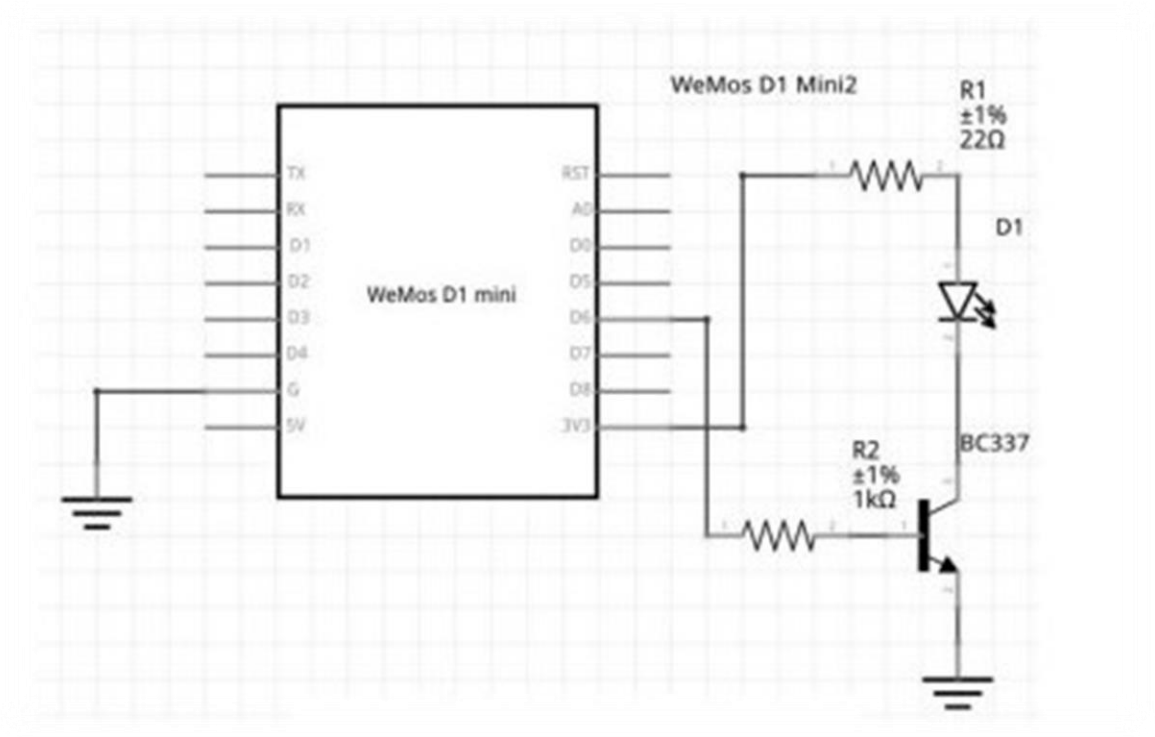

Рисунок 3.3 – ІЧ – світлодіодний передавач, схема1

Поточне значення струму в світлодіоді (D1 на Рис. 10) становило близько 91 мА в постійному струмі. Напруга між його клемами становить 1,3 В. Ми маємо запас безпеки майже в 10 мА, оскільки виробник світлодіода рекомендує не перевищувати 100 мА струму та мати напругу між 1,2 та 1,4 В.

З цим колом нове тестування показало, що зв'язок є швидшим та набагато надійнішим. Однак дальність все ще була обмеженою, і деякі пакети були пошкоджені, що збільшило затримку передачі.

Було зроблено переробку цього кола з метою виправлення цього недоліку, і було запропоновано нове коло з 2 світлодіодами, збільшеної дальності. Це був

дизайн, який ми остаточно вибрали для наших остаточних тестів, оскільки, проаналізувавши витрати, ми висновили, що додаткові інвестиції варто зробити.

Остаточне коло передавання для плат NodeMCU (IoT-пристроїв) має наступний дизайн (рисунок 3.4).

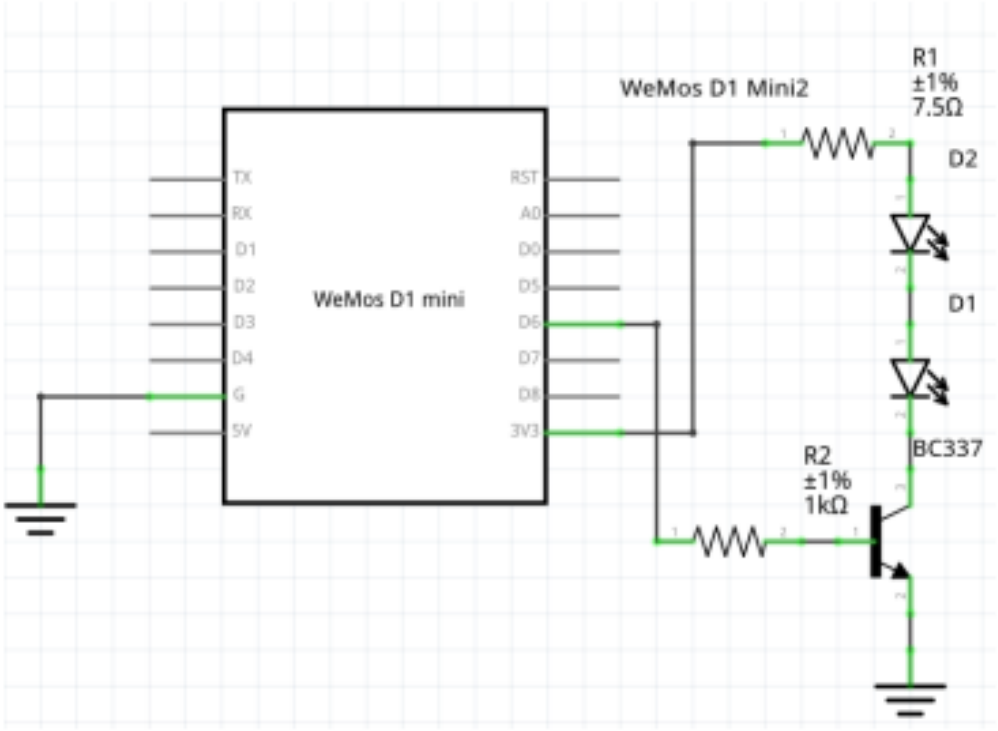

Рисунок 3.4 – ІЧ – світлодіодний передавач, схема2

Як можна побачити, схема подібна до попередньої, хоча тепер у ній є 2 світлодіода в серії, а значення резисторів змінено, щоб забезпечити стабільний струм 93 мА в DC і 1,3 В на клемах діодів.

Оскільки не вдалося знайти комерційне значення 7,5 Ω для резистора, ми замінили його двома резисторами в серії на 5,6 Ω і 1,8 Ω, що в результаті дає всього 7,4 Ω. Це означає, що на практиці струм у діодах трохи вищий. Однак це не потребує переробки, оскільки він все ще залишається нижче максимального значення струму 100 мА.

Також, точності резисторів діаграми 1% не було знайдено в магазині. Тому в цих схемах використовуються резистори з точністю 5%.

Було зроблено деякі розрахунки, щоб теоретично визначити відстані та ймовірності помилок у передачі. Результати підсумовано в наступній Таблиці 3.1:

Таблиця 3.1 – Імовірність помилок одного та двух світлодіодів

Імовірність помилки на 4 метри		
	1-бітний блок	32-бітний блок
1 світлодіод	$6.4 * 10^{-4}$	$2 * 10^{-2}$
2 світлодіоди	$8 * 10^{-6}$	$2.6 * 10^{-4}$

Як можна побачити, ймовірність помилки значно зменшилась (біля 100 разів менша ймовірність помилки), якщо використовувати два інфрачервоних світлодіода передачі.

Оскільки вартість додаткового інфрачервоного світлодіода є доступною, даний дизайн нарешті використовує два світлодіоди в серії.

Ланцюг інфрачервоного приймача простіший, оскільки він працює коректно, якщо підключити його безпосередньо до GPIO контактів і не потребує будь-якого другого налаштування.

Компонентом ІЧ, який використовується як приймач, є TL1838. Було вибрано цю модель через її низьку ціну та правильну продуктивність.

Однак є кращі, але дорожчі моделі, такі як TSOP2238, з більш високою продуктивністю у виявленні помилкових позитивів та фільтрації інфрачервоних навколишніх перешкод.

Схема ІЧ-приймача зображена на рисунку 3.5.

Компонент інфрачервоного приймача має 3 контакти: Ground, Data та Vcc. Дизайн просто з'єднує кожен з них з аналоговим PIN-кодом на дошці NodeMCU.

У якості контролера використовується інша плата, яка втілює мікросхему ESP8266.

Контролер є ключовим пристроєм у дизайні, оскільки він відповідає за завантаження ключа на кожен пристрій IoT одним натисканням кнопки, а також дає можливість користувачеві отримати згенеровану пару SSID та PSK для налаштування точки доступу.

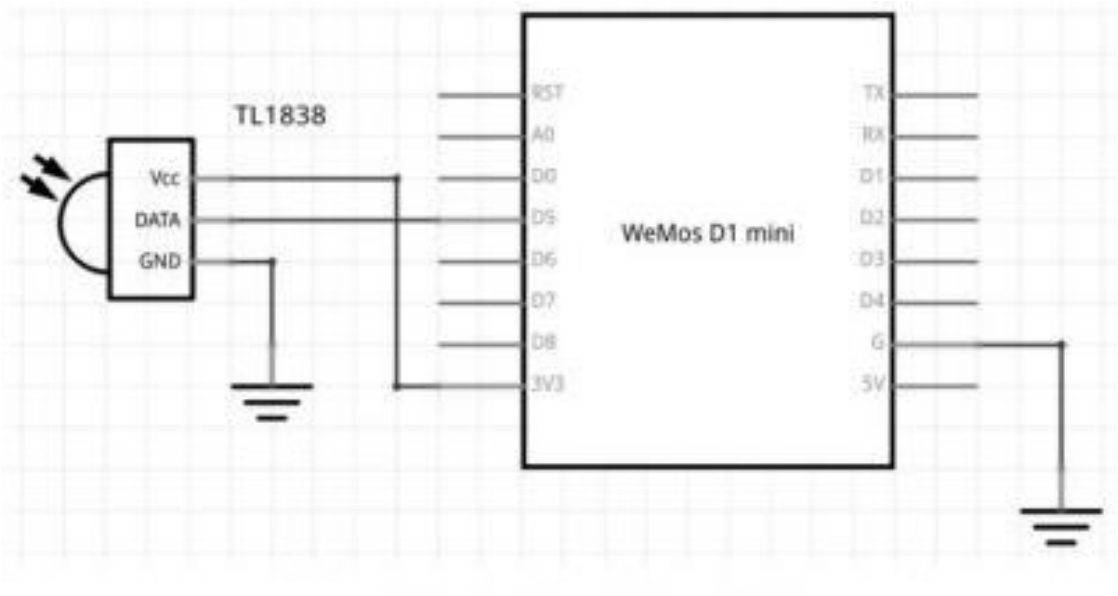

Рисунок 3.5 – Схема ІЧ–приймача

Особливості Wi-Fi Kit 8, що використовується:

– Мікропроцесор ESP8266, RISC 32-бітний CPU на частоті 160 МГц
– 32 кіБ Інструкційної RAM в залежності від моделі
– 96 кіБ RAM користувача + системних даних
– 4 МБ флеш-пам'яті
– IEEE 802.11 b/g/n Wi-Fi
– 17 GPIO-виводів (нерозпаяних)
– Налаштування напруги USB для живлення
– Вбудований OLED-дисплей 128x32 точок
– Інтерфейс для батареї з можливістю зарядки.

Плата Heltec Wi-Fi Kit 8 зображена на рисунку 3.6.

Основні відмінності від плати NodeMCU - це OLED-екран та інтерфейс батареї. Саме ці особливості стали причиною, чому обрано саме цю модель для

роботи в якості контролера, незважаючи на те, що вона коштує майже вдвічі дорожче, ніж плата NodeMCU.

OLED-екран використовується для відображення користувальницьких меню (які навігуються коротким натисканням та підтверджуються вибором довгим натисканням), а також QR-коду. Цей QR-код є способом спілкування пристрою з користувачем, щоб дозволити йому отримати випадково згенерований ключ. Цей QR-код можна легко зчитати за допомогою камери мобільного телефону.

Рисунок 3.6 – Плата Heltec Wi-Fi Kit 8

Інтерфейс батареї дозволяє користувачеві вільно рухатися по своєму будинку, спрямовуючи пристрої Інтернету речей на комунікацію з ними та надсилання ключа AP. Цю батарею також можна заряджати, просто підключивши плату до комп'ютера за допомогою кабелю USB.

Оскільки ця плата повністю відрізняється від NodeMCU, вона також видає різний рівень струму у своїх GPIO-пінах, що виявляється вищим, ніж у платі NodeMCU. Тому в цьому випадку вирішено просто підключити ІЧ-світлодіод та

приймач безпосередньо до плати, оскільки струм, що подається на світлодіод, був достатнім, а комунікація працювала правильно таким чином.

Даний дизайн пропонує створення ізольованої мережі для пристроїв Інтернету Речей всередині наявної домашньої мережі. Тому було потрібно пристрій, що може забезпечувати Wi-Fi зв'язок, функції маршрутизатора та брандмауера, а також підтримку скриптів. Ми вибрали Raspberry Pi через його універсальність та можливість оновлення, а також через широкий вибір відкритих бібліотек, які можуть бути використані для надання додаткових функціональностей.

Raspberry Pi 3b+ має наступні характеристики:

– 64-бітний процесор з тактовою частотою 1,4 ГГц;
– 1 Гб LPDDR2 SDRAM;
– Модуль Wi-Fi на частотах 2,4 та 5 ГГц, IEEE 802.11b/g/n;
– Модуль Bluetooth 4.2;
– Ethernet через USB 2.0 порт;
– Повноформатний порт HDMI;
– Підтримка камери;
– Підтримка сенсорного екрану;
– Відео та аудіо в стереоформаті;
– Порт Micro SD Card для зберігання операційної системи та даних;
– 40 GPIO пінів.

Для того, щоб завантажити згенерований ключ на RaspPi та створити точку доступу, передбачено три альтернативи:

1) Підключення до неї через сеанс SSH та запуск програмованого заздалегідь скрипту Python, вручну вставляючи інформацію про облікові дані.

2) Сканування QR-коду за допомогою камери Rasp Pi. У цьому випадку якість фокусування камери недостатньо хороша, щоб прочитати його безпосередньо з екрану Heltec Wi-Fi Kit 8. Користувач повинен буде прочитати його за допомогою мобільного телефону та відповідної програми, яка може

перемикатися між декодованою інформацією та прочитаним QR-кодом. Таким чином, камера Rasp Pi нарешті зможе прочитати більший код з екрана смартфона.

3) Введення коду за допомогою ІЧ-зв'язку між Rasp Pi та контролером ESP8266. Однак ця альтернатива була відхилена через проблеми як з програмним забезпеченням (які будуть описані пізніше в розділі Програмне забезпечення), так і з апаратним забезпеченням, через які не була можлива розробка цієї альтернативи.

Зображення плати Raspberry Pi 3b+ на рисунку 3.7.

Рисунок 3.7 – Порти та Pin-коди Raspberry Pi 3b+

3.3 Програмне забезпечення

3.3.1. ESP8266

ESP8266 використовується як контролер та симулятор вузла Інтернету речей (IoT). Тому було потрібно розробити дві різні програми на мові C для забезпечення комунікації між ними та виконання вимог розробки. Було написано дві програми:

1. Програма приймача: Вона реалізована в кожному вузлі Інтернету Речей та очікує на комунікацію з контролером, взаємодіє з ним та зберігає отримані облікові дані в пам'яті EEPROM. Нарешті, вона підключається до точки доступу, що відповідає отриманим обліковим даним.

2. Програма контролера: Вона працює на платі ESP8266 Wi-Fi Kit 8. Окрім того, що дана програма реалізує ту ж архітектуру комунікації та рутини, що й її аналог, вона також реалізує інші функціональні можливості, такі як генерація справжнього випадкового SSID та ключа, керування екраном OLED, кодування та відображення QR-коду та меню інтерфейсу користувача. На відміну від кожного приймача, контролер не буде намагатися підключитися до точки доступу.

У зв'язку з високим рівнем деталізації наступні підрозділи не включені в загальний зміст документа. Структура частини про ESP8266 наступна:

1) ESP8266 Контролер:
– Керування екраном OLED та взаємодія з користувачем;
– Керування пам'яттю EEPROM;
– Генерація випадкових SSID та PSK-ключів;
2) ESP8266 Приймач:
– Керування пам'яттю EEPROM;
– Підключення до точки доступу;
3) Двонаправлена передача через контролер та приймач:
– Процес проектування ІЧ–зв’язку;
– Тип пакетів і дизайн кадрів ІЧ–пакетів;
– Шифрування даних;
– Діаграма зв’язку ІЧ–протоколу;
– Аналіз еволюції часу.

3.3.1.1 Оперування та взаємодія з OLED екраном

Як показано на зображенні у розділі про обладнання, до плати, що використовується в якості контролера, вбудований OLED дисплей з розмірами 128x32 пікселів. Для того, щоб друкувати інформацію на екрані, використовується бібліотека Adafruit_SSD1306[]. Після її ініціалізації, ми можемо використовувати

бібліотеку Adafruit, щоб друкувати інформацію на OLED екрані так само, як ми це робимо на консолі.

Взаємодія користувача з екраном обмежена двома меню вибору (в основному використовуються для розробки, в комерційній версії вони можуть бути скасовані) та одним кнопковим елементом. Існують два меню, які запитують введення користувача, коли пристрій вмикається:

– Меню вибору вузла: дозволяє розробнику примусити комунікацію з вже збереженим пристроєм у разі зміни облікових даних. За замовчуванням вибрано додавання нового пристрою за допомогою широкомовного пакету, проте вже ініціалізований пристрій проігнорує цей тип пакетів.

– Меню вибору шифрування: дозволяє користувачеві комунікувати зашифрованим способом (за замовчуванням) або відправляти облікові дані у відкритому вигляді. Шифрування збільшує рівень безпеки, але також збільшує час необхідний для передачі даних. Ймовірно, комерційний пристрій повинен пропустити це меню та примусово вимагати шифрування для кожної комунікації пристроїв.

Введення користувача відбувається за допомогою вбудованої кнопки «флеш» на платі. Одне натискання перемикає між варіантами в меню, а довге натискання підтверджує вибраний варіант. На екрані відображається виділений варіант в кожний момент. Після того, як користувач підтверджує обидва меню, контролер намагається спілкуватися з вузлом.

Після успішного завершення зв'язку, на екрані буде відображений QR-код, який кодує згенеровані та передані облікові дані. Для генерації цього коду використовувалася бібліотека з назвою QRCode[]. Незважаючи на те, що бібліотека була призначена для друку QR-коду на терміналі, було легко роздрукувати її на OLED-екрані за допомогою функції printPixel() бібліотеки екрану. Кількість інформації, яку можна закодувати в QR-коді, залежить від його розміру та можливостей коригування помилок (EC). Чим більший розмір та менші можливості коригування помилок, тим більша кількість даних може бути закодована.

QR-коди класифікуються за їх версією та EC. Розмір QR-коду визначається за формулою: size = 4 * version + 17.

Оскільки експерементальний екран обмежений висотою в 32 пікселі, ми можемо реалізувати лише версії 1 (21x21), 2 (25x25) та 3 (29x29). Версія 4 перевищує ліміт висоти, оскільки її розмір складає 33x33.

Також важливо перевірити кількість інформації, яка буде закодована. В даному проекті кодується SSID (16 байтів) та PSK (32 байти). Це складає загалом 48 байтів, крім того, було б розумним додати деякі додаткові символи для розділення обох рядків, щоб користувач міг відрізнити SSID та PSK після прочитання коду.

Через ці обмеження існує лише одна можлива комбінація, яка відповідає всім вимогам.

Це версія 3, 29x29 з низьким рівнем корекції помилок. Цей код дозволить закодувати до 53 байтів, що означає, що є 5 додаткових байтів, щоб додати деякі спеціальні символи (такі як пробіли, розриви рядків, тощо), які розділять обидві рядки.

В прийнятій реалізації використовується низький рівень корекції помилок, що може призвести до того, що навіть один мертвий піксель на OLED екрані зробить QR-код нечитабельним.

Крім того, це обмеження не дозволяє використовувати більший приватний ключ, наприклад, 64 байти, оскільки його не можна закодувати в коді 29x29.

Іншою критичною проблемою використання такого маленького екрану є складність його читання за допомогою звичайної камери. Розмір пікселів та підсвітка екрана можуть запобігти деяким мобільним телефонам виявляти та сканувати код.

Після багатьох спроб стало зрозумілим, що інвертування кольорів QR-коду робить його легше зчитувати за допомогою телефона. Замість використання білих пікселів на чорному фоні, зробилено весь екран білим і вимкнуто включені точки

QR-коду. Це виправлення працює, оскільки допомагає камері налаштувати експозицію світла та спрямувати код з більшою точністю.

Доброю ідеєю було б використовувати плату NodeMCU як контролер Інтернету Речей та підключати до неї зовнішній OLED екран розміром 128x64 пікселів. Це полегшить роботу.

У цьому випадку ми могли б:

1) Зберігайте версію QR-коду та збільшуйте розмір кожної точки QR до 2x2 пікселів, щоб зробити його легше сканувати, або

2) Зберігайте 1 піксель на точку QR і використовуйте QR-версію 11 (61x61) з високим рівнем корекції помилок. Це дозволить збільшити розмір ключа PSK до 64 байтів і буде можливість його закодувати, підвищуючи стійкість QR-коду.

Однак, більший екран зробить контролер менш зручним у використанні, зменшуючи його портативність та час роботи від батареї.

3.3.1.2 Управління пам'яттю EEPROM

Для того, щоб інформація залишалась постійною, використовується вбудована пам'ять EEPROM, яку інтегрує ESP8266. Хоча доступно більше пам'яті, контролер використовує лише 256 байт пам'яті.

Існує бібліотека (включена в саму devkit ESP8266) під назвою EEPROM, яка спрощує запис/читання.

Однак на основі цієї бібліотеки включено деякі власні функції для зчитування/запису інформації набагато простіше.

Кожна адреса пам'яті може зберігати до одного байту даних. Схема пам'яті, яка надає інформацію про те, де зберігається кожен елемент даних в пам'яті, виглядає наступним чином(Таблиця 3.2):

Перші 48 байтів зарезервовано для інформації про SSID та PSK точки доступу. Наступний байт, 0x30, вказує на кількість вузлів, які були ініціалізовані та мають облікові дані в своїй пам'яті.

Це значення дозволяє пристрою знати, де зупинитися при читанні пам'яті для ідентифікаторів вузлів.

Таблиця 3.2 – Відображення пам’яті EEPROM контролера

Відображення EEPROM контролера(256 біт)		
Адреса	Інформація	Загальний розмір
0x00-0x0F	SSID	16
0x10-0x2F	PSK	32
0x30	Число вузлів	1
0x31	Вузол 1	1
0x32	Вузол 2	1
---	---	---
0xFE	Вузол 206	1
0xFF	Control Char	1

Оскільки кількість вузлів - це лише один байт, це обмежує кількість вузлів до 28 = 256 пристроїв. Тим не менш, оскільки деякі адреси використовуються для облікових даних Wi-Fi, кількість збережених пристроїв менша - 206. Однак цього числа повинно бути достатньо для більшості випадків.

Нарешті, в останній використаній адресі пам'яті є важливий символ - 0xFF. Адреса керуючого символу є першою, яка читається при запуску пристрою. Він дозволяє пристрою визначити, чи була пам'ять ініціалізована раніше чи ні. Якщо значення, прочитане, дорівнює символу «Y» або 0x59 в шістнадцятковому форматі, це означає, що в пам'яті вже є дані, збережені в інших адресах. В іншому випадку пристрій буде випадковим чином генерувати SSID та PSK та ініціалізувати кількість вузлів на 0.

3.4 Генерація випадкового SSID та PSK ключа

Однією з найважливіших функцій ESP8266 є його здатність генерувати справжні випадкові числа з високорівневих джерел ентропії. Однак, стандартна функція генерації випадкових чисел в C не є справжньою випадковою, тому була використана зовнішня бібліотека. Ця бібліотека - ESP8266TrueRandom. Ця бібліотека отримує ентропію, читаючи регістр генератора випадкових чисел ESP8266 або вимірюючи рівень напруги на піні A0/TOUT.

1. Генерація SSID: SSID має 16 байт, формат: IOT_aaaaaa-nnnnn. Починається з префіксу "IOT_", 6 символів, роздільника "-" та 5 цифр. Кожен символ 'a' представляє будь-яку літеру, нижнього або верхнього регістру, англійського алфавіту (52 можливості на символ). Символ 'n' може бути будь-якою цифрою від 0 до 9 (10 можливостей на символ). Випадковість SSID не є критичною, оскільки він стане загальнодоступним значенням при анонсуванні точки доступу.

2. Генерація PSK: Метод, який використовується для генерації PSK, досить простий:

– Функція random() зі спеціальної бібліотеки, яка генерує число з 32 біт, викликається 4 рази поспіль.

– Далі, кожне число додається до попереднього, що призводить до 128-бітного випадкового числа.

– Нарешті, розбивається ланцюжок 128 біт на 32 шматки по 4 біти, що дає 32 випадкових шістнадцятирічних числа.

– Дані шістнадцятирічні числа перетворюються з 4-бітових шістнадцятирічних символів у своє ASCII-представлення (8 біт). Це перетворення подвоює довжину ключа до 256 біт, забезпечуючи читабельність інформації. Наприклад, символ 'C' у шістнадцятирічній формі буде 0b1101, а його перетворення в ASCII - 0b1000 0011.

У кінці, незважаючи на те, що є 32 символи або 256 біт інформації, справжня ентропія полягає у 128 бітах через доповнення.

3.5 ESP8266 Receiver (дошка NodeMCU)

Подібно до частини контролера, розділ 3.5 охоплює код, спеціально розроблений для пристроїв-отримувачів.

Керуючись тими ж принципами та механікою, що й у контролері, управління пам'яттю в приймачі використовується для читання та запису постійних даних. У цьому випадку загальна використана пам'ять - 64 байти. У цих пристроях кількість збереженої інформації нижче, оскільки немає потреби зберігати повний список ініціалізованих пристроїв.

Також використовуються ті ж спеціальні функції для зручного читання запису в EEPROM. Спосіб організації інформації в пам'яті приймача зображено в Таблиці 3.3:

Таблиця 3.3 – Відображення пам’яті EEPROM-приймача

Відображення EEPROM-приймача(64 біти)		
Адреса	Інформація	Загальний розмір
0x00-0x0F	SSID	16
0x10-0x2F	PSK	32
0x30	Dev ID	1
0x31-0x3E	Unused	14
0x3F	Control Char	1

Зі 64 призначених байтів, 14 не використовуються. Однак, є доцільним виділити обсяг пам'яті, що є степенем двійки (28).

Також, як і в разі контролера, є контрольний символ, який, якщо дорівнює символу «Y», означає, що пам'ять була ініціалізована. У цьому випадку пристрій буде прямо читати SSID та PSK і намагатися підключитися до точки доступу.

Нарешті, на відміну від випадку контролера, де немає необхідності зберігати його власний ідентифікатор пристрою, оскільки він закодований наперед і завжди дорівнює 1, приймачу необхідно зберігати свій ідентифікатор в разі вимкнення живлення. ID використовується для ідентифікації пристрою як у інфрачервоній комунікації з контролером, так і для визначення його статичної IP-адреси.

3.6 Підключення до AP (точки доступу)

Набір розробки ESP8266 Arduino містить бібліотеку для керування Wi-Fi з'єднанням в пристрої. Вона також включає легковагий стек протоколів IP в чипі. Бібліотека називається WiFiClient[] та дозволяє пристрою діяти як точка доступу або як станція та підключатися до іншої AP.

Мережа, що використовується для IoT екосистеми, має підмережу 192.169.100.0/24. Шлюз - 129.168.100.1, а кожний пристрій IoT запитує статичне з'єднання на основі свого ID. Кожен пристрій запитує з'єднання з адресою типу 192.168.100.devID.

Поки пристрій підключений до точки доступу, він буде слухати інфрачервоний зв'язок від контролера, який надсилає свій власний ID. Це дозволяє контролеру змінювати збережені облікові дані пристрою.

Оскільки визначення рівня застосування не є предметом дослідження цієї роботи, то після підключення пристрою до точки доступу його функціональність буде обмежено. Однак він все ще буде відповідати деяким протоколам нижніх рівнів, таким як ICMP, і пристрій відповідатиме на будь-який запит ping.

3.7 Дизайн обладнання

3.7.1 Дизайн ESP8266

При першому налаштуванні спостерігалося, що ІЧ-світлодіод та приймач працюють добре, якщо підключені безпосередньо до контактів плати. Але після розробки більш складного протоколу передачі, було виявлено, що струм ІЧ-світлодіода не відповідає вимогам, а висока кількість помилок призводить до зіпсованих пакетів.

Крім того, з’ясувалося, що існують різні плати NodeMCU, які можуть виглядати схоже, але мати деякі відмінності, і, в кінці кінців, струм, що подається на їх контакти, трохи різний, що призводить до проблем при перемиканні ІЧ-світлодіода для передачі.

3.7.2 Протокол передачі ІЧ-сигналу

Це завжди було основною проблемою розробки, і було б точніше вважати його великою кількістю невеликих взаємопов'язаних проблем, аніж однією великою. Крім того, попередній пункт зробив протокол передачі ще більш непередбачуваним до тих пір, поки не вирішено питання з апаратними проблемами.

Список проблем протоколу досить великий, і багато з них вже були описані в документі: перетворення одностороннього протоколу без зворотного зв'язку в бічно-визнане, збільшення швидкості передачі, змінюючи сам протокол RC6, боротьба з високою BER, додавши два ІЧ-світлодіоди замість одного, створення десятків версій та ітерацій протоколу, тестування великої кількості підходів до визначення, яка версія працює краще тощо.

3.7.3 Комунікація від пристрою до користувача

Ще однією важливою проблемою є комунікація від пристрою до користувача. Оскільки ім'я мережі (SSID) та пароль (PSK) генеруються внутрішньо контролером, користувач не має прямого доступу до цієї інформації. Доступ до неї через кабель не є варіантом, оскільки контролер був розроблений для мобільного використання та має бути доступним з будь-якої точки.

З цими обставинами на увазі вирішено використовувати іншу плату як контролер, яка містить вбудований OLED-екран, таку як Wi-Fi Kit 8. Однак, як можна вважати, пікселі екрану виявилися досить малими, і в результаті сканування QR-коду не можна вважати задовільним досвідом, оскільки доводиться мати проблеми з фокусом та відстанню камери, доки мобільний телефон нарешті декодує QR-код.

Отже, цю проблему можна вважати частково вирішеною, оскільки було б цікаво спробувати інший екран з більшими пікселями та перевірити, чи легше сканувати QR-код.

3.7.4 Дизайн Інтернету Речей Роутера

Планувалося використовувати комерційну модель як Інтернету Речей маршрутизатор, для більш реалістичного сценарію. Однак, відсутність програмованих комерційних роутерів та їх висока ціна змусили вибрати Raspberry Pi 3b+ через його універсальність, потужність обробки та налаштування.

3.7.5 ІЧ-Комунікація від Контролера до Інтернету Речей роутера

ІЧ-комунікація між Контролером та Інтернету Речей роутером була початковою ідеєю для передачі облікових даних до маршрутизатора. Однак, ці два пристрої є абсолютно різними, і після вивчення багатої документації та

інтенсивних тестів було вирішено знайти альтернативи. Основна проблема полягає в тому, що необхідно розробити протокол для роутера, який повністю сумісний з тим, що вже використовує контролер. Крім того, цей протокол має бути повністю прозорим для контролера, що означає, що контролер буде підключатися та передавати інформацію до роутера точно так само, як і з будь-яким вузлом.

3.8 Висновки

Розглянуті в третьому розділі архітектура, апаратне та програмне забезпечення, процес генерації ключів аутентифікації, дизайн обладнання та комунікації від пристрою до користувача є необхідними елементами для реалізації методу та програмно-технічного засобу генерування ключів аутентифікації.

Зокрема, було досліджено апаратне забезпечення, таке як ESP8266 NodeMCU, а також розглянуто проектування схеми ІЧ світлодіодного передавача. Далі, було проаналізовано програмне забезпечення, зокрема ESP8266, операції взаємодії з OLED екраном та управління пам'яттю EEPROM. Для генерації випадкового SSID та PSK ключа було розроблено відповідний процес.

Дизайн обладнання було досліджено з різних аспектів, включаючи дизайн ESP8266 та протокол передачі ІЧ-сигналу. Було також досліджено комунікацію від пристрою до користувача, а також розроблено дизайн Інтернету Речей Роутера та відповідний процес ІЧ-комунікації від контролера до Інтернету Речей роутера.

Узагалі, третій розділ магістерської роботи містить важливі деталі та інформацію, яка допомогла створити метод та програмно-технічний засіб генерування ключів аутентифікації.

4 РЕЗУЛЬТАТИ ТА АНАЛІЗ РОЗРОБЛЕНИХ МЕТОДУ ТА ПРОГРАМНО–ТЕХНІЧНОГО ЗАСОБУ

4.1 Архітектура зв’язку ІЧ-протоколу

Описана кодова реалізація кожного етапу та процесу зв’язку на рисунку 4.1.

Однак, цей робочий процес не відображає складність самої фази комунікації. Як детальна альтернатива, наступна графіка зосереджена на набагато меншому рівні, показуючи спосіб, яким кожен пакет відправляється на кожному етапі. Вона представляє найскладніший випадок: підключення до нового вузла з шифруванням.

Також вона дає чітку відповідь на те, як в цілому здійснюється послідовність передачі, як і коли надсилається пакет, а також час, витрачений на кожну з фаз комунікації.

Оскільки нові версії були програмовані з подальшими оптимізаціями та оновленнями, цікаво порівняти два різних підходи, щоб побачити різницю та зрозуміти, як кожне рішення щодо розробки може вплинути на проект щодо продуктивності, зручності використання, адаптації та пам'яті.

4.2 Аналіз третьої робочої версії

Дана діаграма на Рис.16 була створена на основі логів терміналу зв'язку в реальному сценарії зв'язку між нашим контролером та приймачем, використовуючи третю робочу версію. Кожен переданий пакет був представлений, а відображені часи (в мілісекундах) точно відображають час, який займало кожну з фаз зв'язку на основі логів.

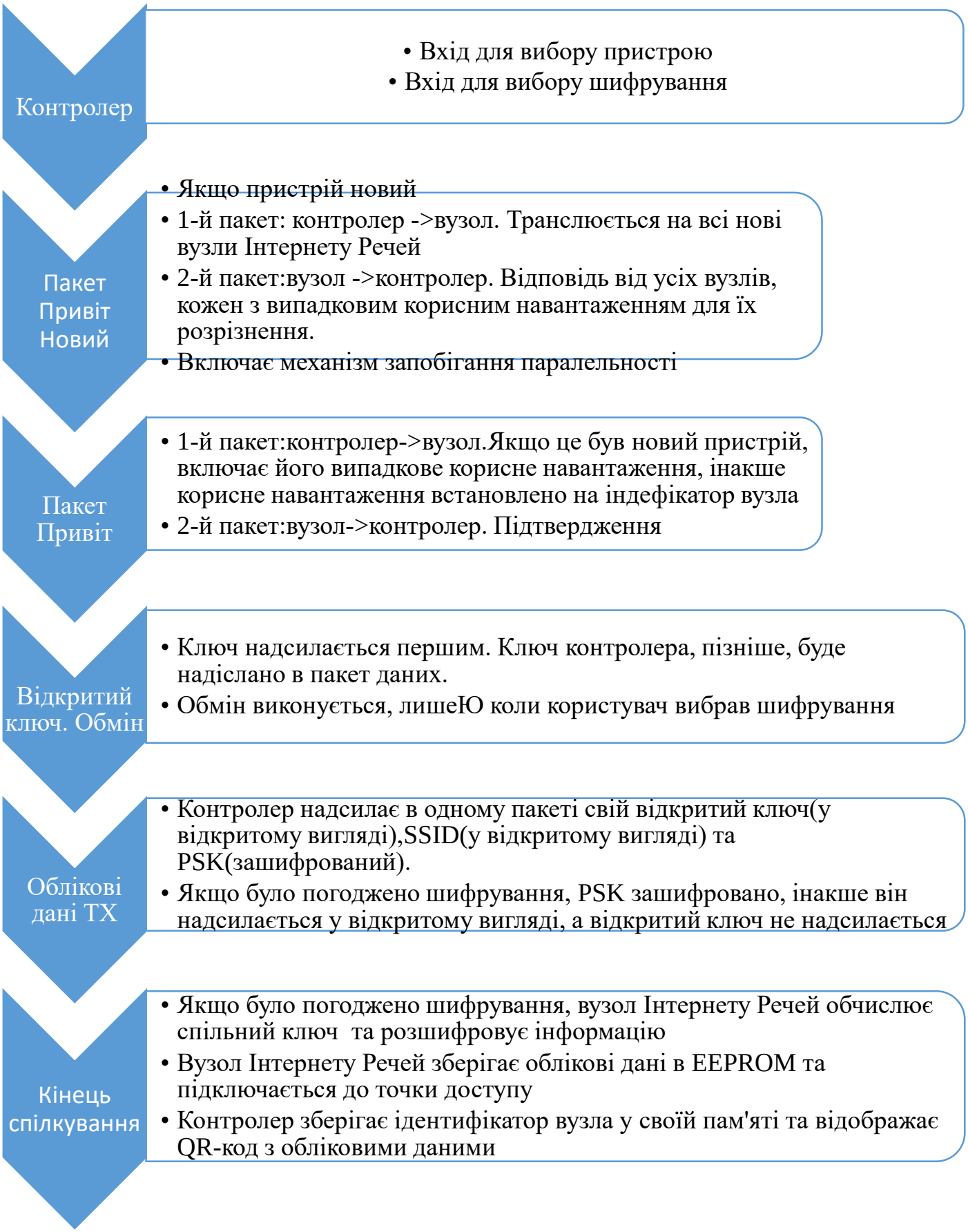

Рисунок 4.1 – Робочий процес підсумкової передачі ІЧ-протоколу

Це починається з передачі першого пакета Привіт Новий, який надсилається контролером, коли пристрій вмикається та підтверджуються нові пристрій та меню шифрування. У цей момент контролер періодично передаватиме пакет, поки не отримає відповідь. У нашому сценарії першим пакетом зазвичай є той, на який отримано відповідь, проте відповідь не дійде до контролера, поки він не надішле Привіт Новий 2.

Привіт Новий, відправлений спочатку контролером, матиме ідентифікатор призначення в його навантаженні (у нашому випадку id=2). Тому приймач знає, що наступний пакет Привіт, якщо містить 2, буде відправлений до нього.

Привіт Новий 2 реалізує механізм запобігання конфліктів за допомогою випадкового числа з 24 бітами в його навантаженні.

Цей пакет Привіт Новий 2 також періодично надсилається, поки не припинить отримувати пакети Привіт Новий. Цей момент є найбільш критичним у реалізації, оскільки пристрої ще не синхронізовані, і можуть виникати конфлікти. Як захисний захід, цей період передачі Привіт Новий 2 було зроблено випадковим, щоб уникнути множинних зіткнень через постійну синхронізацію.

На діаграмі початкові 2 пакети Привіт Новий 2, були втрачені, ймовірно, через колізію з другим пакетом Привіт Новий, відправленим контролером.

Однак, нарешті третій пакет дійшов до свого призначення та повідомив контролер, що новий не конфігурований вузол слухає.

Контролер припинить передачу Привіт Новий та негайно передасть Привіт. Колізії все ще можуть відбуватися, оскільки приймач продовжує періодично відправляти пакети Привіт Новий 2.

Таймери відправки були оптимізовані для запобігання колізіям на цій стадії, що перекладається у те, що перший або другий пакет Привіт зазвичай отримується приймачем.

У цьому тесті перший пакет Привіт правильно отриманий вузлом Інтернету Речей. На цьому етапі вузол відправить тричі поспіль остаточний пакет Привіт 2. Хоча перший пакет правильно отримано, контролер буде тримати вікно прийому

відкритим, оскільки він знає, що приймач відправить його тричі, і до того моменту, поки він не закінчиться, він не зможе слухати жодних вхідних пакетів.

Через введення користувачем початкової конфігурації, у цьому прикладі діаграми вибрано режим шифрування.

Тому пакети Привіт, відправлені контролером, насправді є пакетами Привіт Шифрований, які повідомляють вузлу приймача, що комунікація буде здійснюватися зашифровано, і їм спочатку потрібно обмінятися своїми публічними криптографічними ключами.

Після того, як Привіт підтверджено і вікно прийому закрите, контролер надсилає вибух даних з 9 пакетів, що містять його публічний ключ.

Довжина публічного ключа складає 32 байти, кожен пакет даних може містити до 4 байтів, тому в кінці є 8 пакетів, що містять дані, і додатковий пакет, що містить контрольну суму попередніх 8 пакетів.

Зазвичай, у випадку невдалої перевірки контрольної суми через відсутність пакета, вибух даних повторно надсилається до отримання підтвердження (Burst ACK). У цьому тесті всі пакети були отримані з першої спроби, тому повторної передачі не було.

Важливою характеристикою передачі даних є швидкість передачі, яка зазвичай залежить від використовуваних протоколів та засобів комунікації. Наприклад, більшість бездротових мереж Wi-Fi підтримують стандарт 802.11ac з максимальною швидкістю передачі даних більше 1 Гбіт/с.

Крім того, існують різні методи оптимізації передачі даних, такі як компресія даних, яка дозволяє зменшити розмір передаваних даних, і кешування, яке зберігає попередньо отримані дані для швидшого доступу до них у майбутньому.

У разі використання протоколів, які підтримують відновлення після збоїв, передача даних може бути надійнішою та ефективнішою, оскільки ці протоколи дозволяють передачу даних з відновленням після збоїв без необхідності повторного відправлення всього пакета даних.

У випадку невдачі відправник увійде в стан тайм-ауту ACK та повторно надішле весь вибух даних.

Немає функціональності NACK або способу виявлення пакета, який був пошкоджений або не досяг призначення.

Burst ACK є критичною частиною комунікації. Оскільки немає функціональності подвійного ACK для прискорення процесу, ACK надсилається завжди 4 рази поспіль, щоб переконатися, що вони отримані хоча б один раз.

Якщо будь-яка інформація правильно отримана, а всі ACK пропущені, приймач перейде до наступного стану, тоді як передавач буде продовжувати відправляти вибух даних.

Обидва пристрої більше не зможуть зв'язуватися між собою, тому в кожному етапі комунікації була прийнята глобальна схема тайм-ауту.

Якщо етап комунікації займає в 3 рази більше часу, ніж зазвичай, процес почнеться з початку.

Механізм повної скидання процесу є екстремальним рішенням для дуже рідкісної ситуації.

У звичайних тестах ця обставина стається лише у випадку, якщо її спровокувати вручну (перериваючи лінію зору), оскільки зазвичай всі пакети правильно отримуються вперше, як тільки було завершено рукостискання. Отримавши PK контролера, приймач відправляє свій використовуючи точно такий самий алгоритм.

Контролер отримує його, підтверджує отримання і одразу починає відправляти дані для підключення до точки доступу в одному потоці.

Ця передача була оптимізована, щоб вона помістилася в 8 пакетів даних. Ім'я SSID міститься в перших 4 пакетах, відкритим текстом, тоді як PSK шифрується в наступних 4 пакетах даних.

Останній пакет даних буде містити контрольну суму попередніх 8 пакетів. Весь процес комунікації реального тесту, який відображено на діаграмі, зайняв менше ніж 4,5 секунди, як ми можемо побачити.

Найповільнішою та найменш ефективною частиною у рукостисканні є та, яка стикається з передачею даних у випадку повторного надсилання та зіткнень, в той час як вона є найповільнішою частиною, коли фактично передається менше інформації. Фактично, весь потік даних з 9 пакетів та їх ACK відправляється за близько 30-40% менше часу.

Приклад Третьої робочої версії на рисунку 4.2.

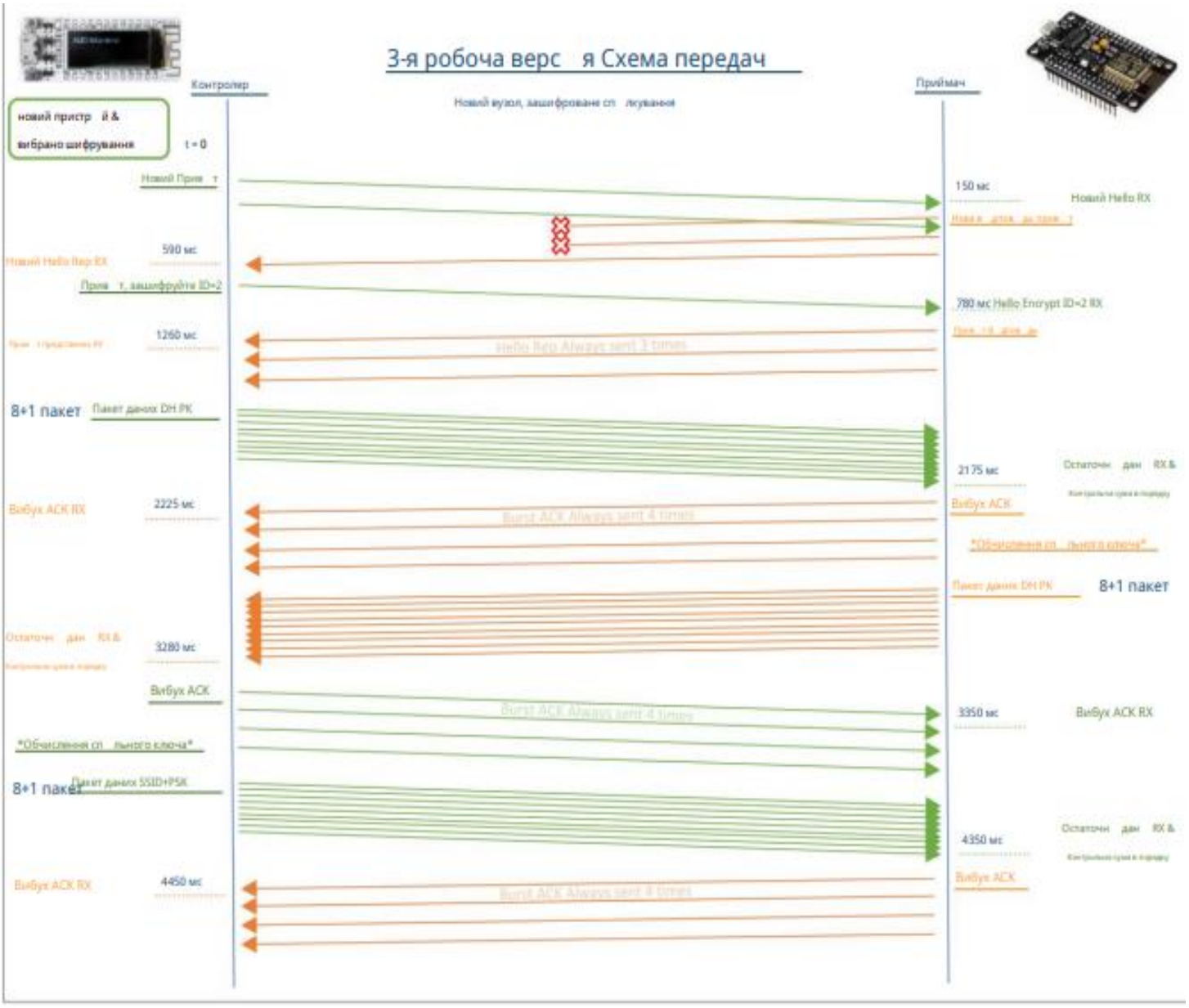

Рисунок 4.2 – Діаграма Третьої робочої версії схеми передач

4.3 Аналіз четвертої останньої робочої версії

Поточна робоча версія, яка є четвертою, розроблена у вересні 2019 року, зменшує затримку ще на 25% порівняно з попередньою версією. Її повна діаграма передачі зображена на рисунку 4.3.

Ця реалізація відрізняється тільки в загальній кількості передач даних. У 3-й версії була реалізована окрема процедура обміну ключами, тому в кінці контролер відправляв дві послідовності даних з 8+1 пакетів кожна. Цей дизайн є повністю логічним для більшості випадків. Однак, в цьому випадку, коли передається дуже мало реальної інформації (тільки 32 байти), неефективно створювати дві послідовності даних і надсилати їх незалежно.

У цій 4-й версії був вилучений Burst обміну публічним ключем контролера, а 32 байти ключа були включені в послідовність даних. Це дуже економить час, оскільки найменш ефективний момент передачі - це вікно ACK, в якому той самий маленький шматок інформації обмінюється кілька разів протягом великого вікна часу, щоб гарантувати прибуття підтвердження.

Також, розрахунок загального ключа та розшифрування інформації були перенесені на момент після закінчення передачі у вузлі Інтернету Речей. Це рішення зберегло ще більше часу в комунікації, оскільки розрахунок загального ключа та розшифрування інформації можуть зайняти до 0,5 секунди. Однак у контролері розрахунок загального ключа потрібно зробити одразу після отримання публічного ключа вузла, оскільки загальний ключ буде негайно використовуватися для шифрування інформації перед її відправкою. Це означає втрату 0,4-0,5 секунд, що затримує весь процес передачі.

Удосконалення у 4-й версії забезпечують час 3,7 секунд до отримання ACK даних в контролері. Це зменшення на 0,75 секунди порівняно з попередньою версією, що становить 15% менше часу.

Додатково потрібно 0,4 секунди пізніше в приймачі для обробки інформації, але цей крок не належить самій комунікації, оскільки контролер не бере участь в цій стадії.

Якщо інформація надсилається відкритим текстом, часу не економиться, оскільки ключів не існує, і процес комунікації майже ідентичний 3-й версії.

Приклад Четвертої робочої версії на рисунку 4.3.

4.4 Аналіз часу робочих версій

Затримка під час передачі є однією з найважливіших змінних, які потрібно враховувати. Затримка передачі для системи безпечного введення ключа через інфрачервоне випромінювання може зробити корисний дизайн непридатним для комерційного використання, оскільки кінцевий користувач не прийме витрачання занадто багато часу на такий простий процес. Було виконано багато роботи з метою мінімізації цієї затримки.

Одним з важливих чинників, що впливає на ефективність робочих версій, є аналіз часу роботи. Це дозволяє визначити, наскільки швидко працює програма і чи потрібні додаткові оптимізації. Для цього можна використовувати різні інструменти для аналізу продуктивності, такі як профілювальні програми або програмні засоби для вимірювання часу виконання.

Для того щоб забезпечити максимальну ефективність, можна використовувати різні стратегії оптимізації, такі як кешування даних, асинхронність та паралелізм обчислень. Ці стратегії дозволяють зменшити час виконання програми та забезпечити кращу відповідь для кінцевого користувача.

Окрім цього, важливо розуміти, які операції займають більше часу в програмі. Це дозволяє зосередитися на конкретних частинах програми, які потребують оптимізації. Наприклад, можна зменшити кількість операцій з файлами або використовувати більш ефективні алгоритми обчислень.

Один зі способів аналізу часу роботи полягає у використанні тестових наборів даних. Це дозволяє визначити час виконання програми на різних обсягах даних та визначити, як ефективно програма працює з різними розмірами вхідних даних.

Окрім цього, можна використовувати профілювальні програми для аналізу часу виконання різних функцій та методів в програмі (рисунок 4.3).

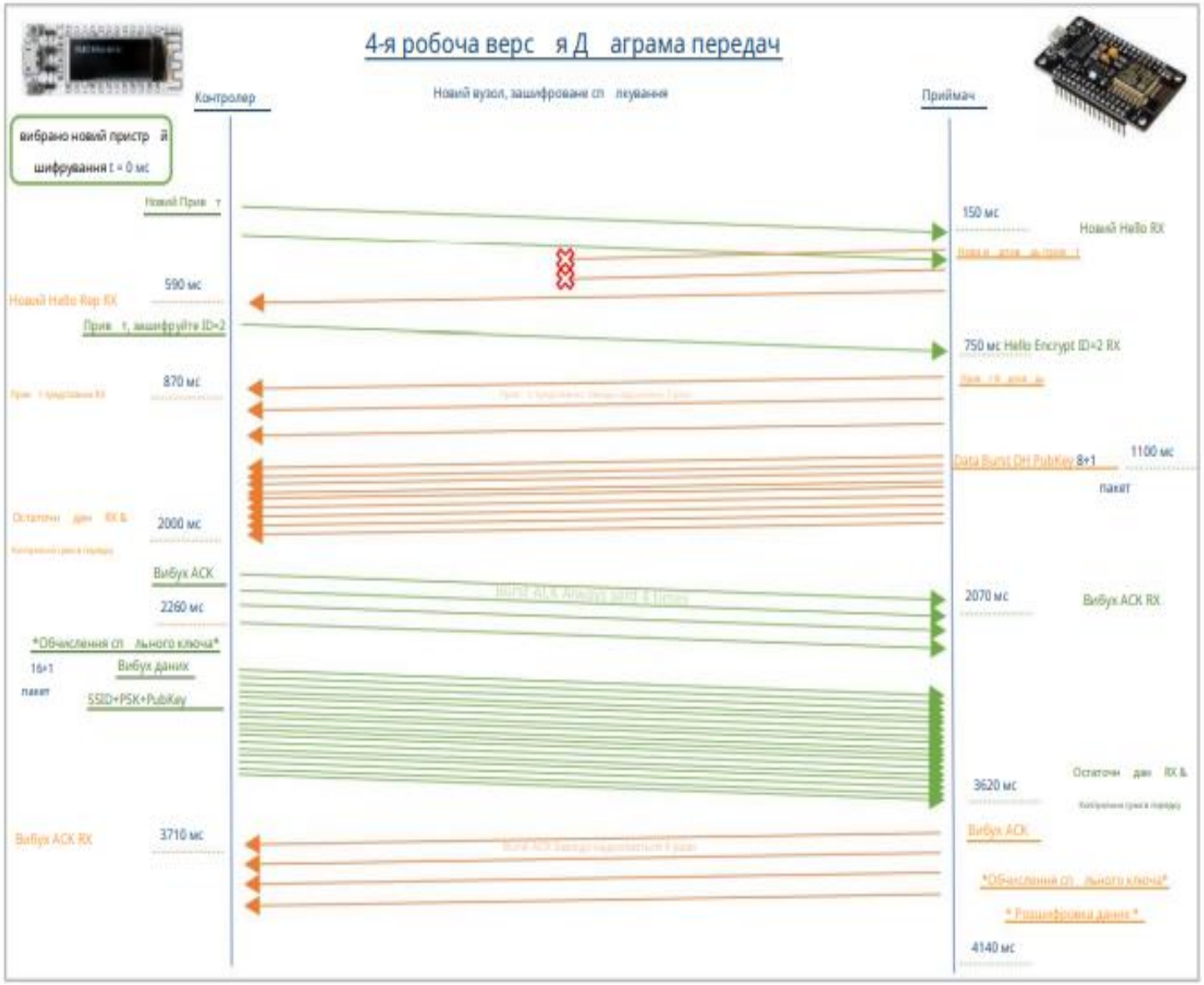

Рисунок 4.3 – Діаграма Четвертої робочої версії

Нижче, представлено діаграму порівняння різних часових графіків та підвищення ефективності з кожною новою версією. Усі відображені часи були

отримані з рутини зв'язку приймача в різні моменти розробки різних версій протоколу. Вони використовують як посилання час, в який перший пакет Привіт Новий від контролера надходить до приймача.

Щоб мати початкову точку посилання, перша робоча версія, яка не представлена тут через її ранній етап та неефективність, використовувала дизайн Stop&Go, і на передачу інформації відкритою мовою потрібно було майже 10 секунд. Було вирішено не включати діаграму через її надмірну неефективність.

Ці три часові лінії належать трьом різним підверсіям цієї другої робочої версії. У ній SSID та PSK були відправлені незалежно у різних спалахах.

Крім того, не було шифрування, тому процедура обміну публічним ключем не виконувалася. Це скорочує затримку, оскільки інформацію було надіслано чітко.

Приклад Першої версії на рисунку 4.4.

Графік прийому 15 лютого.Без шифрування.SSID та PSK в 2 пакетах.
Надсилання в чистому режимі - 3032мс

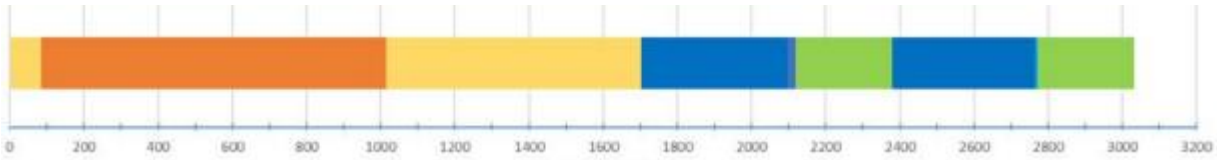

Рисунок 4.4 – Часова діаграма Другої версії (1)

1. Старт «Привіт Нов» (84)
2. Старт «Привіт» (931)
3. Надсилання «Привіт» (688)
4. Очікування SSID (394)
5. Збереження SSID (23)
6. Надсилання ACK SSID (260)
7. Очікування PSK (385)
8. Збереження PSK(6)
9. Надсилання ACK PSK (261)

Графік прийому 20 лютого.Без шифрування.SSID та PSK в 2 пакетах.

Надсилання в чистому режимі - 2556мс

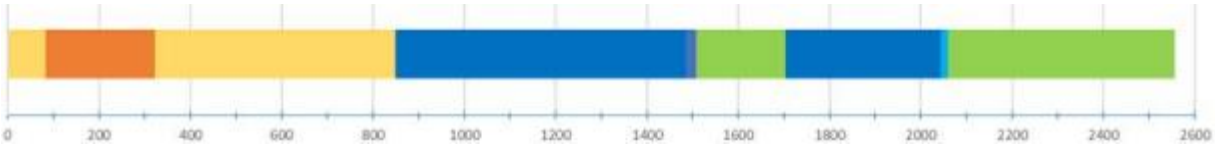

Рисунок 4.4 – Часова діаграма Другої версії(2)

1. Старт «Привіт Нов» (84)
2. Старт «Привіт» (931)
3. Надсилання «Привіт» (688)
4. Очікування SSID (394)
5. Збереження SSID (23)
6. Надсилання ACK SSID (260)
7. Очікування PSK (385)
8. Збереження PSK (6)
9. Надсилання ACK PSK (261)

Графік прийому 3 березня.Без шифрування.SSID та PSK в 1 пакеті.

Надсилання в чистому режимі - 2013мс

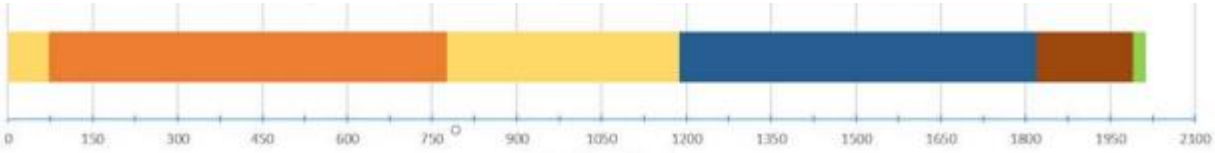

Рисунок 4.4 – Часова діаграма Другої версії(3)

1. Старт «Привіт Нов» (84)
2. Старт «Привіт» (931)
3. Надсилання «Привіт» (688)
4. Очікування SSID (394)
5. Надсилання даних (171)
6. Збереження даних (23)

У попередніх трьох графіках показано сприятливу еволюцію загального часу передачі. Ці три графіки належать одній і тій же версії з різними налаштуваннями.

Вдалося скоротити час з 3 до 2 секунд, що становить 33% ефективності. Це відбулося в основному завдяки двом різним налаштуванням:

– Модифікації протоколу IR RC6: параметри протоколу були налаштовані для зменшення порожніх періодів та підвищення швидкості.

– Логіка відправлення: у перші два графіки інформація про SSID та PSK відправляється в двох різних блоках або імпульсах.

У третьому графіку це було змінено, де як SSID, так і PSK включені в один більший імпульс. Це значно зменшило час, оскільки скасовано одну контрольну суму і тепер пропускається час очікування на підтвердження останнього імпульсу.

Однак жодна з розділів другої версії не використовувала шифрування, оскільки на той момент головною проблемою була швидкість та ефективність.

Як тільки було додано шифрування, часи логічно збільшилися(рисунок 4.5).

Рисунок 4.5 – Часова діаграма Третьої версії(1)

1. Старт «Привіт Нов» (73)
2. Старт «Привіт» (705)
3. Надсилання «Привіт» (410)
4. Очікування даних (632)
5. Надсилання даних (236)
6. Збереження даних (14)

Шкала прийому 3 березня(2 версія).

Реалізовано шифрування.Надсилання шифрування[4951мс]

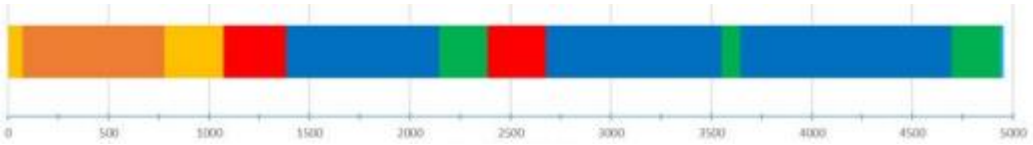

Рисунок 4.5 – Часова діаграма Третьої версії(2)

1. Старт «Привіт Нов» (73)
2. Старт «Привіт» (705)
3. Надсилання «Привіт» (295)
4. Розрахунок ключів шифрування (311)
5. Очікувааня Конторолера Pubkey (763)
6. Надсилання ACK Pubkey (239)
7. Збереження спільного ключа (17)
8. Надсилання власного власного ключа до контролера (874)
9. Очікування ACK PubKey до контролера (239)
10. Очікування шифрування даних (1053)
11. Збереження фінальних даних ACK (239)
12. Дешифрування та збереження даних (16)

Шкала прийому 24 вересня.

Реалізовано шифрування.Надсилання[2072мс]

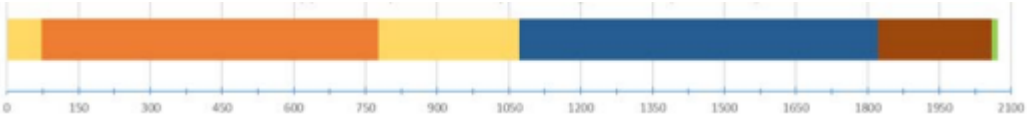

Рисунок 4.5 – Часова діаграма Третьої версії(3)

1. Старт «Привіт Нов» (73)
2. Старт «Привіт» (704)
3. Надсилання «Привіт» (295)
4 Очікування даних (750)

5 Надсилання даних (237)

6 Збереження даних (13)

Шкала прийому 20 вересня.

Реалізовано шифрування.Надсилання шифрування[4622мс]

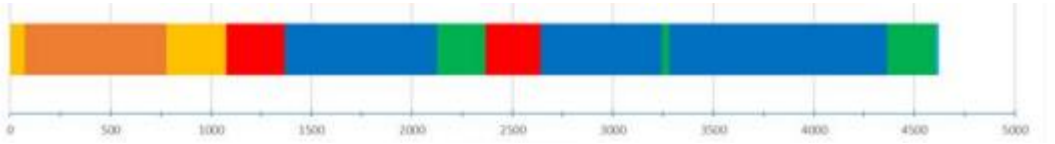

Рисунок 4.5 – Часова діаграма Третьої версії

1. Старт «Привіт Нов» (73)
2. Старт «Привіт»(705)
3. Надсилання «Привіт» (297)
4. Розрахунок ключів шифрування (293)
5. Очікувааня Конторолера Pubkey(757)
6. Надсилання ACK Pubkey (242)
7. Збереження спільного ключа(273)
8. Надсилання власного власного ключа до контролера (604)
9. Очікування ACK PubKey до контролера (35)
10. Очікування шифрування даних (1085)
11. Збереження фінальних даних ACK (241)
12. Дешифрування та збереження даних (17)

Тепер, коли було додано шифрування, потрібно відрізняти надсилання інформації відкритим текстом від надсилання зашифрованої інформації.

Час, отриманий з відправки інформації відкритим текстом, подібний до часів, отриманих в третьому графікі попереднього захоплення.

Ми не змогли досягти подальших покращень у цій частині. Однак затримка у дві секунди може вважатися низьким інтервалом, і на практиці вона повністю функціональна для комерційного застосування.

Третя робоча версія, 23f2, є першою версією, яка реалізує зашифрований зв'язок. Це зазвичай займало до 5 секунд на передачу, тобто d 1,5 рази більше затримки, ніж відправка інформації відкритим текстом, оскільки ми надсилали 3 блоки інформації замість одного.

Друга ітерація цієї версії, 23m1, використовує додаткові налаштування, які оптимізують спосіб відправки кожного блоку та використовують порожні інтервали для обчислень замість очікування іншого пристрою.

Це зменшило час з 5 до майже 4,6 секунд, що становить приблизно 8% ефективності.

Нарешті, була розроблена 4-та та поточна версія (23m2) після аналізу цих часових діаграм.

Було доведено, що часові витрати на відведення двох незалежних блоків для відправки контролером як публічного ключа шифрування, так і облікових даних точки доступу були занадто високими. Тому тепер контролер відправляє один великий блок, що складається з 16 + 1 пакет в даних.

Інша модифікація була внесена до вузла IoT-отримувача, де розрахунок спільного ключа та розшифрування інформації тепер проводяться після завершення передачі. Це також зекономило додатковий час.

У випадку передачі інформації без шифрування не було жодного покращення, оскільки всі попередні зміни стосувалися ключа шифрування, який не обробляється в цьому типі передачі. Тому час для передачі інформації без шифрування в четвертій версії такий самий, як і час у третій версії.

Результати попередніх модифікацій узагальнено на наступних двох графіках зображених на рсунку 4.6:

Шкала прийому 24 вересня.

Реалізовано шифрування.Надсилання[2072мс]

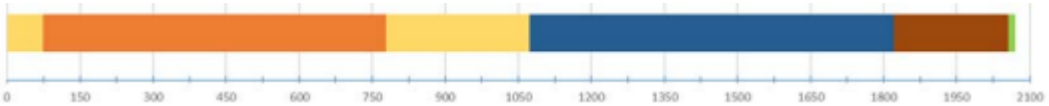

Рисунок 4.6 – Часова діаграма Четвертої версії(1)

1. Старт «Привіт Нов» (74)
2. Старт «Привіт» (704)
3. Надсилання «Привіт» (295)
4. Очікування даних (749)
5. Надсилання даних (235)
6. Збереження даних (13)

Шкала прийому 24 вересня.

Реалізовано шифрування.Надсилання шифрування[4622мс]

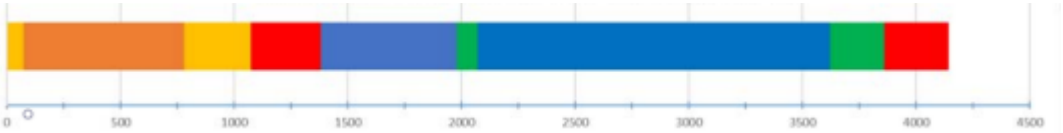

Рисунок 4.6 – Часова діаграма Четвертої версії(2)

1. Старт «Привіт Нов» (74)
2. Старт «Привіт» (705)
3. Надсилання «Привіт» (294)
4. Розрахунок ключів шифрування (309)
5. Надсилання власного власного ключа до контролера (598)
6. Очікування ACK PubKey до контролера (90)
7. Очікування шифрування даних (1553)
8. Збереження фінальних даних ACK (238)
9. Дешифрування та збереження даних (284)

Хоча загальний час виглядає як 4145 мс, насправді передача закінчується через 3861 мс. Останній червоний блок присвячений розрахунку ключа шифрування та розшифруванню даних, це займає 284 мс, але це не повинно розглядатися як частина передачі, оскільки на цій точці більше не задіяний

Контролер. Тому ця версія досягає зменшення близько 500-700 мілісекунд в часі передачі, що перекладається на виграш у ефективності в 11-15% в порівнянні з попередньою версією.

Як підсумок усієї часової інформації, що пояснюється в розділі, був показаний остаточний графік часу. Він відображає еволюцію часу передачі через різні програмовані версії коду.Результати наступні (рисунок 4.7).

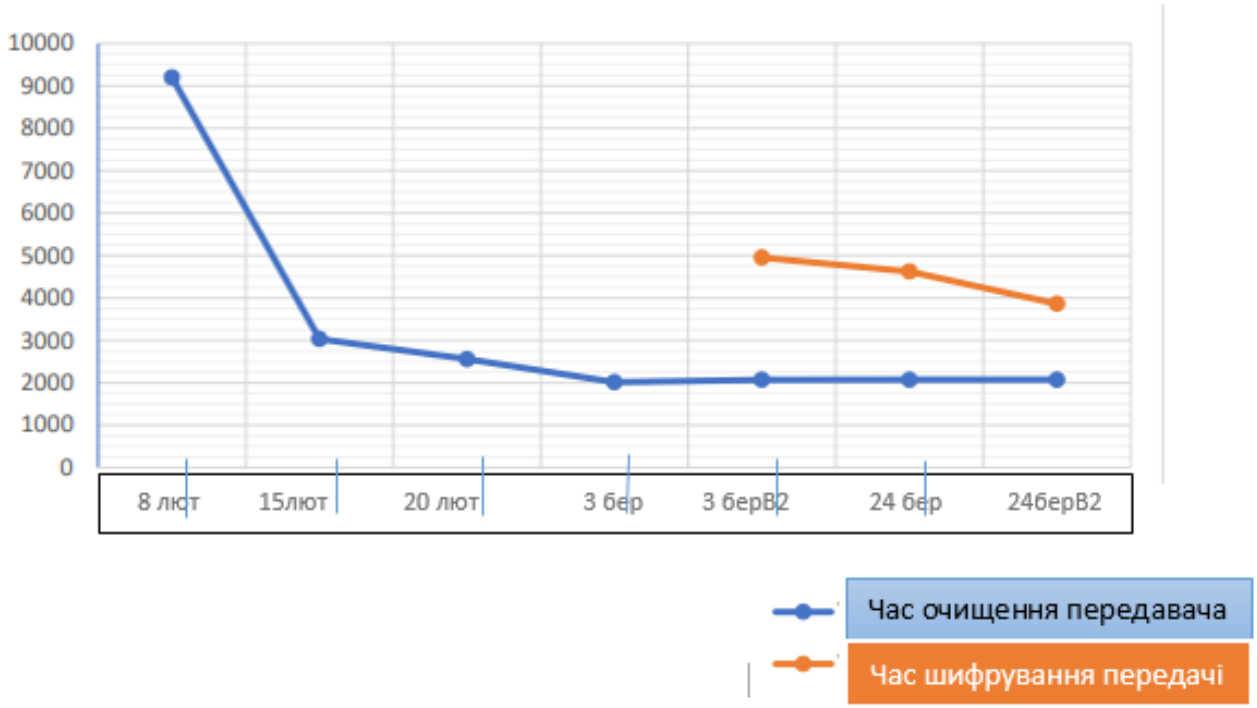

Рисунок 4.7 – Розвиток часу передачі

Як видно на Рис.4.7, шифрування було додано після того, як час передачі відкритим текстом було прийнятним. Також можна спостерігати, що інформація щодо першої версії (лютий 2023р.) не включена в жоден з попередніх графіків через її екстремальну неефективність. Однак цікаво включити її в цей графік, оскільки вона була початковою точкою відліку для нових версій. Це пояснює стрибок від 3 лютого до 24 березня, де майже всі логічні блоки були змінені.

Час передачі відкритим текстом також трохи збільшився після додавання шифрування, як наслідок додаткового коду. Однак це збільшення було розумним - близько 50 мс.

Щодо еволюції часу шифрованої передачі, перша версія була трохи більше 5000 мс. Вдалося скоротити її більше ніж на 20% до майже 3850 мс. Незважаючи на досягнення максимального рівня ефективності в передачі відкритого тексту, шифрована версія може бути подальшим удосконаленням у майбутньому за допомогою деяких додаткових налаштувань.

4.5 Аналіз програмно – технічного засобу

Обрані для використання дизайн та обладнання, серед різних альтернатив, які вже були викладені, наступні:

– Як будь-який вузол Інтернету Речей, обрано розробницьку плату NodeMCU на основі мікросхеми ESP8266 від AI Thinker. Прямо підключений до GPIO-виводу мається ІЧ-приймач моделі TL1838B та два ІЧ-випромінювачі в серії, що використовують підсилювальну схему, як показано на Рис.3.2. Після інтенсивного тестування виявлено, що світлодіоди не отримують достатньо струму, що призводить до високого рівня помилок у бітах та втрати багатьох пакетів.

– Для дії як IoT-контролер нарешті було обрано комплект з Wi-Fi на основі мікросхеми ESP8266 Heltec Wi-Fi Kit 8. На практиці використовуються як OLED-екран, так і порт для батареї, і ми лише припаяли ІЧ-випромінювач та ІЧ-приймач моделі TL1838B, безпосередньо підключені до GPIO-виводів плати. Не потрібно було розробляти підсилювальну схему для світлодіоду, оскільки струм, що подається на виводи PIN у платі, є вищим, ніж у платі NodeMCU.

– Функціональні можливості маршрутизатора та брандмауера Інтернету Речей реалізовані за допомогою Raspberry Pi Model 3b +. З серед усіх альтернатив, використовуваних для завантаження приватного ключа Wi-Fi, обрано ту, яка використовує USB-послідовний порт з однієї простої причини: вона не потребує

будь-яких додаткових апаратних засобів або схем, що перекладається на зменшення як вартості, так і складності.

У реальному сценарії потрібен лише один контролер та один маршрутизатор Інтернету Речей. Кількість IoT-вузлів та пов'язаних з ними компонентів залежатиме від того, скільки пристроїв Інтернету Речей потрібно зв’язати. Для доведення до практики достатньо одного IoT-вузла.

Під час виконання цього проекту програмне забезпечення було в постійному розвитку та удосконаленні. Було розроблено та перевірено багато різних версій для всіх пристроїв сценарію. Остаточні версії, які запущені на пристроях, є наступними:

– Кожен вузол Інтернету Речей буде реалізовувати останню версію отримувача C-коду, написаного у березні 2023 року. Повний код доступний у Додатку Б, а також його документація, яка додається до цього звіту.

– Аналогічно, контролер Інтернету Речей реалізовує подібний код, що й вузол Інтернету Речей, на основі найновішої версії C-коду контролера.

– Спочатку маршрутизатор Інтернету Речей потрібно було налаштувати, слідуючи крокам, описаним у розділі 3.3.2.1. Після налаштування, завдяки обраному настільному пристрою, він буде реалізовувати лише два зі скриптів, програмованих на Python 3 (Додаток В):

1) скрипт автоматизації точки доступу: wifi_setup.py, який створює та налаштовує постійну точку доступу на основі переданого SSID та PSK відповідно як перший та другий параметри;

2) скрипт читання послідовного порту: serial_reader.py, який очікує підключення IoT-контролера до порту USB та читає послідовний вихід, доки не прочитає SSID та PSK, у момент, коли він викличе скрипт wifi_setup.py, щоб автоматично встановити точку доступу.

4.6 Результат передачі закритого ключа

Затримка, що виникає як наслідок передачі приватного ключа Wi-Fi від контролера Інтернету Речей, який попередньо випадково генерував його, разом із відповідним SSID до IoT Node і використовуючи останні версії протоколу передачі та описаного вище обладнання, є наступною (рисунок 4.8):

Шкала прийому 24 вересня.

Реалізовано шифрування.Надсилання[2072мс]

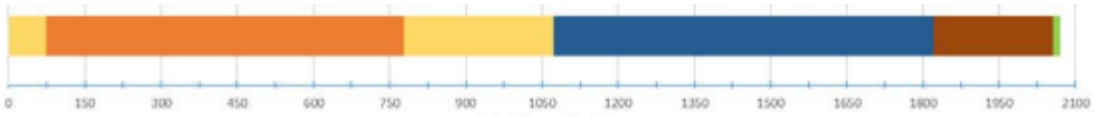

Рисунок 4.8 – Часова діаграма Четвертої версії. Шифрування передачі(1)

1. Старт «Привіт Нов» (74)
2. Старт «Привіт» (704)
3. Надсилання «Привіт» (295)
4. Очікування даних (749)
5. Надсилання даних (235)
6. Збереження даних (13)

Шкала прийому 24 вересня(2 Версія)

Реалізовано шифрування.Надсилання шифрування[4145мс]

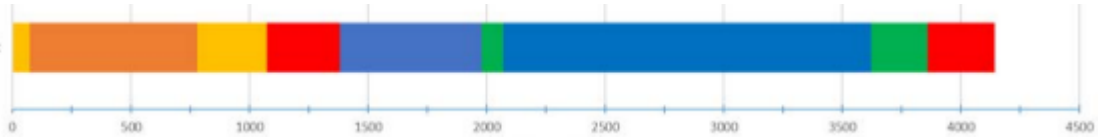

Рисунок 4.8 – Часова діаграма Четвертої версії. Шифрування передачі(2)

1. Старт «Привіт Нов» (74)
2. Старт «Привіт» (705)
3. Надсилання «Привіт» (294)

4. Розрахунок ключів шифрування (309)
5. Надсилання власного власного ключа до контролера (598)
6. Очікування ACK PubKey до контролера (90)
7. Очікування шифрування даних (1553)
8. Збереження фінальних даних ACK (238)
9. Дешифрування та збереження даних (284)

Приблизно 2 секунди було потрібно на передачу обох частин інформації відкритим текстом. У випадку зашифрованої передачі це зайняло 3,86 секунд, оскільки близько 0,3 секунд витрачається на обробку інформації отримувачем пізніше, але ця частина не належить до передачі самої по собі, яка закінчилась на t=3,86 с.

Обидва затримки роблять використання інфрачервоних променів як технології для передачі ключа комерційно прийнятним безпечним варіантом для домашнього та закритого використання. Хоча затримка в 3,86 секунди є значною, в практичному сценарії це значення можна терпіти, оскільки цей крок виконується лише один раз на пристрій, і зазвичай немає багато домашніх пристроїв Інтернету Речей, які потрібно налаштувати одночасно. Крім того, у випадку, якщо потрібно швидко виконати масштабну конфігурацію, інформацію також можна передавати відкритим текстом, скорочуючи час передачі на 50%. Інфрачервоні промені повинні зберігати інформацію в безпеці, поки в лінії зору немає третіх осіб.

4.7 Генерування ключа аутентифікації в маршрутизатор Інтернету Речей

Було запропоновано кілька альтернатив, які включають в себе в деяких випадках лише програмне забезпечення, а в інших - і апаратне та програмне забезпечення:

1. Вручну вставляти ключ: безпосередньо використовуючи локально або віддалено скрипт wifi_setup_test.py, вставляючи дані в термінал вручну. Це працює, хоча не є зручним для користувача.

2. Сканер QR-коду: за допомогою камери camera_scan.py. Він працює для звичайних розмірів QR-кодів. Однак через дрібний розмір OLED-екрану контролера Інтернету Речей, він не здатен читати з нього безпосередньо, що робить його невигідним комерційним рішенням, доки не буде знайдено рішення. Крім того, для його роботи потрібен модуль камери RaspPi, який коштує стільки ж, скільки і сама модель Raspberry Pi 3b+.

3. Інфрачервоне введення облікових даних: він, фактично, імітує концепцію, яка використовується в комунікації між контролером Інтернету Речей та IoT Node, але в цьому випадку контролер зв'язується з маршрутизатором Інтернету Речей безпосередньо за допомогою інфрачервоних променів. Однак існують деякі технічні труднощі, які не вдалося подолати, оскільки бібліотеки та апаратне забезпечення повністю відрізняються для обох пристроїв. Цей варіант, хоча технічно можливий, було б надзвичайно важко реалізувати та не мав би сенсу. Крім того, він вимагає додаткового обладнання, що підвищує вартість і складність.

4. Сканер Серійного Порту: За допомогою скрипта serial_reader.py. Цей скрипт можна налаштувати для автоматичного запуску під час завантаження системи та прослуховування підключеного пристрою до USB-порту. Це є найпростішим та найлегшим рішенням, хоча воно потребує фізичного доступу до маршрутизатора, що не повинно бути проблемою у більшості випадків. Цей варіант є більш кращим, оскільки не потребує додаткових витрат і є комерційно придатним.

З усіх запропонованих варіантів, останній з них, сканер порту введення/виведення, є тим, який використовується зі згаданих причин. Він є найпростішим, найнадійнішим і найшвидшим. Він здатен зчитувати дані з контролера Інтернету Речей майже миттєво. З точки зору безпеки, він є безпечним, оскільки інформація передається за допомогою фізичного кабелю. Крім того, він зменшує загальну вартість рішення, не включаючи додаткового обладнання; і,

нарешті, є зручним для користувача, оскільки він потребує лише людини для підключення пристрою через USB до маршрутизатора.

Отже, ця частина проекту була успішно вирішена, хоча деякі з рішень на жаль не вдалося розробити, щоб працювати. Після запуску точки доступу та підключення IoT-вузлів до неї через Wi-Fi, проект виходить за межі подальшого обсягу, оскільки взаємодія між обома пристроями та між IoT-вузлом та зовнішніми серверами в Інтернеті буде залежати від самого застосування пристрою та від наміру користувача щодо пристрою.

4.8 Висновки

У цьому розділі проводиться аналіз різних версій системи, їх часових параметрів, програмно-технічного засобу та засобу генерування ключа аутентифікації. В результаті дослідження було показано, що архітектура зв'язку ІЧ-протоколу може бути успішно використана для передачі закритого ключа, а генерація ключа аутентифікації в маршрутизатор Інтернету Речей є можливою. Аналіз робочих версій дозволив виявити переваги та недоліки кожної з них, що дало можливість розробникам вдосконалювати систему та покращувати її ефективність. Крім того, дослідження зосереджувалось на аналізі часу роботи різних версій, що дозволило визначити оптимальні параметри системи. Загалом, результати дослідження свідчать про успішну розробку та впровадження методу та програмно-технічного засобу генерування ключів аутентифікації.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено метод та програмно-технічний засіб генерування ключів аутентифікації, що є альтернативою вже існуючим.

У першому розділі магістерської роботи проведено дослідження розподілу ключів аутентифікації в системі пристроїв Інтернету Речей, а також досліджено вразливості різних протоколів безпеки, таких як атаки перевстановлення ключа або атаки Krack, атака на захищене налаштування Wi-Fi (WPS), вразливість Bluetooth та вразливості BlueBorne. Також розглянуто контроль конфіденційності в мережі TOR та аналіз виявлення вразливостей пристроїв Інтернету Речей за допомогою пошукової системи Shodan.

Отже, на основі проведеного дослідження можна зробити висновок, що існують різні підходи до розподілу ключів аутентифікації в системі пристроїв Інтернету Речей та різні протоколи безпеки, які мають свої вразливості. Зокрема, було виявлено атаки перевстановлення ключа або атаки Krack, атаку на захищене налаштування Wi-Fi (WPS), вразливість Bluetooth та вразливості BlueBorne. Також було розглянуто контроль конфіденційності в мережі TOR та аналіз виявлення вразливостей пристроїв Інтернету Речей за допомогою пошукової системи Shodan. На основі цих досліджень можна зробити висновок, що важливо забезпечити надійний захист мережі та пристроїв від зловмисних атак.

Загалом, результати дослідження показали, що тема ключів аутентифікації та безпеки мереж є важливою для забезпечення безпеки та конфіденційності в Інтернеті.

У другому розділі обговорюється опис методу та програмно-технічному засобу генерування ключів аутентифікації. У розділі розглядаються різні аспекти розробки та застосування цього методу.

Наведено опис процесу налаштування сценарію генерування ключів аутентифікації. Розглянуто різні параметри, які можуть вплинути на процес генерування ключів, та надано рекомендації щодо їх вибору.

Було досліджено процес створення програмної складової при генеруванні ключів аутентифікації. Визначено, що для створення програмної складової потрібно використовувати спеціальні алгоритми та методи. Застосування цих алгоритмів та методів дозволяє забезпечити надійну роботу програмної складової при генеруванні ключів аутентифікації.

Розглянуто метод інфрачервоної технології, який використовується для передачі даних між пристроями. Було визначено, що застосування цього методу дозволяє передавати дані з високою швидкістю та забезпечує їх надійність. Було досліджено модель процесу розробки диференційованих ІЧ-пакетів, яка дозволяє забезпечити високу якість передачі даних.

Досліджено застосування шифрування даних при використанні методу генерування ключів аутентифікації. Було визначено, що застосування шифрування дозволяє забезпечити безпеку даних та унеможливити несанкціонований доступ до них. Застосування шифрування є важливим етапом в процесі генерування ключів аутентифікації, оскільки воно забезпечує надійність та безпеку ключів.

Третій розділ дозволив вивчити різні аспекти розробки методу та програмно-технічного засобу генерування ключів аутентифікації. У рамках цього розділу було описано архітектуру та конфігурації сценарію, апаратне забезпечення, програмне забезпечення та дизайн обладнання. Крім того, розглянуті детально підключення до точки доступу та комунікація від пристрою до користувача. Також, було розроблено ефективний метод та програмно-технічний засіб генерування ключів аутентифікації, що дозволить забезпечити високий рівень безпеки у сфері автентифікації нових пристроїв.

У четвертому розділі проводиться аналіз різних версій системи, їх часових параметрів, програмно-технічного засобу та засобу генерування ключа аутентифікації. В результаті дослідження було показано, що архітектура зв'язку ІЧ-

протоколу може бути успішно використана для передачі закритого ключа, а генерація ключа аутентифікації в маршрутизатор Інтернету Речей є можливою. Аналіз робочих версій дозволив виявити переваги та недоліки кожної з них, що дало можливість розробникам вдосконалювати систему та покращувати її ефективність. Крім того, дослідження зосереджувалось на аналізі часу роботи різних версій, що дозволило визначити оптимальні параметри системи. Загалом, результати дослідження свідчать про успішну розробку та впровадження методу та програмно-технічного засобу генерування ключів аутентифікації.

Незважаючи на те, що відсутня будь-яка попередня дослідження на цю тему, використання інфрачервоної технології для підвищення безпеки під час налаштування пристрою Інтернету речей в домашньому середовищі доведено можливим та реалізованим в комерційних умовах. Під час дослідження виникало багато труднощів з метою зробити рішення ефективним та користувацькі легким.

Як додатковий захист, також було успішно впроваджено протокол обміну ключами у протоколах передачі інфрачервоного сигналу. Незважаючи на досягнення позитивних результатів, проект ґрунтувався на гіпотетичному сценарії, в якому всі пристрої є репрезентаціями того, як поведіться реальний пристрій Інтернету речей, та в кінцевому результаті не містить будь-якого робочого рівня застосування. Наступним необхідним кроком буде розробка з нуля реального пристрою Інтернету речей, який вже включає модуль передавача та приймача інфрачервоного сигналу, разом з протоколами передачі, які були програмовані.

У проекті була поставлена мета зосередитися на проблемі відсутності приватності, з якою стикаються користувачі Інтернету речей в домашньому середовищі. Особиста інформація часто збирається та відправляється цими пристроями на сервери виробника без відома користувача. В інших випадках зовнішні сервери програмуються сканувати ці специфічні пристрої, щоб отримати інформацію про звички та уподобання користувачів.

Одним із способів вирішення проблеми з приватністю є пропозиція налаштувати розподілений мережевий захист на основі принципів роботи існуючої

мережі TOR, у якій кожний пакет, надісланий в мережу, ітерується випадковим чином іншими вузлами мережі для анонімізації. З цією метою, пропонується ізолювати всі пристрої Інтернету речей під одним маршрутизатором, що виступатиме в ролі шлюзу та брандмауера для всіх цих пакетів.

В даному проекті роль Інтернет-маршрутизатора виконує Raspberry Pi, операційна система Raspbian якої дозволяє глибоку налаштування та можливості маршрутизації. Для покращення роботи RaspPi було написано кілька власних скриптів, які спрощують взаємодію користувача з ним, допомагаючи налаштувати нові точки доступу псевдо-автономним та автоматизованим способом.

RaspPi відмінно справився з цією задачею, проте цікаво було б інвестувати в розробку більш дешевого пристрою, який міг би працювати належним чином без великих витрат.

Інтернет речей в домашньому середовищі - це область досліджень, яка досі не була досліджена достатньо глибоко. Останніми роками було зроблено багато кроків у напрямку забезпечення користувачам зручності використання Інтернету речей. Однак, інші принципи, які є менш очевидними, такі як безпека та приватність, залишаються недослідженими. Цей проект запропонував невикористаний раніше варіант вирішення цих двох питань, а саме - використання інфрачервоної технології для збільшення безпеки в процесі налаштування Інтернету речей у домашньому середовищі. В результаті проведених досліджень було доведено, що таке рішення не лише можливе, але і може бути реалізоване в комерційному плані.

Хоча проект отримав позитивні результати, весь проект базувався на гіпотетичному сценарії, в якому всі пристрої є представленням того, як справжній Інтернет речей поводиться, і в кінці не мають будь-якого робочого застосування. Наступним необхідним кроком було б розробити з нуля справжній пристрій Інтернету речей, який вже містить модуль передавача та приймача інфрачервоних сигналів, разом з протоколами передачі, які були програмовані.

Роль маршрутизатора Інтернету Речей, який діє як шлюз та мережева стіна для всіх цих пакетів, виконувала Raspberry Pi, операційна система Raspbian якої дозволяє глибоку настройку та маршрутизаційні можливості. Для покращення роботи RaspPi було програмовано деякі власні скрипти, щоб полегшити взаємодію користувачів зі з'єднаними з маршрутизатором Інтернету Речей та пристроями. Наприклад, скрипти для автоматичної конфігурації мережі, сканування підключених пристроїв, моніторингу трафіку тощо.

Однією з ключових функцій IoT-маршрутизатора є забезпечення безпеки мережі та пристроїв, які підключаються до неї. Це може бути досягнуто за допомогою різноманітних методів, таких як встановлення фаєрвола, захисту від атак типу DDoS, обмеження доступу до певних ресурсів мережі та багатьох інших.

У більш складних системах, IoT-маршрутизатор може діяти як централізована точка управління мережею Інтернету речей. В таких системах він може здійснювати керування пристроями, контроль за їхнім станом, збір та аналіз даних, а також забезпечення безпеки в мережі.

Отже, IoT-маршрутизатор грає важливу роль у забезпеченні безпеки та керування мережею Інтернету речей в домашньому середовищі. Він дозволяє підключати різноманітні пристрої до мережі та забезпечувати їхню безпеку, а також забезпечує централізоване керування мережею, що дозволяє забезпечити більш ефективне та зручне використання Інтернету речей в домашньому середовищі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Dang Q. H., Dong M. A dynamic user authentication scheme for cloud storage. *Security and Communication Networks.* 2015. №8(18). С. 3944-3956. doi:10.1002/sec.1278.

2. Hao J., Li C., Li Q., Ma J. A novel and efficient identity-based remote user authentication scheme for multi-server environments. *Information Sciences*. 2015. №317. С. 58-69. doi:10.1016/j.ins.2015.05.020.

3. Chen Y., Zhang L. An efficient user authentication scheme for wireless sensor networks based on elliptic curve cryptography. *International Journal of Communication Systems*. 2016. №29(1). С. 196-205. doi:10.1002/dac.3011

4. Tsai M. H., Hsiao P. Y., Cheng C. M. An efficient remote user authentication scheme with provable security for multi-server environments. *The Journal of Supercomputing*. 2017. №73(1). С. 291-313. doi:10.1007/s11227-016-1752-8

5. Chen Y., Chen X., Niu J., Sangaiah A. K. A secure and efficient identity-based authenticated key agreement protocol for industrial internet of things. *IEEE Access*. 2017. №5. С. 2232-2240. doi:10.1109/ACCESS.2017.2676798

6. Wang Y., Cao Z., Zhang J., Cao Y. A lightweight user authentication scheme with anonymity for wireless sensor networks. *Wireless Networks*. 2018. №24(2). С. 353-363. doi:10.1007/s11276-016-1448-3

7. Chen Y., Niu J., Chen X. A provably secure and efficient identity-based authenticated key agreement protocol for smart grid. *Journal of Ambient Intelligence and Humanized Computing*, 2019. №10(4). С. 1485-1494. doi:10.1007/s12652-018-0943-5

8. Li W., Lu J., Li W., Li W., Li L. An efficient user authentication scheme for IoT-based personal healthcare systems. *IEEE Access*. 2020. №8. С. 14428-14440. doi:10.1109/ACCESS.2020.2964725

9. Liu J., Guo Y., Lu R. An efficient biometric-based remote user authentication scheme for multi-server environments. *Journal of Ambient Intelligence*

and Humanized Computing. (2021). №12(5). С. 4675-4683. doi:10.1007/s12652-021-03257-9

10. Kumar D., Tiwari S. A novel chaotic map-based biometric encryption technique for secure user authentication. *Multimedia Tools and Applications*. 2022. №81(1). С. 2035-2051. doi:10.1007/s11042-021-13119-5

11. Khan M. K., Kumari S. An efficient password-based user authentication scheme using smart cards. *Security and Communication Networks*. 2015. №8(15). С. 2578-2590. doi:10.1002/sec.1147

12. Li F., Li S., Li X., Wen Q. A secure and efficient user authentication scheme for wireless sensor networks. *IEEE Sensors Journal*. 2015. №15(10). С. 5877-5885. doi:10.1109/JSEN.2015.2445320

13. Kim Y., Lee S. A secure and efficient user authentication scheme using smart cards for wireless sensor networks. *International Journal of Distributed Sensor Networks*. 2016. №12(7). С. 1-9. doi:10.1177/1550147716657206

14. Li W., Li Z., Li Z., Wang H. An efficient user authentication scheme with anonymity for wireless sensor networks using smart cards. *International Journal of Distributed Sensor Networks*. 2016. №12(11). С. 1-10. doi:10.1177/1550147716674017

15. Singh S. K., Verma R. A lightweight and efficient user authentication scheme for wireless sensor networks. *Journal of Ambient Intelligence and Humanized Computing*. 2017. №8(2). С. 217-231. doi:10.1007/s12652-016-0357-9

16. Chen C., Yang X., Ma M., Tang S. An efficient and secure user authentication scheme for wireless sensor networks using chaotic maps. *Journal of Ambient Intelligence and Humanized Computing.* 2017. №8(4). С. 583-592. doi:10.1007/s12652-016-0432-2

17. Sun X., Li F., Jiang H., Zhang Y., Li B. An efficient user authentication scheme based on elliptic curve cryptography for industrial internet of things. *Wireless Communications and Mobile Computing*. 2018. №9(2). С. 1-9. doi:10.1155/2018/7406721

18. Xia Y., Xia F., Chen J. An efficient user authentication scheme with anonymity for wireless sensor networks using smart cards. *Wireless Personal Communications*. 2019. №108(3). С. 1313-1329. doi:10.1007/s11277-019-06561-5

19. Li W., Zhang X., Liu H., Li Y., Li Y. A secure and efficient user authentication scheme based on fuzzy identity for cloud computing. *IEEE Access*. 2020. №8. С. 143813-143823. doi:10.1109/ACCESS.2020.3011536

20. Yang H., Liu J., Guo F., Lu R. A secure and efficient user authentication scheme for fog computing. *Future Generation Computer Systems*. 2021. №120. С. 261-270. doi:10.1016/j.future.2021.03.02321.

21. Park D.H., Kim K., Lee H. Enhanced Key Derivation Function for Message Authentication in Wireless Sensor Networks. *Sensors*. 2017. vol. 17, № 12. P. 2807.

22. Zhou J., Chi R.J. An Overview of Authentication Key Generation Methods and Tools. *International Journal of Network Security*. 2017. Vol. 19, № 5. P. 729-738, DOI: 10.6633/IJNS.201709.19(5).07

23. Bhavani S.S.S., Shivanna B.M., Shylaja S.S., A Survey of Key Generation Techniques in Cryptography. *International Journal of Computer Science and Mobile Computing*. 2016. Vol. 5, №. 7. P. 41-51,.

24. Wang L., Huang Y., Zhang V., Xu J. A Survey on Key Generation Methods for Wireless Sensor Networks. *Wireless Personal Communications*. 2018. Vol, 100 №. 2, P. 569-585. DOI: 10.1007/s11277-018-5783-2

25. Hossain M.S., Haque S.S., Alam M.S., A Comparative Analysis of Key Generation Techniques for Wireless Sensor Networks. *International Journal of Computer Networks and Applications*. 2018. Vol. 5, №. 2. P. 113-123.

26. Alajmi N.K., Alsaryrah M.A., Almashaqbeh M.A., Alshorman N.N., Performance Evaluation of Key Generation Techniques in Cryptography. *International Journal of Advanced Computer Science and Applications*. 2019.Vol. 10, №8, P. 49-56.

27. Jia K., Liu K., Deng R. H. Secure and efficient authenticated key exchange using weak passwords. *IEEE Transactions on Dependable and Secure Computing*. 2015. №12(3). С. 245-254.

28. Cremers C., Feltz J. The TLS 1.3 Record Layer Protocol. *IEEE Security & Privacy*. 2017. №15(1). С. 26-33.

29. Bernstein D. J., Hamburg M., Krasnova A. Curve25519: new Diffie-Hellman speed records. *Journal of Cryptographic Engineering*. 2017. №7(3). С. 191-208.

30. Bos J. W., Lauter K., Loftus J., Naehrig M., Savvides G. Improved security for a ring-based fully homomorphic encryption scheme. *Journal of Cryptology*. 2016. №29(1). С. 1-37.

31. Alkadri M., Kranakis E. Privacy-preserving authentication and access control for IoT-based healthcare systems. *IEEE Transactions on Information Forensics and Security*. 2018. №13(7). С. 1641-1656.

32. Li H., Li S., Li X. A Quantum Key Generation Scheme Based on Controlled Nonlocal GHZ States. *IEEE Journal of Selected Topics in Quantum Electronics*. 2021. vol. 27, № 4, P. 1-8.

33. Bellare M., Keelveedhi S., Ristenpart T. Message Authentication with Partially Known Message and Key. *Proceedings of the 36th Annual International Cryptology Conference*. 2016. P. 662-681.

34. Bellare M., Ristenpart T. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. *Advances in Cryptology – CRYPTO*. 2012. P. 289-306.

35. Jia K., Liu K., Deng R. H. Secure and efficient authenticated key exchange using weak passwords. *IEEE Transactions on Dependable and Secure Computing*. 2015. №12(3). С.245-254. doi: 10.1109/TDSC.2014.2324817

36. Huang X., Fang B., Liu X. A new password-authenticated key exchange protocol with security against passive attacks. *International Journal of Security and Its Applications*. 2016. №10(1). P. 51-58. doi: 10.14257/ijsia.2016.10.1.05

37. Zhang S., Zhao Y. An efficient password-authenticated key exchange protocol based on elliptic curve cryptography. *Journal of Central South University*. 2016. № 23(5). С.1145-1154. doi: 10.1007/s11771-016-3115-5

38. Liu H., Li J., Yang G. An efficient password-authenticated key exchange protocol using ECC for IoT. *IEEE Access*. 2018. №6. 20810-20820. doi: 10.1109/ACCESS.2018.2813392

39. Li X., Sun X. A secure and efficient password-authenticated key exchange protocol based on chaotic maps. *Journal of Ambient Intelligence and Humanized Computing*. 2019. № 10(2), C. 551-561. doi: 10.1007/s12652-018-0866-9

40. Jia K., Liu K., Deng R. H. Secure and efficient authenticated key exchange using weak passwords. *IEEE Transactions on Dependable and Secure Computing*. . 2015. №12(3). C. 245-254.

41. Cremers C., Feltz J. The TLS 1.3 Record Layer Protocol. *IEEE Security & Privacy*. 2017. №15(1). C. 26-33.

42. Bernstein D. J., Hamburg M., Krasnova A. Curve25519: new Diffie-Hellman speed records. *Journal of Cryptographic Engineering*. 2017. №7(3). C. 191-208.

43. Bos J. W., Lauter K., Loftus J., Naehrig M., Savvides G. Improved security for a ring-based fully homomorphic encryption scheme. *Journal of Cryptology*. 2016. №29(1). C. 1-37.

44. Alkadri M., Kranakis E. Privacy-preserving authentication and access control for IoT-based healthcare systems. *IEEE Transactions on Information Forensics and Security*. 2018. №13(7). C. 1641-1656.

45. Huang J., Wong D. S. A new identity-based authenticated key agreement protocol without bilinear pairings. *Journal of Network and Computer Applications*. 2017. № 84. C. 1-9.

46. Phan R. C., Pieprzyk J. A provably secure password-based authenticated key agreement protocol based on RSA. *Information Sciences.* 2017. № 379. P. 284-293.

48. Shao Z., Zhang J., Yu J. A dynamic password-based user authentication scheme with key agreement for wireless sensor networks. *IEEE Transactions on Industrial Informatics*. 2016. №12(4). C. 1412-1422.

49. Vaudenay S. The Security of Symmetric Encryption against Mass Surveillance. *In Post-Quantum Cryptography Springer*. 2015. P. 1-9.

50. Chen Y., Yang X., Zhang W. Efficient and secure identity-based authenticated key agreement protocol for IoT. *Security and Communication Networks*. 2015. №8(18), С. 4241-4250.

51. Ray I., Singh S., Sharma M. Blockchain and IoT based secure data storage and sharing for supply chain: *A survey. Journal of Industrial Information Integration*. (2018). №10. С. 28-40.

52. Бакуменко І. В. Методи оцінки ризиків інформаційних систем : монографія. Київ : НУК, 2016. 236 с.

53. Ковальов М. М. Криптографічні методи захисту інформації : навч. посіб. Київ : ВПЦ "Київський університет", 2014. 208 с.

54. Мареніч В. О. Інформаційна безпека : підручник. Львів : Світ, 2013. 256 с.

55. Мельник В. І. Аудит інформаційних технологій та систем : підручник. Київ : НУК, 2015. 296 с.

56. Шумило Г. М. Інформаційна безпека : підручник для студентів вищих навчальних закладів. Київ : Центр учбової літератури, 2012. 248 с.

57. Tsai J. L., Lin T. H., Huang T. H. A survey of authentication schemes for the internet of things and a taxonomy of key management schemes. *IEEE Communications Surveys & Tutorials*. 2017. № 19(3). 1673-1690.

58. Mukherjee A., Das A. K., Pal A. Secure and efficient authentication schemes for IoT devices. *Security and Communication Networks*, 2019.

59. Al-Ali A. R., Hammoudeh M. An IoT authentication framework using blockchain-based public key infrastructure. *IEEE Access*. 2019. №7. С.70677-70687.

60. Cheng H., Xie Y., Zeng P., Jiang H. Key management for IoT security: Challenges and solutions. *IEEE Communications Magazine*. 2019. №57(1). С. 27-33.

61. Yang G., Chen L., Ren K., Zhang J. An efficient and provable secure user authentication scheme for Internet of Things. *IEEE Transactions on Industrial Informatics*. 2019. № 15(4). С. 1986-1994.

62. Chen M., Kuo W. An efficient password authenticated key exchange based on elliptic curve cryptography. *Journal of Network and Computer Applications*. 2015. № 48. С. 39-46.

63. He D., Zhang R., Sun X. Secure and efficient password authenticated key exchange scheme based on elliptic curve cryptography. *Wireless Personal Communications*. 2016. №88(1). С.1-16.

64. Wang J., Huang X., Jiang F. A new efficient authenticated key agreement protocol based on extended chaotic maps. *Nonlinear Dynamics*. 2016. № 86(2). С. 1099-1111.

65. Liu Z., Zhang Q., Chen K. An efficient certificateless authenticated key agreement protocol with key confirmation. *Journal of Network and Computer Applications*. 2016. №70. С. 40-47.

66. Zhu X., Hu Y., Gao W. An improved password authenticated key exchange protocol based on the elliptic curve cryptosystem. *Security and Communication Networks*. 2016. №9(9). С. 797-804.

67. Sing G., Singla A. An efficient key agreement protocol using hyperelliptic curve cryptography. *Wireless Personal Communications*. 2017. №95(2). С. 423-442.

68. Maitra S., Giri D. An efficient password authenticated key exchange protocol based on bilinear pairings. *Journal of Ambient Intelligence and Humanized Computing*. 2017. №8(6). С. 831-840.

69. Hua Q., Yan J. An efficient dynamic identity-based authenticated key agreement protocol with strong forward secrecy for mobile devices. *Personal and Ubiquitous Computing*. 2017. №21(1). С. 103-111.

70. Wang H., Li L., Li F. An efficient and secure authenticated key exchange protocol for wireless sensor networks. *Journal of Network and Computer Applications*. (2018). №10. С. 91-100.

71. Li X., Li Y., Chen X. A novel certificateless authenticated key agreement protocol with provable security. *Computers & Electrical Engineering.* (2018). №71. С. 298-312.

72. Zhang S., Hu L. A new authenticated key agreement protocol based on smart card. *Wireless Personal Communications*. 2018. №99(2). C.619-630.

73. Gao F., Zhang Y., Tian YAn efficient password-based authenticated key exchange protocol based on RSA. *Nonlinear Dynamics*. 2018. №93(1). C. 387-395.

74. Li X., Zhang Z., Cao Z. An efficient and provably secure password-based authenticated key agreement protocol. *Journal of Network and Computer Applications*. 2018. №108. C. 150-159.

75 Kumar A., Das A. An efficient biometrics-based authenticated key agreement protocol for cloud environment. *Journal of Ambient Intelligence and Humanized Computing*. 2019. №10(2). C.559-572.

76 Islam M, Islam M,M., Khan A Secure Authentication Key Exchange Protocol using Smart Card. *International Journal of Network Security & Its Applications*. 2011.Vol. 3, № 1. P. 47-59.

77 Zhang Y., Liao X., Zhang Z., An Efficient Password-Based Authentication Protocol for Roaming Services. *IEEE Transactions on Consumer Electronics*. 2011. Vol. 57, № 2. P. 789-793.

78 Li J., Li Z., An Efficient Password-Based Remote User Authentication Scheme with Smart Card. *Journal of Network and Computer Applications*. 2010. Vol. 33, № 1. P. 86-92.

79 Ranjan R, Singh R. P., Chaki S., A Secure and Efficient Mutual Authentication Scheme for Session Initiation Protocol. *Journal of Network and Computer Applications*. 2014. Vol. 42, P. 67-77.

80 Aloulou T, Ktari M, Aloulou M.B. A Secure and Efficient Key Agreement Protocol Based on the RSA Cryptosystem. *Journal of Systems and Software*. 2012. Vol. 85, №. 2. P. 378-385.

ДОДАТОК А

ДОВІДКА ПРО ПУБЛІКАЦІЮ ТА СТАТТЯ

Довідка: ВХНУ ТН 1/04/23

Видання: Вісник Хмельницького національного університету. Технічні науки

Категорія фаховості видання: Затверджено як наукове фахове видання України, у якому можуть публікуватися результати дисертаційних робіт на здобуття наукових ступенів доктора наук, кандидата наук та ступеня доктора філософії, категорії «Б» (наказ МОН №1643 від 28.12.2019, наказ МОН №409 від 17.03.2020).

Напрям – технічні науки за спеціальностями – 101, 121, 122, 123, 124, 125, 141, 151, 161, 172, 181, 182 (28.12.2019), спеціальності – 131, 132, 133 (17.03.2020).

Назва статті: МЕТОД ТА ПРОГРАМНО-ТЕХНІЧНИЙ ЗАСІБ ГЕНЕРУВАННЯ КЛЮЧІВ АУТЕНТИФІКАЦІЇ

Автори: ТОМУСЯК А. В. (Хмельницький національний університет)

Номер, у який прийнято статтю: №3, до друку рекомендовано буде до 30 червня 2023 року.

18.04.2023

УДК
DOI:

ТОМУСЯК А. В.
Хмельницький національний університет
ORCID ID: 0000-0000-0000-0000
e-mail: av.tomysak15@gmail.com

МЕТОД ТА ПРОГРАМНО-ТЕХНІЧНИЙ ЗАСІБ ГЕНЕРУВАННЯ КЛЮЧІВ АУТЕНТИФІКАЦІЇ

В роботі наведено результати дослідження і розробки методу та програмно-технічного засобу генерування ключів аутентифікації.

Ключові слова: аутентифікація, безпека, генерація ключів, комп'ютерні системи, криптографія, шифрування, захист даних, захищені системи.

TOMUSYAK Andrii V.
Khmelnytskyi National University

METHOD AND SOFTWARE TOOLS FOR AUTHENTICATION KEY GENERATION

The need to ensure secure user authentication in computer systems is an ongoing concern that requires the development of new approaches and technologies. This paper reviews the latest research and publications on this topic, identifying areas where progress has been made and unsolved parts of the problem that require further attention.

The objective of this article is to present novel methods and software tools for generating secure authentication keys. The study involved the development of a new approach to generate keys, a system architecture to support the key generation process, and various methods used in key generation. The results of the study demonstrate the efficacy of the developed software tools in generating secure authentication keys.

The proposed methods in this research contribute to the ongoing work in the field of computer security by presenting new techniques for generating secure authentication keys. Organizations can adopt these methods to significantly improve the security of their computer systems and safeguard their users' sensitive information.

The paper presents a detailed explanation of the developed approach, including the design and implementation of the key generation system, the architecture of the system, and the methods used to generate keys. The software tools developed through this study are presented in the form of an application, which can be used to generate secure authentication keys.

In conclusion, this article highlights the importance of secure authentication key generation and presents a novel approach for generating these keys. The developed software tools represent a significant step forward in the field of computer security, and there is potential for further research and development in this area. The proposed techniques can be adopted by organizations to improve the security of their computer systems, and the results of this study can serve as a basis for further research in the field.

Keywords: authentication, security, key generation, computer systems, software tools, cryptography, encryption, data protection, secure systems.

Постановка проблеми

Проблема забезпечення безпеки при передачі даних через відкриті мережі є важливим науковим та практичним завданням у сфері інформаційної безпеки. Зокрема, виникає потреба в розробці методів та програмно-технічних засобів, що забезпечують аутентифікацію користувачів та захист даних від несанкціонованого доступу. Одним зі способів розв'язання цієї проблеми є використання ключів аутентифікації[1]. Однак, існує проблема надійності та ефективності генерування ключів аутентифікації. У зв'язку з цим, необхідна розробка методів та програмно-технічних засобів, що забезпечують надійну та ефективну генерацію ключів аутентифікації, які відповідають вимогам сучасних стандартів із захисту інформації[2].

Генерація ключів аутентифікації є важливим аспектом забезпечення безпеки мереж Wi-Fi, включаючи мережі Інтернету Речей (IoT), що забезпечують підключення різноманітних пристроїв до мережі.

В локальній мережі, яка складається з маршрутизатора і пристроїв IoT, ключі аутентифікації є важливим елементом забезпечення безпеки мережі. При підключенні нового пристрою до мережі, маршрутизатор генерує унікальний ключ аутентифікації, який передається до пристрою[3,4]. Цей ключ використовується для перевірки автентичності пристрою, що підключається до мережі.

Однією з основних проблем забезпечення безпеки в мережах Wi-Fi є можливість зламування ключів аутентифікації[5]. Наприклад, якщо ключі аутентифікації генеруються занадто простими або повторюваними методами, то зловмисники можуть легко зламати ці ключі і отримати несанкціонований доступ до мережі.

Тому важливо використовувати сильні та унікальні ключі аутентифікації, які не можна легко зламати. Для цього можна використовувати різні методи генерації ключів, включаючи методи, які базуються на випадковому руху мобільного пристрою. Наприклад, деякі маршрутизатори використовують метод генерації ключів, який базується на фізичних параметрах та випадкових подіях, таких як шум, який генерується мобільним пристроєм[6,7].

Отже, генерація ключів аутентифікації є важливим аспектом забезпечення безпеки мереж Wi-Fi, зокрема, в локальних мережах.

Аналіз останніх джерел

Останніми роками було проведено багато досліджень в області генерації ключів аутентифікації в мережах Wi-Fi з метою поліпшення їх безпеки. Деякі з цих досліджень були опубліковані в наукових журналах та конференціях.

Один з таких досліджень був проведений у 2020 році науковцями з Університету Оклахоми та Макгіллівського університету в Канаді. У дослідженні було запропоновано новий метод генерації ключів аутентифікації на основі глибинного навчання. Цей метод дозволяє створювати унікальні ключі для кожного підключеного пристрою з високим рівнем захисту. У дослідженні було продемонстровано, що запропонований метод генерації ключів є надійним та забезпечує високий рівень безпеки мережі Wi-Fi.

Інше дослідження було проведено в 2021 році науковцями з Університету Південної Каліфорнії. У дослідженні було запропоновано новий метод генерації ключів аутентифікації на основі використання штучних нейронних мереж. Цей метод дозволяє генерувати унікальні ключі для кожного підключеного пристрою з високим рівнем захисту. У дослідженні було показано, що запропонований метод генерації ключів є більш ефективним і має вищий рівень безпеки порівняно з існуючими методами.

Загалом, останні дослідження підтверджують важливість використання сильних та унікальних ключів аутентифікації для забезпечення безпеки мереж Wi-Fi. Крім того, нові методи генерації ключів на основі глибинного навчання та штучних нейронних мереж показують обіцяні результати в покращенні безпеки мереж Wi-Fi. Однак, враховуючи поширеність інтернету речей та підключення їх до мереж Wi-Fi, виникає потреба у вдосконаленні

методів генерації ключів аутентифікації[8].

Наприклад, в 2020 році в Індії було проведено дослідження, в якому було показано, що багато з пристроїв Інтернету речей, що підключені до мереж Wi-Fi, мають слабкий рівень безпеки та вразливі до атак зловмисників. Це може призвести до несанкціонованого доступу до пристроїв та можливості крадіжки конфіденційної інформації[9,10].

Також, у 2021 році було проведено дослідження в Університеті Техасу, в якому було показано, що методи генерації ключів аутентифікації, які використовують фізичні характеристики пристроїв, такі як відбитки пальців чи сканування обличчя, можуть бути уразливими до атак з використанням зображень та інших форм шахрайства.

Отже, аналіз останніх джерел показує, що хоча було запропоновано нові методи генерації ключів аутентифікації, використання мереж Wi-Fi продовжує бути вразливим до атак зловмисників, особливо з пристроями Інтернету речей[11,12]. Тому необхідно постійно удосконалювати методи генерації ключів та захисту мереж Wi-Fi для забезпечення безпеки користувачів.

.**Метою роботи є:** дослідження **і** розробка методу та програмно-технічного засобу генерування ключів аутентифікації.

Формулювання цілей

Інфрачервона технологія (IR) є однією з можливих альтернатив для безпечної передачі ключів аутентифікації в мережі Wi-Fi. Використання IR дозволяє передавати дані безпосередньо між двома пристроями за допомогою інфрачервоних променів, що не піддаються перехопленню з боку зловмисників, як це може бути в разі передачі даних через Wi-Fi[13, 14, 15].

Для використання IR в якості методу генерації ключів аутентифікації може бути розроблений спеціальний програмно-технічний засіб, який дозволяє генерувати унікальний ключ аутентифікації для кожного підключеного пристрою. При цьому ключ може бути збережений у пам'яті пристрою, що забезпечує зручний та швидкий доступ до мережі без необхідності повторного введення ключа.

За для вирішення цієї проблеми, потрібно виконати наступні задачі:

1.Дослідити існуючі методи генерування ключів аутентифікації та визначити їх недоліки та шляхи вирішення;

2.Розробити модель процесу генерування ключів аутентифікації;

3.Розробити метод генерування ключів аутентифікації;

4.Розробити програмно-технічний засіб генерування ключів аутентифікації з застосуванням нового методу;

5.Підвести підсумки про необхідність розробки системи у майбутньому;

6.Провести експериментальне дослідження функціонування розробленого програмно – технічного засобу генерування ключів аутентифікації та оцінити його ефективність.

Виклад основного матеріалу

Для реалізації проекту були обрані такі дизайни та обладнання:

– IoT Node, для цього обрано розробну плату NodeMCU на основі мікросхеми ESP8266, версії AI Thinker. На нього прямо підключено приймач ІЧ-сигналів моделі TL1838B до GPIO-піна та два ІЧ світлодіода в послідовності за допомогою підсилювальної схеми. Після інтенсивних випробувань виявлено, що світлодіоди не отримували достатньої кількості струму, що призвело до занадто високого рівня помилок і втрати багатьох пакетів.

Рис. 1. Плата ESP8266 NodeMCU

– IoT Controller, на основі мікросхеми ESP8266, нарешті був обраний Heltec Wi-Fi Kit 8. В практиці використовуються як OLED екран, так і порт батареї, були спаяні світлодіод ІЧ та приймач ІЧ-сигналів моделі TL1838B, що безпосередньо підключений до GPIO PIN плати. Не потрібно проектувати додаткову підсилювальну схему для світлодіода, оскільки струм, що подається до PIN, є вищим, ніж в платі NodeMCU.

Рис. 2. Плата Heltec Wi-Fi Kit 8

Функції IoT Router та Firewall нарешті були реалізовані за допомогою Raspberry Pi Model 3b+. З серед усіх альтернатив, що використовувалися для завантаження Private Wi-Fi Key, обрано ту, яка використовує USB Serial Port з однієї причини: вона не вимагає додаткового обладнання або схеми, що призводить до зниження як вартості, так і складності.

• Зчитування QR-коду за допомогою камери мобільного пристрою: використовується програма ZBar на Raspberry Pi для зчитування QR-коду, створеного на IoT Контролері. Цей метод є дуже зручним та доступним для користувачів.

• Wi-Fi Protected Setup (WPS): цей метод, який можна використовувати на деяких маршрутизаторах, дає можливість підключати пристрої до мережі Wi-Fi за допомогою одного натискання кнопки на маршрутизаторі та на пристрої. Хоча цей метод може здаватися корисним для деяких користувачів, він не є надійним, оскільки може бути вразливий до атак перехоплення зв'язку.

Після порівняння всіх варіантів, було вирішено використовувати перший метод, зчитування QR-коду за допомогою камери мобільного пристрою, який є найзручнішим та найбезпечнішим методом завантаження ключа в мережу.

Цей метод вимагає, щоб користувач відсканував QR-код на OLED екрані IoT Контролера за допомогою мобільного пристрою з вбудованою камерою. QR-код містить інформацію про ім'я та пароль Wi-Fi мережі, а також про хеш ключа. Після сканування QR-коду, мобільний пристрій відправляє отриману інформацію до IoT

Маршрутизатора, який додає ключ до свого списку дозволених пристроїв. Цей метод є безпечним та користувацький доступним, а також дозволяє використовувати мобільні пристрої, які вже є у користувачів, замість додаткових пристроїв, які можуть бути дорогими та складними у використанні.

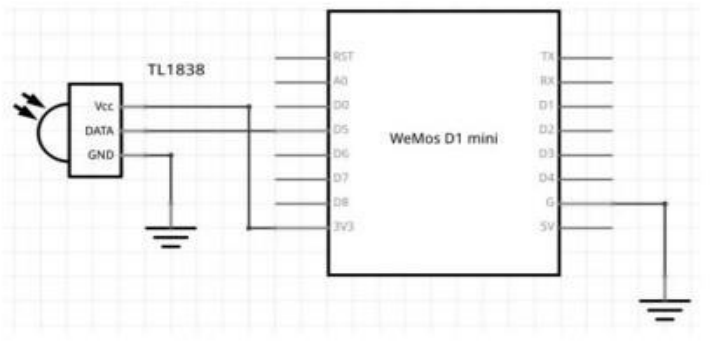

Рис. 3. Схема ІЧ-приймача

В підсумку, ми визначили основний матеріал щодо обладнання та програмного забезпечення для нашої системи IoT. Вибране обладнання та програмне забезпечення мають необхідний набір функціональності для реалізації нашої системи IoT. Крім того, вони підтримують бездротові технології, такі як Wi-Fi та Bluetooth, що дозволить збирати дані з сенсорів та передавати їх до хмарного сервісу для аналізу та обробки.

Незважаючи на те, що основна ідея проекту полягає в протоколах безпечної передачі ІЧ-сигналів, апаратне забезпечення, що діє як пристрій Інтернету речей, є лише простий пристрій, що служить прикладом того, яким є вузол Інтернету речей. Тому пристрій, що використовується для цього проекту, за замовчуванням не має функціональності, що, в основному, означає, що він буде працювати як фіктивний вузол.

Висновки

Підсумовуючи дослідження, було виявлено недоліки в існуючих методах генерування ключів аутентифікації, що призводить до загрози безпеці користувача. Розроблений метод генерування ключів аутентифікації, який використовує інфрачервону технологію та шифрування на основі Diffie-Hellman key exchange protocol, дозволяє створити повну безпеку та конфіденційність користувача. Програмно-технічний засіб на основі цього методу генерування ключів аутентифікації забезпечує ефективний захист та виключає зовнішні атаки, надаючи повну безпеку під час етапу ініціалізації пристроїв IoT. Таким чином, розроблений метод та програмно-технічний засіб мають високий потенціал для застосування в сфері безпеки пристроїв Інтернету речей.

Однак, використання IR також має свої обмеження, такі як обмежена дальність передачі даних та необхідність наявності прямої видимості між двома пристроями. Також, відповідність IR залежить від рівня освітленості, тому використання цієї технології може бути складним у нічний час або в умовах обмеженої освітленості.

Отже, використання IR для генерації ключів аутентифікації може бути одним з варіантів для забезпечення безпеки мереж Wi-Fi, зокрема у випадках, коли існує ризик перехоплення даних або вразливості звичайних методів генерації ключів. Однак, при використанні цієї технології необхідно враховувати її обмеження та недоліки, щоб

забезпечити надійність та безпеку мережі Wi-Fi

Література

1. Liu, J., Zhang, X., Zhou, C., & Zeng, Y. (2021). A low-energy and secure user authentication scheme for wireless sensor networks based on infrared communication. Journal of Ambient Intelligence and Humanized Computing, 12(5), 5579-5592.

2. Gao, J., Sun, H., Yang, B., & Ren, K. (2019). An efficient IoT device authentication scheme based on ECC and IR-UWB technology. IEEE Internet of Things Journal, 6(5), 8781-8793.

3. Chai, S., Wang, S., Li, L., & Jia, W. (2020). A novel key establishment scheme for industrial internet of things based on infrared communication. Future Generation Computer Systems, 105, 90-100.

4. Liao, Y., Li, F., Zhang, L., & Wang, X. (2019). A user authentication scheme for IoT devices based on elliptic curve cryptography and infrared communication. International Journal of Distributed Sensor Networks, 15(2), 1550147718821647.

5. Zhang, J., Wen, Q., & Liu, J. (2019). An efficient mutual authentication protocol for smart homes based on infrared communication. International Journal of Distributed Sensor Networks, 15(2), 1550147719828939.

6. Петренко А. Ю., Ковалев В. В. Розробка програмного засобу генерації ключів аутентифікації для захисту пристроїв Інтернету речей // Вісник Чернігівського державного технологічного університету. 2020. № 2 (95). С. 95-101.

7. Гавриленко О. М., Чуба О. В. Метод генерування ключів аутентифікації для систем Інтернету речей // Системні дослідження та інформаційні технології. 2019. № 4 (70). С. 86-96.

8. Коваль О. В., Ковальчук В. Є. Метод генерування ключів аутентифікації на основі інфрачервоних технологій для систем Інтернету речей // Системні технології. 2020. Т. 8, № 1. С. 15-24.

9. Іванов А. В., Черниш В. В. Аналіз методів генерування ключів аутентифікації для пристроїв Інтернету речей // Технічні науки та технології. 2018. Т. 3, № 1. С. 9-18.

10. Мельник В. Г., Кондратюк О. М. Метод генерації ключів аутентифікації для систем Інтернету речей на основі Diffie-Hellman key exchange protocol // Вісник Національного університету "Львівська політехніка". 2020. Вип. 1 (912). С. 129-134.

11. Клименко, І. Методи інформаційної безпеки в системах Інтернету речей [Електронний ресурс] / І. Клименко, В. Шуляр // Науково-технічний журнал "Комп'ютерні науки та інформаційні технології". - 2019. - № 12(242). - С. 50-57. - Режим доступу: https://doi.org/10.32627/ktit.2019.12.050

12. Інформаційна безпека систем Інтернету речей: проблеми та можливості [Текст] / О. Дмитренко, О. Горбатенко, І. Кузнецова та ін.; за ред. О. Дмитренка. - Київ : КНУТД, 2017. - 188 с.

13. Гончаренко, В. І. Методи створення безпеки в системах Інтернету речей [Електронний ресурс] / В. І. Гончаренко, Є. Є. Клименко // Технології Інтернету речей та кіберфізичних систем. - 2018. - Т. 3, № 4. - С. 4-12. - Режим доступу: http://tiirs.org.ua/wp-content/uploads/2019/02/3_4_2018.pdf

14. Подольчак, О. Аналіз питань безпеки Інтернету речей та шляхи їх вирішення [Електронний ресурс] / О. Подольчак, Л. Голубович // Збірник наукових праць Східноєвропейського національного університету імені Лесі Українки. Серія: "Комп'ютерні науки та інформаційні технології". - 2019. - Вип. 13 (263). - С. 92-98. - Режим доступу: https://cyberleninka.org/article/n/analiz-pytan-bezpeky-internetu-rechei-ta-shlyakhy-yikh-vyreshennia

15. Міщенко, І. В. (2018). Методи аутентифікації користувача в IoT-системах. Інформаційні технології та комп'ютерна інженерія, 1(55), 82-88.

16. Шинкарук, О. О., Корж, І. І., & Стеблій, М. І. (2020). Захист від кіберзагроз в системах Інтернету речей на основі методів аутентифікації та шифрування. Інформаційні технології та комп'ютерна інженерія, 3(63), 83-90.

17. Петренко, В. В. (2017). Методи та засоби захисту інформації в системах Інтернету речей. Системні технології, 1(76), 27-35.

18. Литвиненко, В. С. (2019). Методи та засоби аутентифікації в системах Інтернету речей. Вісник Чернігівського національного технологічного університету, 2(97), 57-61.

19. Кудрявцев, С. М., Гальчинський, А. В., & Мартинюк, І. В. (2019). Методи та засоби забезпечення безпеки в системах Інтернету речей. Технічна електродинаміка, 5(6), 52-61.

References

6.Petrenko A. Yu., Kovalev V. V. Rozrobka prohramnoho zasobu heneratsii kliuchiv autentyfikatsii dlia zakhystu prystroiv Internetu rechei // Visnyk Chernihivskoho derzhavnoho tekhnolohichnoho universytetu. 2020. № 2 (95). S. 95-101.

7. Havrylenko O. M., Chuba O. V. Metod heneruvannia kliuchiv autentyfikatsii dlia system Internetu rechei // Systemni doslidzhennia ta informatsiini tekhnolohii. 2019. № 4 (70). S. 86-96.

8. Koval O. V., Kovalchuk V. Ye. Metod heneruvannia kliuchiv autentyfikatsii na osnovi infrachervonykh tekhnolohii dlia system Internetu rechei // Systemni tekhnolohii. 2020. T. 8, № 1. S. 15-24.

9. Ivanov A. V., Chernysh V. V. Analiz metodiv heneruvannia kliuchiv autentyfikatsii dlia prystroiv Internetu rechei // Tekhnichni nauky ta tekhnolohii. 2018. T. 3, № 1. S. 9-18.

10. Melnyk V. H., Kondratiuk O. M. Metod heneratsii kliuchiv autentyfikatsii dlia system Internetu rechei na osnovi Diffie-Hellman key exchange protocol // Visnyk Natsionalnoho universytetu "Lvivska politekhnika". 2020. Vyp. 1 (912). S. 129-134.

11. Klymenko, I. Metody informatsiinoi bezpeky v systemakh Internetu rechei [Elektronnyi resurs] / I. Klymenko, V. Shuliar // Naukovo-tekhnichnyi zhurnal "Kompiuterni nauky ta informatsiini tekhnolohii". - 2019. - № 12(242). - S. 50-57. - Rezhym dostupu: https://doi.org/10.32627/ktit.2019.12.050

12. Informatsiina bezpeka system Internetu rechei: problemy ta mozhlyvosti [Tekst] / O. Dmytrenko, O. Horbatenko, I. Kuznetsova ta in.; za red. O. Dmytrenka. - Kyiv : KNUTD, 2017. - 188 s.

13. Honcharenko, V. I. Metody stvorennia bezpeky v systemakh Internetu rechei [Elektronnyi resurs] / V. I. Honcharenko, Ye. Ye. Klymenko // Tekhnolohii Internetu rechei ta kiberfizychnykh system. - 2018. - T. 3, № 4. - S. 4-12. - Rezhym dostupu: http://tiirs.org.ua/wp-content/uploads/2019/02/3_4_2018.pdf

14. Podolchak, O. Analiz pytan bezpeky Internetu rechei ta shliakhy yikh vyrishennia [Elektronnyi resurs] / O. Podolchak, L. Holubovych // Zbirnyk naukovykh prats Skhidnoievropeiskoho natsionalnoho universytetu imeni Lesi Ukrainky. Seriia: "Kompiuterni nauky ta informatsiini tekhnolohii". - 2019. - Vyp. 13 (263). - S. 92-98. - Rezhym dostupu: https://cyberleninka.org/article/n/analiz-pytan-bezpeky-internetu-rechei-ta-shlyakhy-yikh-vyreshennia

15. Mishchenko, I. V. (2018). Metody autentyfikatsii korystuvacha v IoT-systemakh. Informatsiini tekhnolohii ta kompiuterna inzheneriia, 1(55), 82-88.

16. Shynkaruk, O. O., Korzh, I. I., & Steblii, M. I. (2020). Zakhyst vid kiberzahroz v systemakh Internetu rechei na osnovi metodiv autentyfikatsii ta shyfruvannia. Informatsiini tekhnolohii ta kompiuterna inzheneriia, 3(63), 83-90.

17. Petrenko, V. V. (2017). Metody ta zasoby zakhystu informatsii v systemakh Internetu rechei. Systemni tekhnolohii, 1(76), 27-35.

18. Lytvynenko, V. S. (2019). Metody ta zasoby autentyfikatsii v systemakh Internetu rechei. Visnyk Chernihivskoho natsionalnoho tekhnolohichnoho universytetu, 2(97), 57-61.

19. Kudriavtsev, S. M., Halchynskyi, A. V., & Martyniuk, I. V. (2019). Metody ta zasoby zabezpechennia bezpeky v systemakh Internetu rechei. Tekhnichna elektrodynamika, 5(6), 52-61.

ДОДАТОК Б

ПРЕЗЕНТАЦІЯ

Метод та програмно-технічний засіб генерування ключів аутентифікації

Науковий керівник: к. т. н., доцент Іванов О. В.

Доповідач: Томусяк А. В.

- **Об'єкт дослідження** - процес створення методу та програмно-технічного засобу генерування ключів аутентифікації.
- **Предмет дослідження** - модель, метод та програмно - технічний засіб генерування ключів аутентифікації
- **Мета** - розробити новий метод та програмно-технічний засіб генерування ключів аутентифікації

Задачі дослідження:

1)провести аналіз вже існуючих методів та програмно - технічних засобів генерування ключів аутентифікації та визначити їх недоліки;

2)розробити метод для безпечного та конфіденційного введення ключів під час етапу генерування ключів аутентифікації, який би став альтернативою вже відомих методів й надав можливість зберігати інформацію користувача та його конфіденційність в системі IoT на більш вищому рівні, ніж вже існуючі ;

3)спроектувати програмно - технічний засіб для вирішення проблем з безпечним та конфіденційним генеруванням ключів аутентифікації, який би надав можливість ефективного захисту пристроїв IoT, в порівнянні з відомими;

4)на основі розроблених методу та програмно - технічного засобу генерування ключів аутентифікації, створити ефективну систему безпеки та конфіденційності при генеруванні ключів аутентифікації, ніж вже існуючі.

Наукова новизна:

1)Вдосконалено метод безпечного та конфіденційного генерування ключів аутентифікації, який, на відміну від відомих методів, що використовують генерування заводських ключів аутентифікації, які не створюють повну безпеку в системі пристроїв IoT, дозволяє користувачу самому налаштовувати систему, при цьому не володіючи технічними знаннями;

2)Спроектовано програмно - технічний засіб генерування ключів аутентифікації, який діє, як шлюз Brandmauer та використовує програмування протоколів обміну ключами у протоколи ІЧ передачі. Застосування засобу зможе перетворити систему пристроїв IoT в анонімну та не дасть впливу третіх осіб на неї.

Практична цінність роботи полягає в проектуванні метода та програмно - технічного засобу

Генерування ключів аутентифікації, яка надає ефективність у безпеці та конфіденційності під час генерування ключів аутентифікації, в порівнянні з існуючими пристроями та контроль за процесом поточним користувачем.

Актуальність завдання

1)Не дивлячись на те, що безпека WPS кожен рік удосконалюється і розвиток технологій йде вперед, завжди будуть з'являтися вразливості в системі. Ця робота включає в себе те, що безпека та конфіденційність, особливо при початковому генеруванні ключів аутентифікації, залишиться протягом довгого часу, виключаючи атаки та вразливості ззовні;

2)Актуальним є й зручне використання програмно - технічного засобу для користувачів, які самостійно, без допомоги, проведуть генерування ключів аутентифікації;

3)З цим методом та програмно - технічним засобом генерування ключів аутентифікації, більшість з них зможуть легко керувати захищеністю своєї інформації та контролювати підключення власних пристроїв IoT до мережі, забезпечуючи захист від впливу ззовні.

Пристрої Iot та спеціальний програмно - технічний засіб

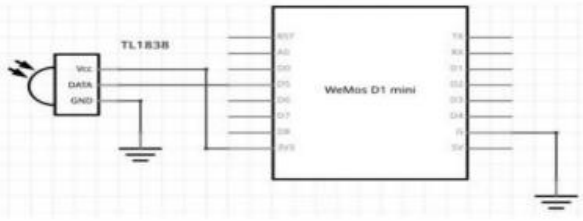

Схема ІЧ-приймача

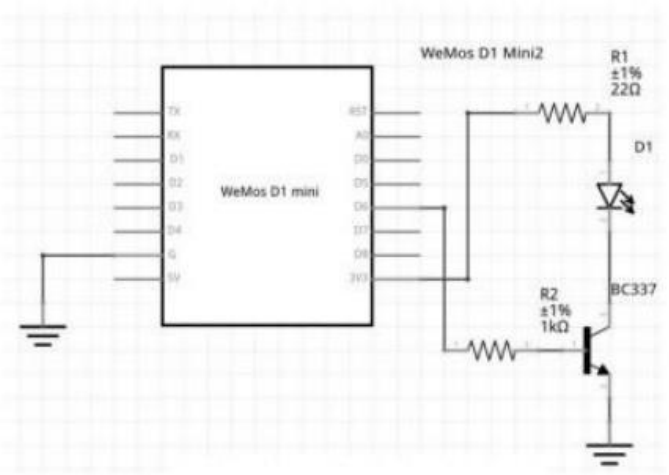

Схема передачі ІЧ -світлодіодного передавача

Три робочі можливості для користувача по налаштуванню точки доступу

```
import requests
router_ip='http://192.168.0.1'
auth_token="Authorization=Basic%20YWRtaW46YjAxYzZmYzYyMDgwMzA5Y2ZiMzc2ZTE4NzI3Yz

def login():
    r = requests.get(router_ip+'/userRpm/LoginRpm.htm?Save=Save',headers={'Refer

    if r.status_code==200:
        x=1
        while x<3:
            try:
                session_id=r.text[r.text.index(router_ip)+len(router_ip)+1:r.tex
                return session_id
                break
            except ValueError:
                return 'Login error'
            x+=1
    else:
        return 'IP unreachable'

r = requests.get(router_ip+'/'+session+'/userRpm/SysRebootRpm.htm?Reboot=%D0%9F%

r = requests.get(router_ip+'/'+session+'/userRpm/VirtualServerRpm.htm?doAll=EnAl

r = requests.get(router_ip+'/'+session+'/userRpm/VirtualServerRpm.htm?doAll=DisA

r = requests.get(router_ip+'/'+session+'/userRpm/ManageControlRpm.htm?port=5110&
r = requests.get(router_ip+'/'+session+'/userRpm/WlanStationRpm.htm',headers={'R

presence='Дома находятся:'
if 'DC-31-54-97-51-06' in r.text:
            presence=presence+'\n'+'DC-31-54-97-51-06'
```

Алгоритм контролю пристрою

Висновки

1)Проведено аналіз методів генерування ключів аутентифікації та визначено недоліки;

2)Розроблено метод генерування ключів аутентифікації, який є альтернативою відомим методам та створює, завдяки включенню нашого коду та схеми шифрування на основі **Diffie–Hellman** key exchange protocol, повну безпеку та конфіденційність користувача;

3)Спроектовано спеціальний засіб на основі розробленого методу генерування ключів аутентифікації, який завдяки вдосконаленню дає ефективний захист та виключає зовнішні атаки, надаючи повну безпеку під час етапу ініціалізації пристроїв IoT.

Дякую за увагу!

ДОДАТОК В

Код - ESP8266 Controller Code

```
/**
 * \defgroup main main.cpp TFM Controller Code
 * 2019 - Rafael de la Rosa Ortiz \n1
 * Allows a ESP8266 to transmit a <SSID,PK> to an IoT node.\n
 * It generates a random SSID and Key, using true random numbers, and saves it in its
EEPROM.\n
 * It allows to send the Key to a node securely by implementing a Diffie Hellman Key
Exchange Protocol.\n
 * Once information is sent correctly, it displays a QR code which encodes it, allowing
the
user to scan it with a phone camera.
 * @{
 */
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>
#include <IRsend.h>
#include <Adafruit_SSD1306.h>
#include <qrcode.h>
#include <EEPROM.h>
#include <CRC32.h>
#include <ESP8266TrueRandom.h>
#include <curve25519-donna.h>
#include <string.h>
#include <math.h>
// ****************************************************************
/**
* \defgroup global_variables 1. Global Variables Definition
* @{
*/
/**
* \defgroup eeprom_control_variables 1.1. Variables to Control EEPROM Status and Content
* @{
*/
/// Remote Controller ID - Always 1 (hardcoded).
const unsigned char devID = 1;
/// Node ID to Communicate to (Selected when pushing the button).
unsigned char selected_devID = 0;
/// Node ID to Communicate to. Auxiliar temporal var assigned in contactNode(), equal to
selected_devID unless the latter is 0.
unsigned char dest_devID;
/// EEPROM Control Char target, 'Y'.
const unsigned char CONTROL_EEPROM_TARGET_CHAR[2] = { 'Y', '\0'};
/// EEPROM Read Control Char is stored here. If equal to 'Y', memory was written before.
unsigned char eeprom_control_char[2];
/// Max Number of Nodes is 206 (Limited by EEPROM).
#define NUM_NODES_MAX 206
81
/// Array of Nodes. +1 to store empty char '\0' at last position.
unsigned char nodes_array[NUM_NODES_MAX + 1];
/// Number of Nodes. Default is 1. Value may change after reading EEPROM.
unsigned char number_nodes = 1;
/**@}*/
/**
```

```
* \defgroup eeprom_address_map 1.2. EEPROM ADDRESS MAP
* It uses 256 addresses for the Controller. Each address can store 8 bits.
* @{
*/
/// 0x00, ssid size = 16 B.
#define ADDR_SSID 0
/// 0x10, psk size = 32 B.
#define ADDR_PSK 16
/// 0x30, n_nod size = 1 B.
#define ADDR_NUM_NODES 48
/// 0x31, number nodes up to 206 Nodes.
#define ADDR_NODE_1 49
/// 0xFF, control character. if char=='Y', memory has data.
#define ADDR_CONTROL_CHAR 255
/**@}*/
/**
* \defgroup wifi_credentials 1.3. <SSID,PSK> global variables
* @{
*/
/// SSID Max Size is 16 Bytes
#define SSID_MAX_SIZE 16
/// SSID Max Size is 32 Bytes
#define PSK_MAX_SIZE 32
/// Variables were EEPROM ssid data is stored. Initialized as empty "".
unsigned char ssid[SSID_MAX_SIZE + 1] = "";
/// Variables were EEPROM psk data is stored. Initialized as empty "".
unsigned char psk[PSK_MAX_SIZE + 1] = "";
/**@}*/
/// Macro which translates a byte (8 bits) to 2 HEX Chars (4+4 bits).
#define TO_HEX(i) ( (i) <= 9 ? '0' + (i) : 'A' - 10 + (i) )
////////////////////////////////////////////////////////////////
// Variables used in IR Send/Receive
////////////////////////////////////////////////////////////////
/**
* \defgroup ir_variables 1.4. Variables used in IR Send/Receive
* @{
*/
/// IR RX Pin D6 GPIO12 in ESP8266 WifiKit 8.
const uint8_t kRecvPin = 12;
/// Set the GPIO to be used to receive data.
IRrecv irrecv(kRecvPin);
/// IR TX D7 GPIO13 in ESP8266 WifiKit 8.
const uint8_t kIrLed = 13;
/// Set the GPIO to be used to send data.
IRsend irsend(kIrLed);
/// Variable where a received message is stored to.
decode_results received_data;
/// Variable where a message is stored before being sent.
decode_results send_data;
/// Transmission and Reception Buffers Size.
#define MAX_PACKET_SEQUENCE 1024
/// Reception data buffer of 32bits/packet.
uint32_t rx_data_buffer[MAX_PACKET_SEQUENCE];
82
/// Transmission Data buffer of 32bits/packe
uint32_t tx_data_buffer[MAX_PACKET_SEQUENCE];
/**@}*/
////////////////////////////////////////////////////////////////
// Variables used in IR Frame Control
////////////////////////////////////////////////////////////////
/** \defgroup frame_packet_types 1.5. Frame Packet Types
 * Variables used in IR Frame Control:
 * 3 bits for type of packet.\n
 * 8 bits for dest deviceID.\n
```

```
 * 32 bits for data.\n
 * 16 bits for CRC16 Checksum.\n
 * Type of Packet:\n
 * 0, 000 -> HELLO_ENCRYPTED.\n
 * 1, 001 -> HELLO.\n
 * 2, 010 -> HELLO_NEW.\n
 * 3, 011 -> DATA.\n
 * 4, 100 -> ACK.\n
 * 5, 101 -> DATA_END (Last Data Packet -> Checksum).\n
 * 6, 110 -> DATA_END_ENCRYPTED (Last Data Packet (encrypted) -> Checksum).\n
 * 7, 111 -> NACK - Not Used.\n
 * @{
 */
#define TYPE_HELLO_ENCRYPT 0b000
#define TYPE_HELLO 0b001
#define TYPE_HELLO_NEW 0b010
#define TYPE_DATA 0b011
#define TYPE_ACK 0b100
#define TYPE_DATA_END 0b101
#define TYPE_DATA_END_ENCRYPT 0b110
#define TYPE_NACK 0b111
/**@}*/
/**
* \defgroup program_flags 1.6. FLAGS used by the program to distinguish between states
* @{
*/
/// Hello Communication Flag. 1 means HELLO Exchange was done, so don't listen for more
HELLO
Packets.
uint8_t FLAG_HELLO_DONE = 0;
/// Encryption Flag. 1 means Data is encrypted(DH Key Exchange). Default is 1.
uint8_t FLAG_ENCRYPT_DATA = 1;
/**@}*/
////////////////////////////////////////
// VARIABLES USED IN OLED DISPLAY
////////////////////////////////////////
/**
* \defgroup oled_display_variables 1.7. Variables used in OLED Display
* @{
*/
/// OLED display width, 128 pixels.
#define SCREEN_WIDTH 128
/// OLED display height, 32 pixels.
#define SCREEN_HEIGHT 32
/// Displat Reset PIN Number, 16.
#define OLED_RESET 16
/// Init declaration for an SSD1306 display connected to I2C (SDA, SCL pins).
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
/// QR Code Variable to be printed in the display.
QRCode qrcode;
83
/**@}*/
/**
* \defgroup button_interrupt_variables 1.8. Variables to handle Button Interruptions
* @{
*/
/// PIN used by the input button of the micro.
#define BUTTON_PIN 0
/// Bool used to distinguish is button was pressed or released.
bool buttonIsHigh = false;
/// Timer representing the instant the button was pressed.
long timeInit = 0.0;
/// Timer representing the instant the button was released.
long timeEnd = 0.0;
```

```
/// Bool that indicates a menu needs to refresh to reflect changes due to user input
bool screen_needs_refresh = false;
/// Bool that turns true when a long buttom press is registered.
bool long_press = true;
/// Bool that allows program to exit devID selection loop.
bool devID_confirmed = true;
/// Bool that allows program to exit encryption selection loop.
bool encrypt_confirmed = true;
/// Bool that prints the stages of communication in the OLED display. It slows down the
transmission several ms.
bool verbose_display = false;
/// Unsigned char representing a number used to show progress in OLED display [x/10].
unsigned char current_entry = 1;
/**@}*/
/**@}*/
// ****************************************************************************
/**
* \defgroup functions 2. Functions
* Module with all the custom functions implemented in the controller source code
* @{
*/
/**
* \defgroup debug_functions 2.1. DEBUG Functions
* Functions which are used for debugging purposes (printing logs and traces mainly)
* @{
*/
/**
 * DEBUG Function which prints in terminal the time elapsed in seconds.
 * @param[in] long ref: Reference time where the counting period begins.
 */
void printTimeSeconds(long ref)
{
    double elapsed = (millis() - ref) / 1000.0;
    Serial.printf(" - Time Elapsed: %.4f seconds.\n", elapsed);
}
/**
 * DEBUG function used to print in terminal information regarding received packets.
 * @param[in] uint32_t header_capture: integer that represents the 32bits captured in a
IR
frame.
 * @param[in] uint16_t received_counter: integer that represents the current packet
counter
in the data exchange.
 * @param[in] uint8_t received_type: integer that represents the type of packet that has
been received.
 */
void printCapturedSeq(uint32_t header_capture, uint16_t received_counter, uint8_t
received_type)
{
    Serial.println("\nDEBUG: printCapturedSeq() HEX:");
    84
 Serial.print("Captured: "); Serial.println(header_capture, HEX);
    Serial.print("Counter: "); Serial.println(received_counter, HEX);
    Serial.print("packetType: "); Serial.println(received_type, HEX);
    Serial.println("*****\n");
}
/// DEBUG function used to print in terminal sent packet frame, represented in the global
variable send_data.
void printSentPacket() {
    Serial.println("printSentPacket() info:");
    serialPrintUint64(send_data.value, BIN);
    Serial.println("");
    serialPrintUint64(send_data.value, HEX);
    Serial.println("");
```

```
}
/// DEBUG function used to print in terminal received packet frame, represented in the
global
variable received_data.
void printReceivedPacket() {
    Serial.println("printReceivedPacket() info:");
    //serialPrintUint64(received_data.value, BIN);
    //Serial.println("");
    serialPrintUint64(received_data.value, HEX);
    Serial.println("");
}
/**
 * DEBUG function used to print in terminal information regarding getRXFrameData()
function
output.
 * @param[in] uint8_t destID: integer that represents frame's Device Destination ID
value.
 * @param[in] uint8_t packet_type: integer that represents the current packet's type.
 * @param[in] uint8_t is_from_controller: integer of which only the LS bit is used. 1
means
packet comes from controller device.
 * @param[in] uint32_t payload: integer representing the payload data of the frame.
 */
void printRXPacketInfo(uint8_t destID, uint8_t packet_type, uint8_t is_from_controller,
uint32_t payload)
{
    Serial.println("\nDEBUG: printRXPacketInfo()[BIN]: ");
    Serial.print("destID: "); Serial.println(destID, BIN);
    Serial.print("packetType: "); Serial.println(packet_type, BIN);
    Serial.print("isfromcontroller: "); Serial.println(is_from_controller, BIN);
    Serial.print("payload: "); Serial.println(payload, BIN);
    Serial.println("*****\n");
}
/// DEBUG function used to print stored EEPROM parameters: SSID, PSK, devID,
number_nodes...
void printEEPROMData()
{
    Serial.print("\nSSID: ");
    Serial.println((char*)ssid);
    Serial.print("PSK: ");
    for (int i = 0; i < 32; i++)
    {
        Serial.printf("%c", TO_HEX(psk[i]));
    }
    Serial.println();
    for (int i = 0; i < 32; i++)
    {
        Serial.printf("%d ", psk[i]);
    }
    Serial.println();
    Serial.print("DeviceID: ");
    Serial.println(devID);
    Serial.print("Number of Nodes: ");
    Serial.println(number_nodes);

    Serial.print("List of devIDs stored: ");
    for (int i = 0; i < number_nodes; i++)
    {
        Serial.print(nodes_array[i]);
        Serial.print(", ");
    }
    85
 Serial.println("");
```

```
    Serial.print("Control char. target/actual: ");
    Serial.print((char)CONTROL_EEPROM_TARGET_CHAR[0]);
    Serial.print("/");
    Serial.println((char)eeprom_control_char[0]);
    Serial.println("");
}
/**@}*/
/**
* \defgroup aux_functions 2.2. Program Auxiliar Functions
* Functions which are called to perform important processes during the execution of the
main
functions
* @{
*/
/**
 * Function which delays execution a variable random amount of time.
 * @param[in] bool short_time: If true, the delay will be a short delay.
 */
void wait(bool short_time)
{
    if (short_time)
        delay(150 + (rand() % 10)); // Wait 150 to 160 ms
    else { delay(2000 + (rand() % 1000)); } // wait 2000 to 3000 ms
}
/**
 * Function that generates a random SSID that will be saved into device EEPROM during its
initialization.
 * @param[out] unsigned char* result: pointer to which random generated SSID will be
saved.
 */
void generateRandomSSID(unsigned char* result)
{
    const unsigned char seq[63] =
   "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    strncpy((char*)result, "IOT_", SSID_MAX_SIZE + 1);
    for (int i = 4; i < SSID_MAX_SIZE; i++)
    {
        if (i < 10) { result[i] = seq[ESP8266TrueRandom.random(0, 52)]; }
        else if (i == 10) { result[i] = '-'; }
        else { result[i] = seq[ESP8266TrueRandom.random(52, 63)]; }
    }
    Serial.print("New random SSID generated: ");
    Serial.println((char*)result);
}
/**
 * Function that generates a random PSK that will be saved into device EEPROM during its
initialization.
 * @param[out] unsigned char* result: pointer to which random generated PSK will be
saved.
 */
void generateRandomPSK(unsigned char* result)
{
    for (int i = 0; i < 4; i++)
    { // 1 long 32 bits, divided by 4bits/HEXchar = 8 chars -> need
        array of 4 longs
         uint32_t aux = ESP8266TrueRandom.random();
        result[8 * i + 0] = (unsigned char) ((aux & 0xF0000000) >> 28);
result[8 * i + 1] = (unsigned char) ((aux & 0x0F000000) >> 24);
result[8 * i + 2] = (unsigned char) ((aux & 0x00F00000) >> 20);
result[8 * i + 3] = (unsigned char) ((aux & 0x000F0000) >> 16);
result[8 * i + 4] = (unsigned char) ((aux & 0x0000F000) >> 12);
result[8 * i + 5] = (unsigned char) ((aux & 0x00000F00) >> 8);
result[8 * i + 6] = (unsigned char) ((aux & 0x000000F0) >> 4);
result[8 * i + 7] = (unsigned char) ((aux & 0x0000000F) >> 0);
```

```
// Serial.print("Raw 32bits: "); serialPrintUint64((uint64_t) aux, BIN);
// Serial.print("\nChars from Byte: ");
// Serial.printf("%c | %c | %c | %c | %c | %c | %c | %c\n\n", TO_HEX(result[8*i+0]),
86
TO_HEX(result[8 * i + 1]), TO_HEX(result[8 * i + 2]), TO_HEX(result[8 * i + 3]),
TO_HEX(result[8 * i + 4]), TO_HEX(result[8 * i + 5]), TO_HEX(result[8 * i + 6]),
TO_HEX(result[8 * i + 7]));
 }
 Serial.print("New random PSK generated: ");
for (int i = 0; i < 32; i++)
{
    Serial.printf("%c", TO_HEX(result[i]));
}
Serial.println();
}
/**
 * Function used to simplify the process of reading a single entry stored in the EEPROM.
 * @param[in] uint8_t start_addr: integer that represents the initial address where the
information is stored.
 * @param[in] uint8_t num_bytes: integer that represents the number of bytes (or
addresses)
to read.
 * @param[out] unsigned char* result: pointer to where the read output should be stored.
 */
void readEEPROM(uint8_t start_addr, uint8_t num_bytes, unsigned char* result)
{
    EEPROM.begin(256);
    uint8_t index = 0;
    result[index] = (EEPROM.read(start_addr + index));
    while (index < num_bytes - 1)
    {
        index = index + 1;
        result[index] = (EEPROM.read(start_addr + index));
    }
    EEPROM.end();
}
/**
 * Function used to simplify the process of writing a single entry to the EEPROM.
 * @param[in] uint8_t start_addr: integer that represents the initial address where the
information is stored.
 * @param[in] unsigned char *data: pointer to where the read output should be stored.
 * @param[in] uint8_t size: integer that represents the number of bytes (or addresses) to
read.
 */
void writeEEPROM(uint8_t start_addr, unsigned char* data, uint8_t size)
{
    EEPROM.begin(256);
    uint8_t index = 0;
    EEPROM.write(start_addr + index, data[index]);
    while (index < size - 1)
    { //data[index] != '\0' &&
        index = index + 1;
        EEPROM.write(start_addr + index, data[index]);
    }
    EEPROM.commit();
    EEPROM.end();
}
/// Function used to represent in the OLED display the device ID selection menu.
void displayDeviceIDSelection()
{
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
```

```
    display.setCursor(0, 0);
    display.println(F("Select devID (Short Push) | Confirm (Long)"));
    display.print(F("["));
    display.print(current_entry);
    display.print(F("/"));
    display.print(number_nodes);
    if (selected_devID == 0)
    {
        display.print(F("] Add New Device"));
        87
    }
    else
    {
        display.print(F("] Connect to ID "))
display.print(selected_devID);
 }
 display.display();
}
/// Function used to show in the display the Communication Encryption Selection Menu.
void displayEncryptedCommSelection()
{
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println(F("Short Push: Switch | Long Push: Confirm"));
    display.print(F("Use Encrypted Communication? "));
    if (FLAG_ENCRYPT_DATA) { display.print("YES"); Serial.println("Current Choice: YES");
}
    else { display.print("NO"); Serial.println("Current Choice: NO"); }
    display.display();
}
/// Function used to display QR Code in both OLED and Terminal
void displayOLEDQRCode()
{
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.display();
    wait(1);
    long dt = millis();
    uint8_t qrcodeData[qrcode_getBufferSize(3)]; // Version 3 -> size 29x29, 53 bytes of
    data if low error correction
 char* str = (char*)malloc(53);
    char* psk_hex = (char*)malloc(PSK_MAX_SIZE + 1);
    strcpy(str, (char*)ssid);
    strcat(str, ":\n");
    for (int i = 0; i < PSK_MAX_SIZE; i++)
    {
        psk_hex[i] = TO_HEX(psk[i]);
        Serial.printf("|%c|", TO_HEX(psk[i]));
    }
    strncat(str, psk_hex, 32);
    printf("\n%s\n", str);
    qrcode_initText(&qrcode, qrcodeData, 3, 0, str);
    printTimeSeconds(dt);
    for (uint8_t y = 0; y < 32; y++)
    {
        for (uint8_t x = 0; x < 128; x++)
        {
            display.drawPixel(x, y, WHITE);
        }
```

```
    }
    for (uint8_t y = 0; y < qrcode.size; y++)
    {
        Serial.print(" ");
        for (uint8_t x = 0; x < qrcode.size; x++)
        {
            // Print each module (UTF-8 \u2588 is a solid block)
            Serial.print(qrcode_getModule(&qrcode, x, y) ? "\u2588\u2588" : " ");
            if (qrcode_getModule(&qrcode, x, y))
            {
                display.drawPixel(48 + x, y + 1, BLACK);
            }
        }
        Serial.println();
    }
    Serial.println("\n\n\n");
    free(str);
    free(psk_hex);
    88
 display.ssd1306_command(SSD1306_SETCONTRAST);
    display.ssd1306_command(0);
    display.display();
    wait(0);
    wait(0);
    wait(0);
}
/// Function used to initialize the EEPROM with default parameters, generating random
< SSID,PSK >.
void setupEmptyEEPROM()
{
    Serial.println("EEPROM WAS NOT SET UP. CHARACTER in 0xFF WAS NOT 'Y'. Proceeding to
   initialize the memory with new SSID/ PSK");

    generateRandomSSID(ssid);
    generateRandomPSK(psk);
    writeEEPROM(ADDR_SSID, ssid, SSID_MAX_SIZE);
    writeEEPROM(ADDR_PSK, psk, PSK_MAX_SIZE);
    // Conversion from int to char array - number of Nodes (initially 1)
    unsigned char number_nodes_char[1];
    number_nodes_char[0] = number_nodes;
    writeEEPROM(ADDR_NUM_NODES, number_nodes_char, 1);
    // Conversion from int to char array - devID of controller (1)
    unsigned char devID_char[1];
    devID_char[0] = devID;
    writeEEPROM(ADDR_NODE_1, devID_char, 1);
    // Write control char 'Y' in 0xFF to tell device's EEPROM has data
    strncpy((char*)&eeprom_control_char, (char*)&CONTROL_EEPROM_TARGET_CHAR, 2);
    writeEEPROM(ADDR_CONTROL_CHAR, eeprom_control_char, 1);
}
/// Function used to read all EEPROM configuration values and load them in the program
global
variables.
void loadConfigValuesFromEEPROM()
{
    // Read SSID,PSK
    readEEPROM(ADDR_SSID, SSID_MAX_SIZE, ssid);
    readEEPROM(ADDR_PSK, PSK_MAX_SIZE, psk);
    // Read Num of registered Nodes
    unsigned char number_nodes_char[2];
    readEEPROM(ADDR_NUM_NODES, 2, number_nodes_char);
    number_nodes = number_nodes_char[0];
    // Read each registered NodeID
    for (int i = 0; i < number_nodes; i++)
    {
```

```
        unsigned char current_devID[2];
        readEEPROM(ADDR_NODE_1 + i, 2, current_devID);
        nodes_array[i] = current_devID[0];
        nodes_array[i + 1] = '\0';
    }
}
/**
 * Function that performs the Data integrity Check for a burst of received packets.\n
 * It does a CRC32 check and determines if the data is valid.
 * @param[in] uint16_t size: Number of received packets in the RX_buffer.
 * @param[in] uint32_t rx_checksum: Checksum Received in the last data packet.
 * @return bool: True is data is valid. False if data is not valid.
 */
bool checkDataIntegrity(uint16_t size, uint32_t rx_checksum)
{
    uint32_t checksum = CRC32::calculate(rx_data_buffer, size);
    Serial.println("Checking Checksums...");
    Serial.printf("Received: %d | calculated: %d\n", rx_checksum, checksum);
    89
 if (checksum == rx_checksum)
        return true;
    Serial.println("Checksums not matching!");
    return false;
}
/**
 * Function used to handle button interruptions, mainly used in selection menu
navigation.\n
 * As there is a single button available, function is called in both press and release
actions.\n
 * It determines if the button was pressed or released, and differenciate between a short
and a long press.
 */
void handleButtonInterrupt()
{
    if (!buttonIsHigh)
    { // Falling Edge (Low to High)
        long timeNow = millis();
        if (timeNow - timeInit < 50)
        { //Anti-Bouncing
            timeInit = timeNow;
            return;
        }
        buttonIsHigh = true;
        Serial.println("BUTTON PRESSED");
        timeInit = millis();
        return;
    }
    // else -> Rising Edge (High to Low), button was released
    Serial.println("BUTTON RELEASED");
    buttonIsHigh = false;
    timeEnd = millis();
    Serial.print("Time Elapsed: ");
    Serial.println(timeEnd - timeInit);
    if (timeEnd - timeInit <= 100) // Anti-Bouncing, too short press, do nothing
        return;

    if (timeEnd - timeInit <= 1000)
    { // Short Press, change selected DeviceID
        timeEnd = 0.0;
        timeInit = 0.0;
        if (!devID_confirmed)
        {
            if (number_nodes > 1)
            {
```

```
                Serial.println("Number nodes: ");
                Serial.println(number_nodes);
                if (selected_devID == nodes_array[number_nodes - 1])
                { // If Last Device was Selected, go
                    Back to 0
                selected_devID = 0;
                    current_entry = 1;
                    screen_needs_refresh = true;
                    return;
                }
                if (selected_devID == 0)
                { // Skip passing from new Device(0) to 1, as ID=1 is his own ID
                    selected_devID = 2;
                    current_entry = 2;
                    screen_needs_refresh = true;
                    return;
                }
                for (int id = 0; id < number_nodes - 1; id++)
                { // Iterate to go to the next ID in list
                    if (selected_devID == nodes_array[id])
                    {
                        selected_devID = nodes_array[id + 1];
                        current_entry = current_entry + 1;
                        screen_needs_refresh = true;
                        return;
                    }
                }
            }
            90
        }
        else if (!encrypt_confirmed)
        {
            FLAG_ENCRYPT_DATA = !FLAG_ENCRYPT_DATA;
            screen_needs_refresh = true;
            return;
        }
    }
    else
    { // Long Press (+1000ms), Confirm Selected Device
        long_press = true
    if (!devID_confirmed) { devID_confirmed = true; return; }
        else if (!encrypt_confirmed) { encrypt_confirmed = true; return; }
    }
}
/**
 * Function used to convert the info to send into a formatted frame ready to be sent.
 * @param[in] uint8_t destID: device to which the packet is sent.
 * @param[in] uint8_t packet_type: integer that represents the current packet's type.
 * @param[in] uint32_t payload: integer representing the payload data of the frame.
 * @return uint64_t: integer which represents the frame ready to be sent (up to 36 bits,
RC6 IR protocol).
 */
uint64_t buildTXFrame(uint8_t destID, uint8_t packet_type, uint32_t payload)
{
    uint64_t data = 0ULL;
    if (packet_type == TYPE_ACK || packet_type == TYPE_NACK)
    {
        data =
        static_cast<uint64_t>(payload & 0xFFF) << 0 |
        static_cast<uint64_t>(1ULL) << 12 | // Packet Sent From Controller,
       always 1
    static_cast<uint64_t>(packet_type & 0b111) << 13;
    }
    else if (packet_type == TYPE_HELLO || packet_type == TYPE_HELLO_ENCRYPT ||
```

```
   packet_type == TYPE_HELLO_NEW)
    {
        data =
        static_cast<uint64_t>(payload) << 0 |
        static_cast<uint64_t>(1ULL) << 24 | // Packet Sent From Controller, always 1
        static_cast<uint64_t>(destID) << 25 |
        static_cast<uint64_t>(packet_type) << 33;
    }
    else if (packet_type == TYPE_DATA || packet_type == TYPE_DATA_END_ENCRYPT ||
   packet_type == TYPE_DATA_END)
    {
        data =
        static_cast<uint64_t>(payload) << 0 |
        static_cast<uint64_t>(1ULL) << 32 | // Packet Sent From Controller, always 1
        static_cast<uint64_t>(packet_type) << 33;
    }

    //Serial.println("DEBUG: buildTXFRAME:");
    //serialPrintUint64(data, HEX);
    //Serial.println();
    //serialPrintUint64(data, BIN);
    //Serial.println("\n");
    return data;
}
/**
 * Function which processes the received frame stored in received_data global variable.\n
 * It processes the information of the frame and determines if the packet is valid.
 * @param[in] uint16_t current_counter: represents the number of received packets in the
current burst.
 * @return uint32_t: integer which contains 2 different variables:\n
 * Packet type (8bits) in the 8 LSBs.\n
 * New Counter (16bits) in the 16 MSBs. If NewCounter==current_counter_input+1, it means
the packet is valid. Everything else means packet was wrong and it get discarded.
 */
uint32_t getRXFrameData(uint16_t current_counter)
{
    uint64_t data = received_data.value;
    uint16_t nbits = received_data.bits;
    91
 uint8_t destID = 0;
    uint8_t packet_type = 0;
    uint32_t payload = 0;
    uint8_t is_from_controller = 1;
    if (nbits == 16)
    { // Short Frame -> TYPE ACK, NACK, BYE
        packet_type = (uint8_t)((data >> 13) & 0b111); // 3 bits
        payload = (uint32_t)(data & 0xFFF); // 12 bits
        is_from_controller = (uint8_t)((data >> 12) & 0b1); // 1 bit
    }
    else if (nbits == 36)
    { // Long Frame -> TYPE HELLO, DATA, DATA_END
        packet_type = (uint8_t)((data >> 33) & 0b111); // Located in same place for all
36-bit
        frames
if (packet_type == TYPE_HELLO || packet_type == TYPE_HELLO_ENCRYPT)
{
    payload = (uint32_t)(data & 0x00FFFFFF); // 24 bits
    is_from_controller = (uint8_t)((data >> 24) & 0b1);
    destID = (uint8_t)(data >> 25); //DestID only present in HELLO seqs
}
else if (packet_type == TYPE_HELLO_NEW)
{
    payload = (uint32_t)(data & 0x00FFFFFF); // 24 bits
    is_from_controller = (uint8_t)((data >> 24) & 0b1);
```

```
    destID = (uint8_t)(data >> 25); //DestID only present in HELLO seqs
}
else if (packet_type == TYPE_DATA || packet_type == TYPE_DATA_END ||
packet_type == TYPE_DATA_END_ENCRYPT)
{
    payload = (uint32_t)(data & 0xFFFFFFFF); // 32 bits
    is_from_controller = (uint8_t)((data >> 32) & 0b1);
}
 }
 if (is_from_controller) { Serial.println("\n**a** (not coming from a node)"); return -1;
}
uint16_t next_counter = current_counter + 1;
uint32_t response_ok = packet_type | (((uint32_t)(next_counter)) << 16);
uint32_t response_ko = packet_type;
//Serial.print("DEBUG: responseOK: ");
//serialPrintUint64(response_ok, BIN);
//Serial.println("\n\n");
switch (packet_type)
{
    case TYPE_ACK:
        //Serial.println("*Got ACK*");
        //Serial.printf("Payload: %d, current_counter: %d. Are equal: ", payload,
        current_counter);
        //Serial.println((uint16_t) (payload)==current_counter);
        if ((uint16_t)(payload) == current_counter)
        { //ACKs should contain n° of packets
            received
        Serial.printf("Got ACK number %d\n", current_counter);
            //printRXPacketInfo(destID, packet_type, is_from_controller, payload);
            return response_ok;
        }
        return response_ko;
        break;
    case TYPE_NACK:
        return response_ko;
        break;
    case TYPE_HELLO_ENCRYPT:
        if (FLAG_HELLO_DONE)
            return response_ko;
        if (devID != destID)
            return response_ko;

        92
 Serial.println("Got Hello Encrypt!");
        //printRXPacketInfo(destID, packet_type, is_from_controller, payload);
        FLAG_ENCRYPT_DATA = 1;
        return response_ok;
        break;
    case TYPE_HELLO:
        if (FLAG_HELLO_DONE)
            return response_ko;
        if (devID != destID)
            return response_ko;

        Serial.println("Got Hello!");
        //printRXPacketInfo(destID, packet_type, is_from_controller, payload);
        FLAG_ENCRYPT_DATA = 0;
        return response_ok;
        break;
    case TYPE_HELLO_NEW:
        if (FLAG_HELLO_DONE)
            return response_ko;
        if (payload == 0)
            return response_ko;
```

```
        Serial.println("Got Hello_NEW!");
        if (destID == devID)
            return response_ok;

        return response_ko;
        break;
    case TYPE_DATA:
        if (current_counter > 0 && rx_data_buffer[current_counter - 1] == payload &&
payload > 0)
            return response_ko;
        //Serial.printf("Got Data! counter: %d\n", current_counter);
        //printRXPacketInfo(destID, packet_type, is_from_controller, payload);
        rx_data_buffer[current_counter] = payload;
        return response_ok;
        break;
    case TYPE_DATA_END:
        if (current_counter == 0)
            return response_ko;
        //Serial.printf("Got *LAST* Data! counter: %d\n", current_counter);
        if (checkDataIntegrity(current_counter, payload))
        {
            FLAG_ENCRYPT_DATA = 0;
            return response_ok;
        }
        return response_ko;
        break;
    case TYPE_DATA_END_ENCRYPT:
        if (current_counter == 0)
            return response_ko;
        //Serial.printf("Got *LAST* Data! counter: %d\n", current_counter);
        if (checkDataIntegrity(current_counter, payload))
        {
            FLAG_ENCRYPT_DATA = 1;
            return response_ok;
        }
        return response_ko;
        break;
    default:
        return response_ko;
        93
 break;
}
}
/**
 * Function which encrypts the data before sending it over IR. It uses a 32-byte
shared_key
to encrypt 32 bytes of information.\n
 * As the key is unique in each session, there is no need to use a Init_Vector, as in
this
case:\n
 * len(data)==len(key), we can use a simple XOR bit comparison to encrypt/decrypt the
Data.
 * @param[in] byte *shared_key: Pointer to the session key used to encrypt/decrypt the
data.
 * @param[in] byte *msg: Pointer to the message to be encrypted.
 * @param[in] bool hasPubKey: Boolean which, if true, allow to append the publickey in
clear to the message. Default false.
 * @param[out] byte *encrypted: Pointer to where the encrypted result will be saved.
*/
void encryptData(byte* shared_key, byte* msg, byte* encrypted, bool hasPubKey = false)
{
    for (int i = 0; i < SSID_MAX_SIZE; i++)
    {
        encrypted[i] = msg[i] ^ shared_key[i];
```

```
        encrypted[i + SSID_MAX_SIZE] = msg[i + SSID_MAX_SIZE] ^ shared_key[i +
SSID_MAX_SIZE];
        if (hasPubKey)
        {
            encrypted[i + PSK_MAX_SIZE] = msg[i + PSK_MAX_SIZE];
            encrypted[i + PSK_MAX_SIZE + SSID_MAX_SIZE] = msg[i + PSK_MAX_SIZE +
SSID_MAX_SIZE];
        }
    }
}
/**
 * Function which decrypts the data received over IR. It uses a 32-byte shared_key to
encrypt 32 bytes of information.\n
 * As the key is unique in each session, there is no need to use a Init_Vector, as in
this
case:\n
 * len(data)==len(key), we can use a simple XOR bit comparison to encrypt/decrypt the
Data.
 * @param[in] byte *shared_key: Pointer to the session key used to encrypt/decrypt the
data.
 * @param[in] byte *encrypted: Pointer to the message to be decrypted.
 * @param[out] byte *decrypted: Pointer to where the decrypted result will be saved.
*/
void decryptData(byte* shared_key, byte* encrypted, byte* decrypted)
{
    for (int i = 0; i < SSID_MAX_SIZE; i++)
    {
        decrypted[i] = encrypted[i] ^ shared_key[i];
        decrypted[i + SSID_MAX_SIZE] = encrypted[i + SSID_MAX_SIZE] ^ shared_key[i +
SSID_MAX_SIZE];
    }
}
/**
 * Function that translates data char array ("ssid_1234") (Between 16 and 32 bytes) to
chunks of 32bits.
 * @param[in] byte *char_data: Pointer to the data 8-bits array to be converted into a
24bits array.
 * @param[in] int size: size of the data in bytes.
 * @param[in] bool is_psk: Optimizes packet building in case the data is from the PSK, as
it uses only 4 bits for each character (50% efficiency otherwise). Default is false.
* @return uint16_t: integer representing the number of data packets the TX_buffer
contains.
 */
uint16_t buildTXDataBuffer(byte* char_data, int size, bool is_psk = false)
{
    uint16_t number_chunks = (size * 8) / 32;
    uint32_t checksum;

    // Casting byte int array into uint32 array.
    // 1 uint32 will hold 4 bytes, as payload is 32bits.
    Serial.println("DEBUG: buildTXDataBuffer [HEX]:");

    if (is_psk)
    {
        for (int i = 0; i < 32; i++)
        {
            Serial.printf("%c", TO_HEX(char_data[i]));
            94
        }
        Serial.println();
        number_chunks = number_chunks / 2;
        for (int i = 0; i < number_chunks; i++)
        {
            tx_data_buffer[i] =
```

```
            static_cast<uint32_t>(char_data[8 * i]) << 0 |
            static_cast<uint32_t>(char_data[8 * i + 1]) << 4 |
            static_cast<uint32_t>(char_data[8 * i + 2]) << 8 |
            static_cast<uint32_t>(char_data[8 * i + 3]) << 12 |
            static_cast<uint32_t>(char_data[8 * i + 4]) << 16 |
            static_cast<uint32_t>(char_data[8 * i + 5]) << 20 |
            static_cast<uint32_t>(char_data[8 * i + 6]) << 24 |
            static_cast<uint32_t>(char_data[8 * i + 7]) << 28;
            serialPrintUint64(tx_data_buffer[i], HEX);
            Serial.println();
        }

    }
    else
    {
        for (int i = 0; i < number_chunks; i++)
        {
            tx_data_buffer[i] =
            static_cast<uint32_t>(char_data[4 * i]) << 0 |
            static_cast<uint32_t>(char_data[4 * i + 1]) << 8 |
            static_cast<uint32_t>(char_data[4 * i + 2]) << 16 |
            static_cast<uint32_t>(char_data[4 * i + 3]) << 24;
            //serialPrintUint64(tx_data_buffer[i], HEX);
            //Serial.println();
        }
    }

    // Avoid sending many 0's packet at the end of data.
    for (int i = 0; i < number_chunks; i += 0)
    {
        if (tx_data_buffer[i] == 0)
        {
            for (int j = i; j < number_chunks - 1; j++)
            {
                tx_data_buffer[j] = tx_data_buffer[j + 1];
            }
            number_chunks -= 1;
        }
        else { i += 1; }
    }
    // Checksum calculation
    checksum = CRC32::calculate(tx_data_buffer, number_chunks);
    tx_data_buffer[number_chunks] = checksum;
    number_chunks += 1;
    Serial.printf("Num of data packets: %d\n\n", number_chunks);
    return number_chunks;
}
/**
 * Function that populates the transmission buffer with the PSK and SSID to be sent.\n
 * It also calculates the checksum of the sending data.
 * @param[in] byte *shared_key: Pointer to the session key shared with the addressee. If
not provided, data will not be encrypted. Default is NULL
 * @param[in] byte *mypublic_key: Pointer to the device public key that will be
transmitted. If not provided, burst will not include those extra 32 bytes. Default is
NULL
 * @return uint16_t: integer representing the number of data packets the TX_buffer
contains.
 */
uint16_t buildAllDataIntoBuffer(byte* shared_key = NULL, byte* mypublic_key = NULL)
{
    uint16_t num_bytes; // 1 PSK char is only 4 bits, we can get 2 chars in a Byte
    if (mypublic_key) { num_bytes = SSID_MAX_SIZE + PSK_MAX_SIZE / 2 + 32; } // 32 B is
pubkey length
    else { num_bytes = SSID_MAX_SIZE + PSK_MAX_SIZE / 2; }
```

```
    uint16_t total_packets = num_bytes / 4; //4 bytes per packet
    uint32_t checksum;
    byte msg[num_bytes];
    95
 Serial.println("DEBUG: buildAllDataIntoBuffer():");
for (int i = 0; i < SSID_MAX_SIZE; i++)
{ // msg[0], msg[1], ..., msg[31] = ssid[0], ssid[1],
    ..., psk[30] + psk[31]
 msg[i] = ssid[i];
    msg[i + 16] = psk[2 * i + 0] << 0 | (psk[2 * i + 1] << 4);
    if (mypublic_key)
    { // Loop goes from 0 to 15 only, we need to access 32 bytes in position
        0 - 31 and allocate them in position 32 - 63
    msg[i + 32] = mypublic_key[i];
        msg[i + 48] = mypublic_key[i + 16];
    }
}
if (shared_key)
{
    //Serial.println("MSG before encryption: ");
    //for(int i=0; i<num_bytes; i++) {
    // Serial.printf("Message i: %d = %d\n", i, msg[i]);
    //}
    if (mypublic_key)
    {
        encryptData(shared_key, msg, msg, true); //result is returned into msg variable
    }
    else
    {
        encryptData(shared_key, msg, msg, false);
    }
    //Serial.println("MSG after encryption: ");
    //for(int i=0; i<num_bytes; i++) {
    // Serial.printf("EncrMessage i: %d = %d\n", i, msg[i]);
    //}
}

for (int i = 0; i < total_packets; i++)
{
    tx_data_buffer[i] =
    static_cast<uint32_t>(msg[4 * i]) |
    static_cast<uint32_t>(msg[4 * i + 1]) << 8 |
    static_cast<uint32_t>(msg[4 * i + 2]) << 16 |
    static_cast<uint32_t>(msg[4 * i + 3]) << 24;
    serialPrintUint64(tx_data_buffer[i], HEX);
    Serial.println();
}

// Checksum calculation
checksum = CRC32::calculate(tx_data_buffer, total_packets);
tx_data_buffer[total_packets] = checksum;
total_packets += 1;
Serial.printf("Num of data packets: %d\n\n", total_packets);
return total_packets;
}
/**
 * Core Function which establishes a connection with a Node, transfers the Wi-Fi
credentials
and closes the connection.\n
 * It is called periodically, until receiving a confirmation with Destination Node that
info
was correctly received.\n
 * It includes several connection stages:\n
 * HELLO_NEW: Handshake with a random proof for new devices, which allows concurrence
```

```
problems with multiple new devices answering the controller at the same time.\n
 * HELLO: Normal handshake, performed for all devices. it also informs the node if the
transmission will be encrypted or not.\n
 * KEY_EXCHANGE: If users selects Encryption, there is a key exchange stage where both
nodes send each other their public keys.\n
 * DATA_EXCHANGE: Sends the data (in clear or encrypted) until receiving final ACK from
node.
 * @return bool: True if data could be transmitted. False otherwise.
96
 */
bool contactWithNode()
{
    FLAG_HELLO_DONE = 0;
    long ref_time = millis();
    long time0 = millis();
    bool HELLO_PENDING = true;
    bool NEW_HELLO_PENDING = true;
    uint32_t rand_seq_hello = 0;
    // Send Broadcast (dest_addr=0, TYPE_HELLO_NEW, new ID in payload)
    if (selected_devID == 0)
    {
        dest_devID = number_nodes + 1;
        uint8_t counter = 1;
        send_data.value = buildTXFrame(0, TYPE_HELLO_NEW, dest_devID);
        irsend.sendRC6(send_data.value, 36);
        irrecv.resume();
        Serial.print("*TIME: Broadcast NEW HELLO Sent! ");
        printTimeSeconds(ref_time);

        //Wait for Node HELLO Answer with Random Payload
        while (NEW_HELLO_PENDING)
        {
            delay(71);
            if (irrecv.decode(&received_data))
            {
                if (received_data.decode_type == RC6)
                {
                    uint32_t header_capture = getRXFrameData(0);
                    uint8_t received_type = (uint8_t)(header_capture);
                    uint16_t received_counter = (uint16_t)(header_capture >> 16);
                    if (received_type == TYPE_HELLO_NEW && received_counter == 1)
                    {
                        uint32_t payload = (uint32_t)(received_data.value & 0x00FFFFFF);
// 24 bits;
                        NEW_HELLO_PENDING = false;
                        rand_seq_hello = payload;
                        Serial.printf("Received Hello with a Random Payload: %d\n",
rand_seq_hello);
                    }
                }
                if (NEW_HELLO_PENDING && counter % 3 != 0)
                    irrecv.resume();
            }
            if (NEW_HELLO_PENDING && counter % 3 == 0)
            {
                delay(33);
                irsend.sendRC6(send_data.value, 36);
                irrecv.resume();
                Serial.println("\nSent new Broadcast HELLO!");
            }

            counter++;
        }
        Serial.print("*TIME: Broadcast NEW HELLO Answer Received! ");
```

```
        printTimeSeconds(ref_time);
        printTimeSeconds(time0);
        if (verbose_display)
        {
            display.clearDisplay();
            display.display();
            display.setTextSize(1);
            display.setTextColor(WHITE);
            display.setCursor(0, 0);
            display.println(F("Communication Established"));
            display.println(F("New Hello Answer Received"));
            display.display();
        }
    }
    else
    { // selected_devID>0 (existing device)
        dest_devID = selected_devID;
        Serial.println(dest_devID);
        97
    rand_seq_hello = selected_devID;
    }
    // *** HELLO FOR A KNOWN DEVICE STARTS HERE ***
    // In case dest_devID was not known, send back hello with randomGen payload
(rand_seq=0 if
    dev is known)
 if (FLAG_ENCRYPT_DATA)
{
    send_data.value = buildTXFrame(dest_devID, TYPE_HELLO_ENCRYPT,
rand_seq_hello);
}
else { send_data.value = buildTXFrame(dest_devID, TYPE_HELLO, rand_seq_hello); }

irsend.sendRC6(send_data.value, 36);
irrecv.resume();
Serial.print("*TIME: HELLO for Known DevID SENT! ");
printTimeSeconds(ref_time);

time0 = millis();
uint8_t counter = 1;

//Wait for Node HELLO Answer
while (HELLO_PENDING)
{
    delay(71);
    if (irrecv.decode(&received_data))
    {
        if (received_data.decode_type == RC6)
        {
            uint32_t header_capture = getRXFrameData(0);
            uint8_t received_type = (uint8_t)(header_capture);
            uint16_t received_counter = (uint16_t)(header_capture >> 16);
            if ((received_type == TYPE_HELLO || received_type == TYPE_HELLO_ENCRYPT) &&
           received_counter == 1)
            {
                uint8_t payload = (uint8_t)received_data.value;
                if (payload == dest_devID)
                {
                    HELLO_PENDING = false;
                    Serial.println("Received Hello for Known DevID Answer!");
                }
            }
        }
        if (HELLO_PENDING && counter % 3 != 0)
            irrecv.resume();
```

```
    }

    if (HELLO_PENDING && counter % 3 == 0)
    {
        delay(33);
        irsend.sendRC6(send_data.value, 36);
        irrecv.resume();
        Serial.println("\n Sent another HELLO for Known DevID!");
    }
    counter++;
    if (HELLO_PENDING && millis() - time0 > 5000)
        return false;
}
if (!FLAG_ENCRYPT_DATA)
{
    Serial.print("*TIME: HELLO Answer Received! ");
    printTimeSeconds(ref_time);
    printTimeSeconds(time0);
    if (millis() - time0 < 600)
        delay(610 - (millis() - time0)); // Wait for receiver to stop sending hellos,
before
    sending us
 }
FLAG_HELLO_DONE = 1; // Flag=1 means we are ready to store data packets (in case they
arrive)
 Serial.print("*TIME: HELLO FINAL answer received! ");
printTimeSeconds(ref_time);
98
 printTimeSeconds(time0);
if (verbose_display)
{
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println(F("Hello Final Answer Received"));
    if (FLAG_ENCRYPT_DATA)
        display.println(F("Sending PublicKey for Encryption..."));
    else { display.println(F("Sending Data in Clear...")); }
    display.display();
}
uint8_t packet_max_number;
// ***************** HELLO DONE, Exchanging Curve PublicKeys *****************
if (FLAG_ENCRYPT_DATA)
{
    static const uint8_t basepoint[32] = { 9 };
    uint8_t mycurve_secret[32];
    uint8_t mycurve_public[32];
    uint8_t hiscurve_public[32];
    uint8_t mycurve_shared[32];
    for (int i = 0; i < 32; i++) { mycurve_secret[i] = ESP8266TrueRandom.randomByte(); }
    curve25519_donna(mycurve_public, mycurve_secret, basepoint);
    //for(int i=0; i<32; i++) { Serial.printf("Mycurve_public Byte %d: %d\n", i,
    mycurve_public[i]);
}
// *** Waiting for Device's PublicKey Reception ***
bool IS_LAST_DATA = false;
bool CHECKSUM_ERROR;
time0 = millis();
uint16_t packet_sequence;
// Loop for Device's PublicKey Reception
while (!IS_LAST_DATA)
{
```

```
    CHECKSUM_ERROR = false;
    packet_sequence = 0;
    irrecv.resume();
    if (millis() - time0 > 1500)
        return false;
    while (!IS_LAST_DATA && !CHECKSUM_ERROR)
    {
        delay(2);
        if (irrecv.decode(&received_data))
        {
            if (received_data.decode_type == RC6)
            {
                uint32_t header_capture = getRXFrameData(packet_sequence);
                uint8_t received_type = (uint8_t)(header_capture);
                uint16_t received_counter = (uint16_t)(header_capture >> 16);
                if (received_type == TYPE_DATA && received_counter == packet_sequence +
1)
                {
                    Serial.printf("PublicKey Data %d received.\n", packet_sequence);
                    packet_sequence++;
                }
                if (received_type == TYPE_DATA_END_ENCRYPT)
                {
                    if (received_counter == packet_sequence + 1)
                    {
                        Serial.printf("**LAST** PublicKey Packet %d received.\n",
packet_sequence);
                        IS_LAST_DATA = true;
                        99
                    packet_sequence++;
                    }
                    else
                    {
                        Serial.println("Checksum ERROR"); memset(&rx_data_buffer, 0,
                   packet_sequence); CHECKSUM_ERROR = true;
                    }
                }
            }
            if (!IS_LAST_DATA)
                irrecv.resume();
        }
        if (packet_sequence >= 9)
            CHECKSUM_ERROR = true;
    }
}
Serial.print("*TIME: Device's PublicKey Received");
printTimeSeconds(ref_time);
printTimeSeconds(time0);
Serial.println("Sending DATA ACK");
send_data.value = buildTXFrame(selected_devID, TYPE_ACK, packet_sequence);
time0 = millis();
// ACK for device's PublicKey
for (int i = 0; i < 4; i++)
{
    delay(5);
    irsend.sendRC6(send_data.value, 16); //ACK
}
//while(millis()-time0 < 180) {
// irsend.sendRC6(send_data.value, 16);
// delay(15);
//}
Serial.print("*TIME: ACK for PublicKey Sent - Pre-Saving Data");
printTimeSeconds(ref_time);
printTimeSeconds(time0);
```

```
for (int i = 0; i < packet_sequence - 1; i++)
{ // PublicKey is 32bytes, there are 4bytes per
packet: packet_seq = 8
 hiscurve_public[4 * i + 0] = (uint8_t)(rx_data_buffer[i] >> 0 * 8);
    hiscurve_public[4 * i + 1] = (uint8_t)(rx_data_buffer[i] >> 1 * 8);
    hiscurve_public[4 * i + 2] = (uint8_t)(rx_data_buffer[i] >> 2 * 8);
    hiscurve_public[4 * i + 3] = (uint8_t)(rx_data_buffer[i] >> 3 * 8);
    //Serial.printf("Buffer HisPublicKey %d: %d | %d | %d | %d\n", i,
    hiscurve_public[4 * i + 0], hiscurve_public[4 * i + 1], hiscurve_public[4 * i + 2],
hiscurve_public[4 * i + 3]);
 //Serial.println();
 }
 memset(&rx_data_buffer, 0, packet_sequence);
curve25519_donna(mycurve_shared, mycurve_secret, hiscurve_public);
//for(int i=0; i<32; i++) { Serial.printf("Mycurve_shared Byte %d: %d\n", i,
mycurve_shared[i]); }
 Serial.println("Entering Encrypting function...");
packet_max_number = buildAllDataIntoBuffer(mycurve_shared, mycurve_public);
if (verbose_display)
{
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println(F("PublicKey Received & SharedSecret calculated!"));
    display.println(F("Sending Encrypted Data..."));
    display.display();
}
 }
 else { packet_max_number = buildAllDataIntoBuffer(); }
// ***************** Curve Public Key Exchange Done, Data TX Starts Here
*****************
bool ACK_PENDING = true;
time0 = millis();
while (ACK_PENDING)
{
    Serial.println("\nEntering Sending DATA Loop...");
    for (int i = 0; i < packet_max_number; i++)
    {
        delay(5);
        if (millis() - time0 > 2500)
        {
            break;
        }
        if (i + 1 == packet_max_number)
        {
            if (FLAG_ENCRYPT_DATA)
            {
                send_data.value = buildTXFrame(selected_devID,
           TYPE_DATA_END_ENCRYPT, tx_data_buffer[i]);
            }
            else
            {
                send_data.value = buildTXFrame(selected_devID, TYPE_DATA_END,
           tx_data_buffer[i]);
            }
        }
        else
        {
            send_data.value = buildTXFrame(selected_devID, TYPE_DATA,
       tx_data_buffer[i]);
        }
```

```
        Serial.printf("Packet number %d\n", i);
        irsend.sendRC6(send_data.value, 36);
    }
    Serial.print("*TIME: All info sent");
    printTimeSeconds(ref_time);
    printTimeSeconds(time0);
    irrecv.resume();
    Serial.println("Waiting for ACK");
    time0 = millis();
    uint8_t n_attempts = 0; // Return after 3 missing acks

    while (ACK_PENDING)
    {
        delay(2);
        if (irrecv.decode(&received_data))
        {
            if (received_data.decode_type == RC6)
            {
                uint32_t header_capture = getRXFrameData(packet_max_number);
                uint8_t received_type = (uint8_t)(header_capture);
                uint16_t received_counter = (uint16_t)(header_capture >> 16);
                if (received_type == TYPE_ACK && received_counter == packet_max_number +
1)
                {
                    ACK_PENDING = false;
                    Serial.println("\n\n**Burst ACK Received for all DATA**\n");
                }
            }
            if (ACK_PENDING)
                irrecv.resume();
        }
        if (millis() - time0 > 300)
            break;
    }
    if (ACK_PENDING)
    {
        n_attempts += 1;
        if (n_attempts >= 3)
            return false;
    }
}
101
 Serial.print("*TIME: DATA ACK received ");
printTimeSeconds(ref_time);
printTimeSeconds(time0);
if (verbose_display)
{
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println(F("Data Correctly Sent!"));
    display.println(F("ACK Final Received. Printing QRCode..."));
    display.display();
}
wait(0);
// QR code Printing
long_press = false;
while (!long_press)
{
    displayOLEDQRCode();
}
if (selected_devID == 0)
```

```
{
    Serial.println("Saving New NodeID into Memory:");
    number_nodes += 1;
    unsigned char array_number_nodes[1];
    array_number_nodes[0] = number_nodes;
    writeEEPROM(ADDR_NUM_NODES, array_number_nodes, 1);
    unsigned char array_dest_devID[1];
    array_dest_devID[0] = dest_devID;
    writeEEPROM(ADDR_NODE_1 + number_nodes - 1, array_dest_devID, 1);
}
return true;
}
/**@}*/
/**
* \defgroup main_functions 2.3. Program Main Functions
* Functions which must be implemented by an ESP8266. They execute the main logic of the
program with the aid of the previous functions
* @{
*/
/// Setup function which checks EEPROM, generate random <SSID,PSK> if not initialized,
waits
for user input in destID selection and Encryption menus, communicates with node, and
prints
the QR_Code.
void setup() {
    Serial.begin(115200);
    while (!Serial)
        delay(50);
    Serial.println();

    time_t t;
    srand((unsigned)time(&t));
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), handleButtonInterrupt, CHANGE);
    //Uncomment this to reset memory by removing 'Y' control char
    /* unsigned char a[2] = {'A', '\0'};
    * writeEEPROM(ADDR_CONTROL_CHAR, a, 1);
    */
    102
 readEEPROM(ADDR_CONTROL_CHAR, 2, eeprom_control_char);
    Serial.println("Memory before loading EEPROM:");
    printEEPROMData();
    // Control Char not found, init EEPROM (random <ssid,psk>, num_nodes=1, devID_1=1,
    control_char = 1)
 if (!(eeprom_control_char[0] == CONTROL_EEPROM_TARGET_CHAR[0]))
    setupEmptyEEPROM();
loadConfigValuesFromEEPROM();
Serial.println("Memory AFTER loading EEPROM:");
printEEPROMData();
// ********************************************
Serial.println("Select devID (ShortPush)\n &\nConfirm (Long Push)\n");
// number_nodes = 6;
// unsigned char aux_array[6] = {1, 2, 3, 4, 5, 6};
// strncpy((char *) &nodes_array, (char *) &aux_array, number_nodes);

for (int i = 0; i < number_nodes; i++)
{
    Serial.print(nodes_array[i]);
    Serial.print(", ");
}
Serial.println();
// SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
{ // Address 0x3C for 128x32
```

```
    Serial.println(F("SSD1306 allocation failed"));
}
// QR code Printing
long_press = false;
while (!long_press)
{
    displayOLEDQRCode();
}
display.display();
displayDeviceIDSelection();
devID_confirmed = true;
while (!devID_confirmed)
{
    if (screen_needs_refresh)
    {
        screen_needs_refresh = false;
        displayDeviceIDSelection();
    }
    delay(100);
}
Serial.println("Want to Encrypt Transmission?");
displayEncryptedCommSelection();
encrypt_confirmed = true;
while (!encrypt_confirmed)
{
    if (screen_needs_refresh)
    {
        screen_needs_refresh = false;
        displayEncryptedCommSelection();
    }
    delay(100);
}
display.clearDisplay();
display.display();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0, 0);
display.println(F("Selection Saved!"));
display.print(F("Device Selected: "));
if (selected_devID == 0)
{
    display.println(F("NEW."));
}
else
{
    display.print(selected_devID);
    display.println(F("."));
}
display.println(F("Trying to Communicate"));
display.println(F("Point towards it..."));
display.display();
// ****************************************
irrecv.enableIRIn(); // Getting IR Receiver Ready
irsend.begin(); // Getting IR Transmitter Ready
int num_attempts = 1;
Serial.print("Contacting with the node. Current Attempt: ");
Serial.println(num_attempts);
while (!contactWithNode())
{
    num_attempts += 1;
    Serial.print("Contacting with the node. Current Attempt: ");
    Serial.println(num_attempts);
}
Serial.printf("\nInfo Correctly Sent to Node ID: %d!\n", dest_devID);
```

```
}
/// Loop() function not used. Just sleep.
void loop()
{
    delay(50);
}
/**@}*/
/**@}*/
/**@}*/
```

ДОДАТОК Г

Код для ESP8266 Receiver Code

```
/**
 * \defgroup main main.cpp TFM Receiver Code
 * 2019 - Rafael de la Rosa Ortiz \n
 * Allows a ESP8266 Node to receive a <SSID,PK> from an IoT Controller.\n
 * It checks if device has EEPROM initialized. If not, it will start listening for a
controller until receiving <SSID,PK>\n
 * It receives the Key from the controller securely by implementing a Diffie Hellman Key
Exchange Protocol.\n
 * Once information is received correctly, it saves it in its EEPROM and tries to connect
to
the Access Point with the credentials.
 * @{
 */
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>
#include <IRsend.h>
#include <EEPROM.h>
#include <CRC32.h>
#include <ESP8266TrueRandom.h>
#include <curve25519-donna.h>
#include <string.h>
#include <math.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
// ****************************************************************
/**
* \defgroup global_variables 1. Global Variables Definition
* @{
*/
/**
* \defgroup eeprom_control_variables 1.1. Variables to Control EEPROM Status and Content
* @{
*/
/// Node Device ID - If 0, device has no ID assigned (it will start listening for a
controller).
unsigned char devID = 0;
/// Auxiliar device ID. It is temporal during the communication. Will be copied to devID
if
communication ends successfully.
unsigned char devID_aux = 0;
/// Controller Device ID=1 (hardcoded). Nodes can only communicate with the controller.
const unsigned char CONTROLLER_ID = 1;
/// EEPROM Control Char target, 'Y'.
const unsigned char CONTROL_EEPROM_TARGET_CHAR[2] = { 'Y', '\0'};
/// EEPROM Read Control Char is stored here. If equal to 'Y', memory was written before.
unsigned char eeprom_control_char[2];
/// Max Number of Nodes is 206 (Limited by EEPROM).
#define NUM_NODES_MAX 206
/// Array of Nodes. +1 to store empty char '\0' at last position.
unsigned char nodes_array[NUM_NODES_MAX + 1];
105
/// Number of Nodes. Default is 1. Value may change after reading EEPROM.
unsigned char number_nodes = 1;
/**@}*/
/**
```

```
* \defgroup eeprom_address_map 1.2. EEPROM ADDRESS MAP
* It uses 64 addresses for an IoT node. Each address can store 8 bits.
* @{
*/
/// 0x00, ssid size = 16 B.
#define ADDR_SSID 0
/// 0x10, psk size = 32 B.
#define ADDR_PSK 16
/// 0x30, n_nod size = 1 B.
#define ADDR_DEV_ID 48
/// 0x3F, control character. if char=='Y', memory has data.
#define ADDR_CONTROL_CHAR 63
/**@}*/
/**
* \defgroup wifi_credentials 1.3. <SSID,PSK> global variables
* @{
*/
/// SSID Max Size is 16 Bytes
#define SSID_MAX_SIZE 16
/// SSID Max Size is 32 Bytes
#define PSK_MAX_SIZE 32
/// Variables were EEPROM ssid data is stored. Initialized as empty "".
unsigned char ssid[SSID_MAX_SIZE + 1] = "";
/// Variables were EEPROM psk data is stored. Initialized as empty "".
unsigned char psk[PSK_MAX_SIZE + 1] = "";
/**@}*/
//Translates a byte (8 bits) to 2 HEX Chars (4+4 bits).
#define TO_HEX(i) ( (i) <= 9 ? '0' + (i) : 'A' - 10 + (i) )
////////////////////////////////////////////////////////////////
// Variables used in IR Send/Receive
////////////////////////////////////////////////////////////////
/**
* \defgroup ir_variables 1.4. Variables used in IR Send/Receive
* @{
*/
/// IR RX Pin D6 GPIO12 in ESP8266 nodeMCU.
const uint8_t kRecvPin = 12;
/// Set the GPIO to be used to receive data.
IRrecv irrecv(kRecvPin);
/// IR TX D7 GPIO13 in ESP8266 nodeMCU.
const uint8_t kIrLed = 13;
/// Set the GPIO to be used to send data.
IRsend irsend(kIrLed);
/// Variable where a received message is stored to.
decode_results received_data;
/// Variable where a message is stored before being sent.
decode_results send_data;
/// Transmission and Reception Buffers Size.
#define MAX_PACKET_SEQUENCE 1024
/// Reception data buffer of 32bits/packet.
uint32_t rx_data_buffer[MAX_PACKET_SEQUENCE];
/// Transmission Data buffer of 32bits/packet.
uint32_t tx_data_buffer[MAX_PACKET_SEQUENCE];
/**@}*/
106
////////////////////////////////////////////////////////////////
// Variables used in IR Frame Control
////////////////////////////////////////////////////////////////
/** \defgroup frame_packet_types 1.5. Frame Packet Types
 * Variables used in IR Frame Control:
 * 3 bits for type of packet.\n
 * 8 bits for dest deviceID.\n
 * 32 bits for data.\n
 * 16 bits for CRC16 Checksum.\n
```

```
 * Type of Packet:\n
 * 0, 000 -> HELLO_ENCRYPTED.\n
 * 1, 001 -> HELLO.\n
 * 2, 010 -> HELLO_NEW.\n
 * 3, 011 -> DATA.\n
 * 4, 100 -> ACK.\n
 * 5, 101 -> DATA_END (Last Data Packet -> Checksum).\n
 * 6, 110 -> DATA_END_ENCRYPTED (Last Data Packet (encrypted) -> Checksum).\n
 * 7, 111 -> NACK - Not Used.\n
 * @{
 */
#define TYPE_HELLO_ENCRYPT 0b000
#define TYPE_HELLO 0b001
#define TYPE_HELLO_NEW 0b010
#define TYPE_DATA 0b011
#define TYPE_ACK 0b100
#define TYPE_DATA_END 0b101
#define TYPE_DATA_END_ENCRYPT 0b110
#define TYPE_NACK 0b111 // Not Used right now
/**@}*/
/**
* \defgroup program_flags 1.6. FLAGS used by the program to distinguish between states
* @{
*/
/// Hello Communication Flag. 1 means HELLO Exchange was done, so don't listen for more
HELLO
Packets.
uint8_t FLAG_HELLO_DONE = 0;
/// Encryption Flag. 1 means Data is encrypted(DH Key Exchange). Default is 1.
uint8_t FLAG_ENCRYPT_DATA = 1;
/**@}*/
/**@}*/
// ********************************************************************
/**
* \defgroup functions 2. Functions
* Module with all the custom functions implemented in the controller source code
* @{
*/
/**
* \defgroup debug_functions 2.1. DEBUG Functions
* Functions which are used for debugging purposes (printing logs and traces mainly)
* @{
*/
/**
 * DEBUG Function which prints in terminal the time elapsed in seconds.
 * @param[in] long ref: Reference time where the counting period begins.
 */
void printTimeSeconds(long ref)
{
    double elapsed = (millis() - ref) / 1000.0;
    Serial.printf(" - Time Elapsed: %.4f seconds.\n", elapsed);
    107
}
/**
 * DEBUG function used to print in terminal information regarding received packets.
 * @param[in] uint32_t header_capture: integer that represents the 32bits captured in a
IR
frame.
 * @param[in] uint16_t received_counter: integer that represents the current packet
counter
in the data exchange.
 * @param[in] uint8_t received_type: integer that represents the type of packet that has
been received.
 */
```

```
void printCapturedSeq(uint32_t header_capture, uint16_t received_counter, uint8_t
received_type)
{
    Serial.println("\nDEBUG: printCapturedSeq() HEX:");
    Serial.print("Captured: "); Serial.println(header_capture, HEX);
    Serial.print("Counter: "); Serial.println(received_counter, HEX);
    Serial.print("packetType: "); Serial.println(received_type, HEX);
    Serial.println("*****\n");
}
/// DEBUG function used to print in terminal sent packet frame, represented in the global
variable send_data.
void printSentPacket() {
    Serial.println("printSentPacket() info:");
    //serialPrintUint64(send_data.value, BIN);
    //Serial.println("");
    serialPrintUint64(send_data.value, HEX);
    Serial.println("");
}
/// DEBUG function used to print in terminal received packet frame, represented in the
global
variable received_data.
void printReceivedPacket() {
    Serial.println("printReceivedPacket() info:");
    //serialPrintUint64(received_data.value, BIN);
    //Serial.println("");
    serialPrintUint64(received_data.value, HEX);
    Serial.println("");
}
/**
 * DEBUG function used to print in terminal information regarding getRXFrameData()
function
output.
 * @param[in] uint8_t destID: integer that represents frame's Device Destination ID
value.
 * @param[in] uint8_t packet_type: integer that represents the current packet's type.
 * @param[in] uint8_t is_from_controller: integer of which only the LS bit is used. 1
means
packet comes from controller device.
 * @param[in] uint32_t payload: integer representing the payload data of the frame.
 */
void printRXPacketInfo(uint8_t destID, uint8_t packet_type, uint8_t is_from_controller,
uint32_t payload)
{
    Serial.println("\nDEBUG: printRXPacketInfo()[BIN]: ");
    Serial.print("destID: "); Serial.println(destID, BIN);
    Serial.print("packetType: "); Serial.println(packet_type, BIN);
    Serial.print("isfromcontroller: "); Serial.println(is_from_controller, BIN);
    Serial.print("payload: "); Serial.println(payload, BIN);
    Serial.println("*****\n");
}
/// DEBUG function used to print stored EEPROM parameters: SSID, PSK, devID,
number_nodes...
void printEEPROMData()
{
    Serial.print("\nSSID: ");
    Serial.println((char*)ssid);
    Serial.print("PSK: ");
    Serial.println((char*)psk);
    Serial.print("DeviceID: ");
    Serial.println(devID);
Serial.print("Control char. target/actual: ");
Serial.print((char)CONTROL_EEPROM_TARGET_CHAR[0]);
Serial.print("/");
Serial.println((char)eeprom_control_char[0]);
```

```
Serial.println("");
}
/**@}*/
/**
* \defgroup aux_functions 2.2. Program Auxiliar Functions
* Functions which are called to perform important processes during the execution of the
main
functions
* @{
*/
/**
* Function which connects the device to the IoT Acess Point (a RaspPi IoT Hub).
* It assigns a static Ip Address to the device, based on his Device ID:
192.168.100.devID.
* AP Gateway is 192.168.100.1
*/
void connectStaticToAP()
{
    IPAddress staticIP(192, 168, 100, devID); // ESP static IP address
IPAddress gateway(192, 168, 100, 1); // IP Address of Gateway (RaspPi)
IPAddress subnet(255, 255, 255, 0); // Subnet mask
IPAddress dns(8, 8, 8, 8); // DNS (Google)
const char* deviceName = "IoT_Node_" + devID; // Descriptive device name
WiFi.hostname(deviceName);
char* psk_hex = (char*)malloc(PSK_MAX_SIZE + 1);
for (int i = 0; i < PSK_MAX_SIZE; i++)
{
    psk_hex[i] = TO_HEX(psk[i]);
}
psk_hex[PSK_MAX_SIZE] = '\0';
WiFi.disconnect();
WiFi.persistent(false);
WiFi.begin((char*)ssid, psk_hex);
WiFi.config(staticIP, gateway, subnet, dns); // Make config effective
Serial.println((char*)ssid);
Serial.println((char*)psk_hex);
WiFi.mode(WIFI_STA); // WiFi Station mode

// Wait for connection
Serial.print("Connecting to AP ");
Serial.print((char*)ssid);
while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}
Serial.println("\nConnected!");
}
/**
 * Function which delays execution a variable random amount of time.
 * @param[in] bool short_time: If true, the delay will be a short delay.
 */
void wait(bool short_time)
{
    if (short_time)
        delay(150 + (rand() % 10)); //Wait 150 to 160 ms
    else { delay(2000 + (rand() % 1000)); } // wait 2000 to 3000 ms
}
109
/**
 * Function used to simplify the process of reading a single entry stored in the EEPROM.
 * @param[in] uint8_t start_addr: integer that represents the initial address where the
information is stored.
```

```
 * @param[in] uint8_t num_bytes: integer that represents the number of bytes (or
addresses)
to read.
 * @param[out] unsigned char* result: pointer to where the read output should be stored.
 */
void readEEPROM(uint8_t start_addr, uint8_t num_bytes, unsigned char* result)
{
    EEPROM.begin(64);
    uint8_t index = 0;
    result[index] = (char)(EEPROM.read(start_addr + index));
    while (index < num_bytes - 1)
    {
        index = index + 1;
        result[index] = (char)(EEPROM.read(start_addr + index));
    }
    EEPROM.end();
}
/**
 * Function used to simplify the process of writing a single entry to the EEPROM.
 * @param[in] uint8_t start_addr: integer that represents the initial address where the
information is stored.
 * @param[in] unsigned char *data: pointer to where the read output should be stored.
 * @param[in] uint8_t size: integer that represents the number of bytes (or addresses) to
read.
 */
void writeEEPROM(uint8_t start_addr, unsigned char* data, uint8_t size)
{
    EEPROM.begin(64);
    uint8_t index = 0;
    EEPROM.write(start_addr + index, data[index]);
    while (index < size - 1)
    { //data[index] != '\0' &&
        index = index + 1;
        EEPROM.write(start_addr + index, data[index]);
    }
    EEPROM.commit();
    EEPROM.end();
}
/// Function used to initialize the EEPROM with the received information from controller:
< SSID,PSK > and NodeID.
void saveConfigEEPROM()
{
    Serial.println("Saving Received Data From Controller into EEPROM...");
    writeEEPROM(ADDR_SSID, ssid, SSID_MAX_SIZE);
    writeEEPROM(ADDR_PSK, psk, PSK_MAX_SIZE);
    // Conversion from int to char array - devID of controller (1)
    unsigned char devID_char[1];
    devID_char[0] = devID;
    writeEEPROM(ADDR_DEV_ID, devID_char, 1);
    // Write control char 'Y' in 0xFF to tell device's EEPROM has data
    strncpy((char*)&eeprom_control_char, (char*)&CONTROL_EEPROM_TARGET_CHAR, 2);
    writeEEPROM(ADDR_CONTROL_CHAR, eeprom_control_char, 1);
}
/// Function used to read all EEPROM configuration values and load them in the program
global
variables.
void loadConfigValuesFromEEPROM()
{
    // Read SSID,PSK
    readEEPROM(ADDR_SSID, SSID_MAX_SIZE, ssid);
    readEEPROM(ADDR_PSK, PSK_MAX_SIZE, psk);
    // Read Node DeviceID
    unsigned char devID_char[2];
    110
```

```
 readEEPROM(ADDR_DEV_ID, 2, devID_char);
    devID = devID_char[0];
    // Read Control Char
    unsigned char control_char[2];
    readEEPROM(ADDR_CONTROL_CHAR, 2, control_char);
    eeprom_control_char[0] = control_char[0];
}
/**
 * Function that performs the Data integrity Check for a burst of received packets.\n
 * It does a CRC32 check and determines if the data is valid.
 * @param[in] uint16_t size: Number of received packets in the RX_buffer.
 * @param[in] uint32_t rx_checksum: Checksum Received in the last data packet.
 * @return bool: True is data is valid. False if data is not valid.
 */
bool checkDataIntegrity(uint16_t size, uint32_t rx_checksum)
{
    uint32_t checksum = CRC32::calculate(rx_data_buffer, size);
    Serial.println("Checking Checksums...");
    Serial.printf("Received: %d | calculated: %d\n", rx_checksum, checksum);
    if (checksum == rx_checksum)
        return true;
    Serial.println("Checksums not matching!");
    return false;
}
/**
 * Function used to convert the info to send into a formatted frame ready to be sent.
 * @param[in] uint8_t destID: device to which the packet is sent.
 * @param[in] uint8_t packet_type: integer that represents the current packet's type.
 * @param[in] uint32_t payload: integer representing the payload data of the frame.
 * @return uint64_t: integer which represents the frame ready to be sent (up to 36 bits,
RC6 IR protocol).
 */
uint64_t buildTXFrame(uint8_t destID, uint8_t packet_type, uint32_t payload)
{
    uint64_t data = 0ULL;
    if (packet_type == TYPE_ACK || packet_type == TYPE_NACK)
    {
        data =
        static_cast<uint64_t>(payload & 0xFFF) << 0 |
        static_cast<uint64_t>(0ULL) << 12 | //Packet Sent From Node, always 0
        static_cast<uint64_t>(packet_type & 0b111) << 13;
    }
    else if (packet_type == TYPE_HELLO || packet_type == TYPE_HELLO_ENCRYPT ||
   packet_type == TYPE_HELLO_NEW)
    {
        data =
        static_cast<uint64_t>(payload) << 0 |
        static_cast<uint64_t>(0ULL) << 24 | //Packet Sent From Node, always 0
        static_cast<uint64_t>(destID) << 25 |
        static_cast<uint64_t>(packet_type) << 33;
    }
    else if (packet_type == TYPE_DATA || packet_type == TYPE_DATA_END_ENCRYPT ||
   packet_type == TYPE_DATA_END)
    {
        data =
        static_cast<uint64_t>(payload) << 0 |
        static_cast<uint64_t>(0ULL) << 32 | //Packet Sent From Node, always 0
        static_cast<uint64_t>(packet_type) << 33;
    }

    //Serial.println("DEBUG: buildTXFRAME:");
    //serialPrintUint64(data, HEX);
    //Serial.println();
    //serialPrintUint64(data, BIN);
```

```
    //Serial.println("\n");
    return data;
}
111
/**
 * Function which processes the received frame stored in received_data global variable.\n
* It processes the information of the frame and determines if the packet is valid.
 * @param[in] uint16_t current_counter: represents the number of received packets in the
current burst.
 * @return uint32_t: integer which contains 2 different variables:\n
 * Packet type (8bits) in the 8 LSBs.\n
 * New Counter (16bits) in the 16 MSBs. If NewCounter==current_counter_input+1, it means
the packet is valid. Everything else means packet was wrong and it get discarded.
 */
uint32_t getRXFrameData(uint16_t current_counter, uint32_t rand_hello_target = 0)
{
    uint64_t data = received_data.value;
    uint16_t nbits = received_data.bits;
    uint8_t destID = 0;
    uint8_t packet_type = 0;
    uint32_t payload = 0;
    uint8_t is_from_controller = 0;
    if (nbits == 16)
    { // Short Frame -> TYPE ACK
        packet_type = (uint8_t)((data >> 13) & 0b111); // 3 bits
        payload = (uint32_t)(data & 0xFFF); // 12 bits
        is_from_controller = (uint8_t)((data >> 12) & 0b1); // 1 bit
    }
    else if (nbits == 36)
    { // Long Frame -> TYPE HELLO, DATA, DATA_END
        packet_type = (uint8_t)((data >> 33) & 0b111); // Located in same place for all
36-bit
        frames
    if (packet_type == TYPE_HELLO || packet_type == TYPE_HELLO_ENCRYPT)
        {
            payload = (uint32_t)(data & 0x00FFFFFF); // 24 bits
            is_from_controller = (uint8_t)((data >> 24) & 0b1);
            destID = (uint8_t)(data >> 25); //DestID only present in HELLO seqs
        }
        else if (packet_type == TYPE_HELLO_NEW)
        {
            payload = (uint32_t)(data & 0x00FFFFFF); // 24 bits
            is_from_controller = (uint8_t)((data >> 24) & 0b1);
            destID = (uint8_t)(data >> 25); //DestID only present in HELLO seqs
        }
        else if (packet_type == TYPE_DATA || packet_type == TYPE_DATA_END ||
       packet_type == TYPE_DATA_END_ENCRYPT)
        {
            payload = (uint32_t)(data & 0xFFFFFFFF); // 32 bits
            is_from_controller = (uint8_t)((data >> 32) & 0b1);
        }
    }
    if (!is_from_controller) { Serial.println("\n**a** (not coming from controller)");
return -1; }
    uint16_t next_counter = current_counter + 1;
    uint32_t response_ok = packet_type | (((uint32_t)(next_counter)) << 16);
    uint32_t response_ko = packet_type;
    //Serial.print("DEBUG: responseOK: ");
    //serialPrintUint64(response_ok, BIN);
    //Serial.println("\n\n");

    switch (packet_type)
    {
        case TYPE_ACK:
```

```
            //Serial.println("*Got ACK*");
            //Serial.printf("Payload: %d, current_counter: %d. Are equal: ", payload,
            current_counter);
//Serial.println((uint16_t) (payload)==current_counter);
if ((uint16_t)(payload) == current_counter)
{ //ACKs should contain n° of packets
    received
 Serial.printf("Got ACK number %d\n", current_counter);
    //printRXPacketInfo(destID, packet_type, is_from_controller, payload);
    return response_ok;
}
112
 return response_ko;
break;
 case TYPE_NACK:
return response_ko;
break;
 case TYPE_HELLO_ENCRYPT:
if (FLAG_HELLO_DONE)
    return response_ko;
//printRXPacketInfo(destID, packet_type, is_from_controller, payload);
if (devID == 0 && devID_aux > 0)
{
    Serial.println("Got a HelloEncrypt after NEW HELLO... ");
    if (payload == rand_hello_target)
    {
        FLAG_ENCRYPT_DATA = 1;
        return response_ok;
    }
    return response_ko;
}
if (devID > 0)
{
    Serial.println("Got a Hello Encrypt... ");
    if (payload != devID)
    {
        Serial.printf("...but it is for another devID: %d\n", payload);
        return response_ko;
    }
    if (devID == destID)
    {
        devID_aux = devID;
        FLAG_ENCRYPT_DATA = 1;
        return response_ok;
    }
}
Serial.println("HELLO Encrypt not for me :(");
return response_ko;
break;
 case TYPE_HELLO:
if (FLAG_HELLO_DONE)
    return response_ko;
//printRXPacketInfo(destID, packet_type, is_from_controller, payload);
if (devID == 0 && devID_aux > 0)
{
    Serial.println("Got a Hello after NEW HELLO... ");
    if (payload == rand_hello_target)
    {
        FLAG_ENCRYPT_DATA = 0;
        return response_ok;
    }
    return response_ko;
}
if (devID > 0)
```

```
{
    Serial.println("Got a Hello... ");
    if (payload != devID)
    {
        Serial.printf("...but it is for another devID: %d\n", payload);
        return response_ko;
    }
    if (devID == destID)
    {
        devID_aux = devID;
        FLAG_ENCRYPT_DATA = 0;
        return response_ok;
    }
}
Serial.println("it is not for me :(");
return response_ko;
113
 break;
 case TYPE_HELLO_NEW:
if (FLAG_HELLO_DONE)
    return response_ko;
if (devID > 0)
    Serial.printf("Broadcast New_Hello Detected, but I have my own id: %d\n", devID);
if (destID > 0)
    Serial.printf("Broadcast New_Hello Detected, but destID is not 0: %d\n", destID);
if (payload == 0)
    Serial.printf("Broadcast New_Hello Detected, but payload does not contain a newID:
   % d\n", payload);


 if (devID > 0 || destID > 0 || payload == 0)
    return response_ko;

devID_aux = (uint8_t)payload;
Serial.print("\nNew devID assigned from HELLO_NEW: ");
Serial.println(devID_aux);
return response_ok;
break;
 case TYPE_DATA:
if (current_counter > 0 && rx_data_buffer[current_counter - 1] == payload && payload > 0)
    return response_ko;
//Serial.printf("Got Data! counter: %d\n", current_counter);
//printRXPacketInfo(destID, packet_type, is_from_controller, payload);
rx_data_buffer[current_counter] = payload;
return response_ok;
break;
 case TYPE_DATA_END:
if (current_counter == 0)
    return response_ko;
//Serial.printf("Got *LAST* Data! counter: %d\n", current_counter);
if (checkDataIntegrity(current_counter, payload))
{
    FLAG_ENCRYPT_DATA = 0;
    return response_ok;
}
return response_ko;
break;
 case TYPE_DATA_END_ENCRYPT:
if (current_counter == 0)
    return response_ko;
//Serial.printf("Got *LAST* Data! counter: %d\n", current_counter);
if (checkDataIntegrity(current_counter, payload))
{
    FLAG_ENCRYPT_DATA = 1;
```

```
    return response_ok;
}
return response_ko;
break;
default:
 return response_ko;
break;
 }
}
/**
 * Function which encrypts the data before sending it over IR. It uses a 32-byte
shared_key
to encrypt 32 bytes of information.\n
void encryptData(byte* shared_key, byte* msg, byte* encrypted)
{
    for (int i = 0; i < SSID_MAX_SIZE; i++)
    {
        encrypted[i] = msg[i] ^ shared_key[i];
        encrypted[i + SSID_MAX_SIZE] = msg[i + SSID_MAX_SIZE] ^ shared_key[i +
SSID_MAX_SIZE];
    }
}
/**
 * Function which decrypts the data received over IR. It uses a 32-byte shared_key to
encrypt 32 bytes of information.\n
 * As the key is unique in each session, there is no need to use a Init_Vector, as in
this
case:\n
 * len(data)==len(key), we can use a simple XOR bit comparison to encrypt/decrypt the
Data.
 * @param[in] byte *shared_key: Pointer to the session key used to encrypt/decrypt the
data.
 * @param[in] byte *encrypted: Pointer to the message to be decrypted.
 * @param[out] byte *decrypted: Pointer to where the decrypted result will be saved.
*/
void decryptData(byte* shared_key, byte* encrypted, byte* decrypted)
{
    for (int i = 0; i < SSID_MAX_SIZE; i++)
    {
        decrypted[i] = encrypted[i] ^ shared_key[i];
        decrypted[i + SSID_MAX_SIZE] = encrypted[i + SSID_MAX_SIZE] ^ shared_key[i +
SSID_MAX_SIZE];
    }
}
/**
 * Function that translates data char array ("ssid_1234") (Between 16 and 32 bytes) to
chunks of 32bits.
 * @param[in] byte *char_data: Pointer to the data 8-bits array to be converted into a
24bits array.
 * @param[in] int size: size of the data in bytes.
 * @param[in] bool is_psk: Optimizes packet building in case the data is from the PSK, as
it uses only 4 bits for each character (50% efficiency otherwise). Default is false.
 * @return uint16_t: integer representing the number of data packets the TX_buffer
contains.
 */
uint16_t buildTXDataBuffer(byte* char_data, int size)
{
    uint16_t number_chunks = ceil((size * 8) / 32.0);
    uint32_t checksum;

    // Casting byte int array into uint32 array.
    // 1 uint32 will hold 4 bytes, as payload is 32bits.
    Serial.println("DEBUG: buildTXDataBuffer [HEX]:");
```

```
    for (int i = 0; i < number_chunks; i++)
    {
        tx_data_buffer[i] =
        static_cast<uint32_t>(char_data[4 * i]) << 0 |
        static_cast<uint32_t>(char_data[4 * i + 1]) << 8 |
        static_cast<uint32_t>(char_data[4 * i + 2]) << 16 |
        static_cast<uint32_t>(char_data[4 * i + 3]) << 24;
        //serialPrintUint64(tx_data_buffer[i], HEX);
        //Serial.println();
    }

    // Avoid sending many 0's packet at the end of data.
    for (int i = 0; i < number_chunks; i += 0)
    {
        if (tx_data_buffer[i] == 0)
        {
            for (int j = i; j < number_chunks - 1; j++)
            {
                tx_data_buffer[j] = tx_data_buffer[j + 1];
            }
            115
        number_chunks -= 1;
        }
        else { i += 1; }
    }
    // Checksum calculation
    checksum = CRC32::calculate(tx_data_buffer, number_chunks);
    tx_data_buffer[number_chunks] = checksum;
    number_chunks += 1;
    //Serial.printf("Num of data packets: %d\n\n", number_chunks);
    return number_chunks;
}
/**
 * Core Function which listens for a connection from a Controller, receives the Wi-Fi
credentials and writes the data in the EEPROM.\n
 * It is called periodically, until correctly receiving the information from a
Controller.\n
 * It includes several connection stages:\n
 * HELLO_NEW: Handshake with a random proof for new devices, which allows concurrence
problems with multiple new devices answering the controller at the same time.\n
 * HELLO: Normal handshake, performed for all devices. it also informs the node if the
transmission will be encrypted or not.\n
 * KEY_EXCHANGE: If users selects Encryption in the Controller, there is a key exchange
stage where both nodes send each other their public keys.\n
 * DATA_EXCHANGE: Receives the data (in clear or encrypted) until receiving final ACK
from node.
 * @return bool: True if data was received. False otherwise.
 */
bool waitForControllerCommunication()
{
    Serial.println("DEBUG: waitForControllerCommunication2 begins - Waiting for HELLO
   packet.");

    long ref_time = millis();
    uint32_t rnd_rep = 0;
    devID_aux = devID;
    FLAG_HELLO_DONE = 0;

    bool HELLO_PENDING = true;
    bool NEW_HELLO_PENDING = true;
    uint16_t packet_sequence = 0;
    long time0 = millis();
    irrecv.resume();
    if (devID == 0)
```

```
    {
        Serial.println("*TIME: HELLO_NEW Starts");
        while (NEW_HELLO_PENDING)
        {
            ref_time = millis();
            delay(4);
            if (irrecv.decode(&received_data))
            {
                if (received_data.decode_type == RC6)
                {
                    uint32_t header_capture = getRXFrameData(packet_sequence);
                    uint8_t received_type = (uint8_t)(header_capture);
                    uint16_t received_counter = (uint16_t)(header_capture >> 16);
                    if (received_type == TYPE_HELLO_NEW && received_counter == 1)
                    {
                        NEW_HELLO_PENDING = false;
                        printTimeSeconds(ref_time);
                        Serial.println("HELLO_NEW from Controller Received!");
                        rnd_rep = ESP8266TrueRandom.random(1, 1 << 24) & 0x00FFFFFF;
                        Serial.printf("Rnd Payload Gen.: %d\n", rnd_rep);
                        send_data.value = buildTXFrame(CONTROLLER_ID, TYPE_HELLO_NEW,
rnd_rep);
                        irsend.sendRC6(send_data.value, 36);
                    }
                }
                116
            irrecv.resume();
            }
        }
        Serial.print("*TIME: HELLO_NEW Ends ");
        printTimeSeconds(ref_time);
    }
    time0 = millis();
    Serial.println("*TIME: HELLO Starts");
    while (HELLO_PENDING)
    {
        if (NEW_HELLO_PENDING) { ref_time = millis(); delay(30); }
        else { delay(123); }

        if (irrecv.decode(&received_data))
        {
            if (NEW_HELLO_PENDING)
                irrecv.resume();
            if (received_data.decode_type == RC6)
            {
                uint32_t header_capture = getRXFrameData(packet_sequence, rnd_rep);
                uint8_t received_type = (uint8_t)(header_capture);
                uint16_t received_counter = (uint16_t)(header_capture >> 16);
                if ((received_type == TYPE_HELLO || received_type == TYPE_HELLO_ENCRYPT)
&&
               received_counter == 1)
                {
                    HELLO_PENDING = false;
                    printTimeSeconds(ref_time);
                    Serial.println("HELLO from Controller Received!");
                    Serial.printf("REceived_type %d\n", received_type);
                    Serial.printf("Encrypt Flag: %d\n", FLAG_ENCRYPT_DATA);
                    if (FLAG_ENCRYPT_DATA)
                    {
                        send_data.value = buildTXFrame(CONTROLLER_ID,
                   TYPE_HELLO_ENCRYPT, devID_aux);
                    }
                    else
                    {
```

```
                        send_data.value = buildTXFrame(CONTROLLER_ID, TYPE_HELLO,
                    devID_aux);
                     }
                }
            }
        }
        if (!NEW_HELLO_PENDING)
        {
            delay(49);
            Serial.println("Sent NEW_Hello");
            irsend.sendRC6(send_data.value, 36);
            irrecv.resume();
        }
        if (!NEW_HELLO_PENDING && HELLO_PENDING && (millis() - time0 > 2500))
            return false;
    }
    Serial.print("*TIME: HELLO Received (while loop out) ");
    printTimeSeconds(ref_time);
    printTimeSeconds(time0);
    // HELLO MSG successfully received. Send Back a HELLO with our address as payload
    Serial.println("\nAnswering Controller with a HELLO Response");
    time0 = millis();

    for (int i = 0; i < 3; i++)
    {
        delay(55);
        irsend.sendRC6(send_data.value, 36);
    }
    //while(millis()-time0<270) {
    // delay(55);
    // irsend.sendRC6(send_data.value, 36);
    //}
    Serial.print("*TIME: HELLO Response Sent ");
    printTimeSeconds(ref_time);
    117
 printTimeSeconds(time0);
    FLAG_HELLO_DONE = 1;
    // ***************** HELLO DONE, Exchanging Curve PublicKeys *****************


    uint8_t mycurve_shared[32];
    uint8_t hiscurve_public[32];
    uint8_t mycurve_secret[32];
    if (FLAG_ENCRYPT_DATA)
    {
        Serial.println("Calculation Encrypting Keys...");

        // Generating all parameters for Elliptic Curve Exchange
        static const uint8_t basepoint[32] = { 9 };

        uint8_t mycurve_public[32];
        for (int i = 0; i < 32; i++) { mycurve_secret[i] =
ESP8266TrueRandom.randomByte(); }
        curve25519_donna(mycurve_public, mycurve_secret, basepoint);
        //for(int i=0; i<32; i++) { Serial.printf("Mycurve_public Byte %d: %d\n", i,
        mycurve_public[i]); }
 // *** Sending Device's PublicKey to Controller ***
 Serial.print("*TIME: Sending myPubKey ");
time0 = millis();
uint8_t packet_max_number = buildTXDataBuffer(mycurve_public, 32);
bool ACK_PENDING = true;
time0 = millis();
uint8_t n_attempts = 0; // Return after 3 missing acks
                        // Loop for Sending Device's PublicKey
```

```
while (ACK_PENDING)
{
    Serial.println("\nEntering Sending my PublicKey Loop...");
    for (int i = 0; i < packet_max_number; i++)
    {
        delay(5);
        if (i + 1 == packet_max_number)
        {
            if (FLAG_ENCRYPT_DATA)
            {
                send_data.value = buildTXFrame(CONTROLLER_ID,
           TYPE_DATA_END_ENCRYPT, tx_data_buffer[i]);
            }
            else
            {
                send_data.value = buildTXFrame(CONTROLLER_ID, TYPE_DATA_END,
           tx_data_buffer[i]);
            }
        }
        else
        {
            send_data.value = buildTXFrame(CONTROLLER_ID, TYPE_DATA,
       tx_data_buffer[i]);
        }

        Serial.printf("Packet number %d\n", i);
        irsend.sendRC6(send_data.value, 36);
    }
    Serial.print("*TIME: All PublicKey info sent");
    printTimeSeconds(ref_time);
    printTimeSeconds(time0);
    irrecv.resume();
    Serial.println("Waiting for ACK PublicKey");
    time0 = millis();

    118
 while (ACK_PENDING)
    {
        delay(2);
        if (irrecv.decode(&received_data))
        {
            if (received_data.decode_type == RC6)
            {
                uint32_t header_capture = getRXFrameData(packet_max_number);
                uint8_t received_type = (uint8_t)(header_capture);
                uint16_t received_counter = (uint16_t)(header_capture >> 16);
                if (received_type == TYPE_ACK && received_counter == packet_max_number +
1)
                {
                    ACK_PENDING = false;
                    Serial.println("\n\n**Burst ACK Received for PublicKey**\n");
                }
            }
            if (ACK_PENDING)
                irrecv.resume();
        }
        if (millis() - time0 > 300)
            break;
    }
    if (ACK_PENDING)
    {
        n_attempts += 1;
        if (n_attempts >= 4)
            return false;
```

```
    }
}
Serial.print("\n*TIME: myPubKey Sent | Waiting for Real Data ");
printTimeSeconds(ref_time);
printTimeSeconds(time0);
 }
 // ***************** Curve Public Key Exchange Done, Data RX Starts Here
*****************
 bool IS_LAST_DATA = false;
time0 = millis();
while (!IS_LAST_DATA)
{
    bool CHECKSUM_ERROR = false;
    packet_sequence = 0;
    irrecv.resume();
    if (millis() - time0 > 2000)
        return false;
    while (!IS_LAST_DATA && !CHECKSUM_ERROR)
    {
        delay(4);
        if (irrecv.decode(&received_data))
        {
            if (received_data.decode_type == RC6)
            {
                uint32_t header_capture = getRXFrameData(packet_sequence);
                uint8_t received_type = (uint8_t)(header_capture);
                uint16_t received_counter = (uint16_t)(header_capture >> 16);
                if (received_type == TYPE_HELLO && received_counter == packet_sequence +
1)
                {
                    send_data.value = buildTXFrame(CONTROLLER_ID, TYPE_HELLO, devID_aux);
                    delay(51);
                    irsend.sendRC6(send_data.value, 36);
                }
                if (received_type == TYPE_DATA && received_counter == packet_sequence +
1)
                {
                    Serial.printf("Data %d received.\n", packet_sequence);
                    packet_sequence++;
                }
                119
            if (received_type == TYPE_DATA_END || received_type == TYPE_DATA_END_ENCRYPT)
                {
                    if (received_counter == packet_sequence + 1)
                    {
                        Serial.printf("**LAST** Packet %d received.\n", packet_sequence);
                        IS_LAST_DATA = true;
                        packet_sequence++;
                        if (received_type == TYPE_DATA_END_ENCRYPT)
                        {
                            Serial.println("*** Data End
                       Encrypt!!"); }
                        }
                        else
                        {
                            Serial.println("Checksum ERROR"); memset(&rx_data_buffer, 0,
packet_sequence); CHECKSUM_ERROR = true;
                        }
                    }
                }
                if (!IS_LAST_DATA)
                    irrecv.resume();
            }
        }
```

```
    }
    Serial.print("*TIME: All DATA Received");
    printTimeSeconds(ref_time);
    printTimeSeconds(time0);
    Serial.println("Sending DATA ACK");
    send_data.value = buildTXFrame(CONTROLLER_ID, TYPE_ACK, packet_sequence);
    time0 = millis();
    for (int i = 0; i < 4; i++)
    {
        delay(5);
        irsend.sendRC6(send_data.value, 16); //ACK
    }
    //while(millis()-time0 < 180) {
    // irsend.sendRC6(send_data.value, 16); //ACK
    // delay(15);
    //}
    Serial.print("*TIME: ACK DATA Sent - Pre Saving Data");
    printTimeSeconds(ref_time);
    printTimeSeconds(time0);
    byte msg[32];
    for (int i = 0; i < 8; i++)
    { // 8 packets with info
        msg[4 * i + 0] = rx_data_buffer[i] >> 0 * 8;
        msg[4 * i + 1] = rx_data_buffer[i] >> 1 * 8;
        msg[4 * i + 2] = rx_data_buffer[i] >> 2 * 8;
        msg[4 * i + 3] = rx_data_buffer[i] >> 3 * 8;
        //Serial.printf("Buffer %d: ", i);
        //serialPrintUint64(rx_data_buffer[i], HEX);
        //Serial.println();
    }
    if (FLAG_ENCRYPT_DATA)
    {
        for (int i = 8; i < packet_sequence - 1; i++)
        { // PublicKey is 32bytes: packet_seq=8-15
            hiscurve_public[4 * (i - 8) + 0] = (uint8_t)(rx_data_buffer[i] >> 0 * 8);
            hiscurve_public[4 * (i - 8) + 1] = (uint8_t)(rx_data_buffer[i] >> 1 * 8);
            hiscurve_public[4 * (i - 8) + 2] = (uint8_t)(rx_data_buffer[i] >> 2 * 8);
            hiscurve_public[4 * (i - 8) + 3] = (uint8_t)(rx_data_buffer[i] >> 3 * 8);
            //Serial.printf("Buffer HisPublicKey %d: %d | %d | %d | %d\n", i,
hiscurve_public[4*(i8)+0], hiscurve_public[4*(i-8)+1], hiscurve_public[4*(i-8)+2],
hiscurve_public[4*(i-8)+3]);
            //Serial.println();
        }
        curve25519_donna(mycurve_shared, mycurve_secret, hiscurve_public);
        //for(int i=0; i<32; i++) { Serial.printf("Mycurve_shared Byte %d: %d\n", i,
        mycurve_shared[i]);
    }
    Serial.print("*TIME: Key Saved and Secret Calculated.");
    printTimeSeconds(ref_time);
    120
 printTimeSeconds(time0);
    Serial.println("Message was Encrypted! Decrypting...");
    decryptData(mycurve_shared, msg, msg);
}
for (int i = 0; i < SSID_MAX_SIZE; i++)
{
    ssid[i] = (unsigned char) (msg[i]);
psk[2 * i + 0] = (unsigned char) ((msg[i + SSID_MAX_SIZE] >> 0 * 4) & 0x0F);
psk[2 * i + 1] = (unsigned char) ((msg[i + SSID_MAX_SIZE] >> 1 * 4) & 0x0F);
 }
 Serial.print("*TIME: DATA Saved");
printTimeSeconds(ref_time);
printTimeSeconds(time0);
Serial.println("*** SSID Received ***");
```

```
Serial.println((char*)ssid);
Serial.println("*** PSK Received ***");
for (int i = 0; i < 32; i++) { Serial.printf("%c", TO_HEX(psk[i])); }
Serial.println('\n');
printTimeSeconds(ref_time);
Serial.println("All Data Received, saving all Information in EEPROM.");
devID = devID_aux;
saveConfigEEPROM();
Serial.println("Info in Variables After Reading EEPROM:");
loadConfigValuesFromEEPROM();
printEEPROMData();
Serial.println("Exiting Function waitForControllerCommunication()");
return true;
}
/**@}*/
// ********************************************************************
/**
* \defgroup main_functions 2.3. Program Main Functions
* Functions which must be implemented by an ESP8266. They execute the main logic of the
program with the aid of the previous functions
* @{
*/
/// Setup function which checks EEPROM and, if not initialized, starts listening for a
controller until receiving the information, writing it into EEPROM.
void setup() {
    Serial.begin(115200);

    while (!Serial) // Wait for the serial connection to be establised.
        delay(50);
    Serial.println();
    srand(time(NULL)); // Initialize Random Gen Seed
                       //Uncomment this to reset memory by removing 'Y' control char
                       //unsigned char a[2] = {'A', '\0'};
                       //unsigned char id[2] = {0, '\0'};
                       //writeEEPROM(ADDR_CONTROL_CHAR, a, 1);
                       //writeEEPROM(ADDR_DEV_ID, id, 1);
    readEEPROM(ADDR_CONTROL_CHAR, 2, eeprom_control_char);
    Serial.println("Memory before loading EEPROM:");
    121
 printEEPROMData();
    loadConfigValuesFromEEPROM();
    Serial.println("Memory after loading EEPROM:");
    printEEPROMData();

    // ********************************************************************
    irrecv.enableIRIn(); // Preparamos el receptor IR
    irsend.begin(); // Preparamos el emisor IR
    irrecv.resume();
    if (eeprom_control_char[0] != CONTROL_EEPROM_TARGET_CHAR[0] || devID == 0)
        Serial.println("\n\n Memory not Setup or devID not assigned, waiting for
controller
       connection");
    else
        Serial.println("Device has an ID assigned, trying to connect to AP");


    //If control char does not exist or it is 'Y' but device has no ID, start listening
to
    controller
 while ((eeprom_control_char[0] != CONTROL_EEPROM_TARGET_CHAR[0])
 || (eeprom_control_char[0] == CONTROL_EEPROM_TARGET_CHAR[0] && devID == 0))
    {
        waitForControllerCommunication();
        wait(1);
```

```
    }
    Serial.println("\nInfo Received from controller!\n");
    Serial.println("Setting Up connection with Access Point...");
    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.println("Connecting to AP...");
        connectStaticToAP();
        delay(100);
    }
    Serial.printf("Connection with AP successful! Try pinging device: ping
192.168.100.%d\n",
   devID);
}
// ********************************************************************
/// Loop() function not used. Just sleep.
void loop()
{
    delay(50);
}
```

ДОДАТОК Д

Код для RaspPi WIFI_SETUP.PY Code

```
import os
import sys
import subprocess
import re

'''
Script which modifies all the necessary files in the system in order to get a customized
Access Point
running.
It can be called manually, but it will usually be called by another credential extraction
script. The
Access Point
configuration is persistent and should start working after a device reboot.
To call it from a terminal, type: python3 wifi_setup.py <ssid> <key>"
Author: Rafael de la Rosa Ortiz
'''
FAIL = b'Contrase\xc3\xb1a: \r\nsu: Fallo de autenticaci\xc3\xb3n\r\n'
WLAN0_LINES = [
    'interface wlan0\n',
    ' static ip_address=192.168.100.1/24\n',
    ' nohook wpa_supplicant\n',
]
DNSMASK_LINES = [
    'interface=wlan0\n',
    'dhcp-range=192.168.100.200,192.168.100.250,255.255.255.0,24h\n',
]
HOSTAPD_LINES = [
    'interface=wlan0\n',
    'driver=nl80211\n',
    'ssid=IoT_test\n',
    'hw_mode=g\n',
    'channel=7\n',
    'wmm_enabled=0\n',
    'macaddr_acl=0\n',
    'auth_algs=1\n',
    'ignore_broadcast_ssid=0\n',
    'wpa=2\n',
    'wpa_passphrase=ThisIsMyWPAPassword\n',
    'wpa_key_mgmt=WPA-PSK\n',
    'wpa_pairwise=TKIP\n',
    'rsn_pairwise=CCMP\n'
]
LOCALRC_LINE = "sudo iptables-restore < /etc/iptables.ipv4.nat\n"
# Main function of the script: it will install all needed packets, and configure all
necessary files in
order
to
create
# an Access Point with the credentials passed as first and second argument.
# It can be called as many times as needed, it will destroy previous AP and create a new
one.
# It is persistent, Access Point will still be running after a device reboot.
# It needs sudo access, if not provided, it will call itself with sudo privileges to be
able to install
packages and
```

```
# modify some files
def main():
    if os.geteuid() == 0:
        print("Root Access OK")
    else:
        print("No Root... Executing again with sudo")
    subprocess.call(['sudo', 'python3', *sys.argv])
    sys.exit(1)
    if len(sys.argv) < 3:
        print("Wrong format!")
    print(" Format: python3 wifi_setup.py <ssid> <key>")
    exit(1)
    myssid = sys.argv[1]
    mypsk = sys.argv[2]
    global HOSTAPD_LINES
    HOSTAPD_LINES[2] = 'ssid=' + myssid + '\n'
    HOSTAPD_LINES[10] = 'wpa_passphrase=' + mypsk + '\n'


123
print("\nConfiguring Custom Access Point with the following parameters:")
print(" SSID: " + myssid)
print(" PSK: " + mypsk)
print()
# Creates file to save the current ssid and password for user consulting
with open("/home/accesspoint_params.txt", 'w') as f:
    f.write('Current Access Point Credentials:\n')
f.write("ssid: " + myssid + '\n')
f.write("psk: " + mypsk + '\n')
os.system('sudo apt install -y dnsmasq hostapd')
os.system('sudo systemctl stop dnsmasq')
os.system('sudo systemctl stop hostapd')
os.system('sudo apt install -y iptables-persistent')
cmd = 'sudo grep -n "interface wlan0" /etc/dhcpcd.conf'
resp = '0'
try:
    resp = subprocess.check_output(cmd, shell=True).decode('utf-8')
except subprocess.CalledProcessError:
    pass
lineNum = -1
overwrite = False
if re.search('interface wlan0', resp):
    print(resp)
print("Interface exists, overwriting it...")
lineNum = int(re.split(':', resp, 1)[0])
overwrite = True
if overwrite:
    lines = None
with open("/etc/dhcpcd.conf", 'r') as f:
    lines = f.readlines()
if len(lines) > lineNum:
    for l in range(0, len(WLAN0_LINES)):
        try:
        lines[lineNum + l - 1] = WLAN0_LINES[l]
except IndexError:
lines.append(WLAN0_LINES[l])
with open("/etc/dhcpcd.conf", 'w') as f:
    f.writelines(lines)
else:
with open("/etc/dhcpcd.conf", 'a+') as f:
    f.writelines(['\n'] + WLAN0_LINES)
os.system('sudo service dhcpcd restart')
with open("/etc/dnsmasq.conf", 'w') as f:
    f.writelines(DNSMASK_LINES)
```

```
os.system('sudo systemctl reload dnsmasq')
with open("/etc/hostapd/hostapd.conf", 'w') as f:
    f.writelines(HOSTAPD_LINES)  # ToDo set custom SSID and PSK!
os.system('''sudo sed -i 's+#DAEMON_CONF=""+DAEMON_CONF="/etc/hostapd/hostapd.conf"+g'
/etc/default/hostapd''')
os.system('sudo systemctl unmask hostapd')
os.system('sudo systemctl enable hostapd')
os.system('sudo systemctl start hostapd')
os.system('sudo systemctl start dnsmasq')
os.system("sudo sed -i 's/#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/g'
/etc/sysctl.conf")
os.system("sudo iptables -F INPUT")
os.system("sudo iptables -F OUTPUT")
124
os.system("sudo iptables -F FORWARD")
os.system("sudo iptables -t nat -F")
os.system("sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE")
os.system('sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"')  # for rc.local, not
working
currently as rc.local
gets
executed
before
iptables
service
os.system('sudo sh -c "iptables-save > /etc/iptables/rules.v4"')  # for iptables-
persistent package
with open("/etc/rc.local", 'r') as f:
    lines = f.readlines()
exists = False
for line in lines:
    if re.search(LOCALRC_LINE, line):
        exists = True
if not exists:
    lines[lines.index("exit 0\n")] = LOCALRC_LINE
lines.append("exit 0\n")
with open("/etc/rc.local", 'w') as f:
    f.writelines(lines)
print('\n Configuration completed. The WiFi AP should be up and running. It will persist
if device
gets
rebooted.
')
if __name__ == "__main__":
    main()
```

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1014800560

Дата перевірки:
25.04.2023 18:17:01 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
25.04.2023 18:18:20 EEST

ID користувача:
100005591

Назва документа: **Томусяк_Метод та програмно-технічний засіб генерування ключів аутентифікації**

Кількість сторінок: **91** Кількість слів: **18249** Кількість символів: **138430** Розмір файлу: **2.07 MB** ID файлу: **1014504771**

2.26%
Схожість

Найбільша схожість: **0.5%** з Інтернет-джерелом (http://bitterli.fr/page/88)

2.05% Джерела з Інтернету	204	Сторінка 93
0.86% Джерела з Бібліотеки	113	Сторінка 94

0.29% Цитат

Цитати	1	Сторінка 95
Посилання	1	Сторінка 95

0%
Вилучень

Немає вилучених джерел

Довідка: ВХНУ ТН 1/04/23

Видання: Вісник Хмельницького національного університету. Технічні науки

Категорія фаховості видання: Затверджено як наукове фахове видання України, у якому можуть публікуватися результати дисертаційних робіт на здобуття наукових ступенів доктора наук, кандидата наук та ступеня доктора філософії, категорії «Б» (наказ МОН №1643 від 28.12.2019, наказ МОН №409 від 17.03.2020).

Напрям – технічні науки за спеціальностями – 101, 121, 122, 123, 124, 125, 141, 151, 161, 172, 181, 182 (28.12.2019), спеціальності – 131, 132, 133 (17.03.2020).

Назва статті: МЕТОД ТА ПРОГРАМНО-ТЕХНІЧНИЙ ЗАСІБ ГЕНЕРУВАННЯ КЛЮЧІВ АУТЕНТИФІКАЦІЇ

Автори: ТОМУСЯК А. В. (Хмельницький національний університет)

Номер, у який прийнято статтю: №3, до друку рекомендовано буде до 30 червня 2023 року.

18.04.2023

Начальника відділу
інтелектуальної власності та трансферу технологій Ю.В.Кравчик

ВІДДІЛ КАДРІВ

І.С.Мартинюк

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Томусяк Андрій Валерійович

Тема: Метод та програмно-технічний засіб генерування ключів аутентифікації

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 0 Кількість сторінок записки 85

1. Короткий зміст роботи та прийнятих рішень: в результаті виконаного наукового дослідження розроблено метод та програмно-технічний засіб генерування ключів аутентифікації

2. Висновок про відповідність роботи дипломному завданню: Дипломна робота відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі магістерської роботи проведено дослідження розподілу ключів аутентифікації в системі пристроїв Інтернету Речей, а також досліджено вразливості різних протоколів безпеки, таких як атаки перевстановлення ключа або атаки Krack, атака на захищене налаштування Wi-Fi (WPS), вразливість Bluetooth та вразливості BlueBorne. У другому розділі обговорюється опис методу та програмно-технічному засобу генерування ключів аутентифікації. У розділі розглядаються різні аспекти розробки та застосування цього методу. Наведено опис процесу налаштування сценарію генерування ключів аутентифікації. Розглянуто різні параметри, які можуть вплинути на процес генерування ключів, та надано рекомендації щодо їх вибору. Третій розділ дозволив вивчити різні аспекти розробки методу та програмно-технічного засобу генерування ключів аутентифікації. У рамках цього розділу було описано архітектуру та конфігурації сценарію, апаратне забезпечення, програмне забезпечення та дизайн обладнання. У четвертому розділі проводиться аналіз різних версій системи, їх часових параметрів, програмно-технічного засобу та засобу генерування ключа аутентифікації.

4. Позитивні сторони роботи: Найбільшою перевагою роботи є грунтовний аналіз існуючих методів та розробка програмно-технічного засобу для застосування більш вдосконалених методів при генеруванні ключів аутентифікації.

5. Негативні сторони роботи: граматичні помилки

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: Робота виконана на високому науковому рівні

8. Інші зауваження: –

9. Оцінка дипломної роботи: Розглянувши представлену роботу, вважаю, що робота заслуговує оцінки 3,5 (D).

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Мартинюк Валерій Володимирович, зав. каф. „Автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

"10" травня 2023 р. (підпис)

Завідувачу кафедри КІС
д-р.техн.наук, проф. Говорущенко Т. О.

Томусяка Андрія Валерійовича
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

8 травня 2023 року

Томусяк АВ

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод та технічно програмний засіб генерування ключів аутентифікації

Автор: Томусяк Андрій Валерійович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Іванов Олексій Валентинович

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах є збіг зі звітом з науково-дослідної практики автора Андрія Томусяка "Метод та технічно програмний засіб генерування ключів аутентифікації ", який було додано в репозитраій ХНУ 11 травня 2023 року;
2) усі запозичення фрагментарні, або мають належним чином оформлені посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами. про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
4) в якості запозичень в окремих місцях системою зафіксовано послідовності чотирьохрозряних двійкових кодів, які є вхідними даними до великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
5) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck. складає 2.26% і адресується до 80 першоджерела; та системою Anti-Plagiarism складає 0%, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О. В. Іванов

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко