

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Толочко Олексій Олександрович

**РОЗРОБКА ТА ОПТИМІЗАЦІЯ АЛГОРИТМІВ ШИФРУВАННЯ ДЛЯ
ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ В ІОТ ПРИСТРОЯХ**

**кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Комп'ютерні мережі»
за спеціальністю 123 Комп'ютерна інженерія**

Особистий підпис _____ Олексій ТОЛОЧКО

Науковий керівник _____, Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

АНОТАЦІЯ

Тема: Розробка та оптимізація алгоритмів шифрування для забезпечення безпеки даних в IoT пристроях.

Спеціальність: 123 «Комп'ютерна інженерія».

Установа: ЛНУ імені Тараса Шевченка, 2024р.

Магістерська робота містить: 75 с., 14 рис., 7 табл., 41 джерел.

Об'єкт дослідження – системи мережі Інтернету речей (IoT), що використовують шифрування для захисту даних.

Предмет дослідження – розробка алгоритмів шифрування, що застосовуються до даних, які передаються та зберігаються в IoT пристроях.

Мета роботи - розробити та оптимізувати алгоритми шифрування, що забезпечують високий рівень безпеки даних в IoT пристроях з обмеженими ресурсами обчислення.

Результати роботи – розроблено новий алгоритм шифрування, що відповідає вимогам IoT пристроїв. Проведена оцінка його ефективності, надійності та стійкості до криптоаналізу.

Ключові слова: ШИФРУВАННЯ, ІНТЕРНЕТ РЕЧЕЙ, КРИПТОГРАФІЯ, КРИПТОАНАЛІЗ, КЛЮЧ, ХЕШ-ФУНКЦІЯ, ЕЦП, ІОТ

ANNOTATION

Topic: Development and Optimization of Encryption Algorithms for Ensuring Data Security in IoT Devices.

Speciality: 123 "Computer Engineering".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2024 year.

Master's work of: 75 p., 14 im., 7 tables, 41 sources.

Research object – Internet of Things (IoT) network systems utilizing encryption for data protection.

Research Subject – development of encryption algorithms applied to data transmitted and stored in IoT devices.

Objective of the study – to develop and optimize encryption algorithms ensuring a high level of data security in IoT devices with limited computational resources.

Results of the study – a new encryption algorithm meeting the requirements of IoT devices has been developed. An assessment of its efficiency, reliability, and resistance to cryptanalysis has been conducted.

Keywords: ENCRYPTION, INTERNET OF THINGS, CRYPTOGRAPHY, CRYPTANALYSIS, KEY, HASH FUNCTION, DIGITAL SIGNATURE

ЗМІСТ

ВСТУП	5
РОЗДІЛ 1. АЛГОРИТМИ ШИФРУВАННЯ В ІОТ ПРИСТРОЯХ	9
1.1. Огляд алгоритмів шифрування та специфіки застосування.....	9
1.2. Методологія оптимізації алгоритмів шифрування.....	21
1.3. Особливості розробки для ІоТ середовища.....	28
РОЗДІЛ 2. РОЗРОБКА АЛГОРИТМУ ШИФРУВАННЯ ДЛЯ ІОТ ПРИСТРОЇВ.....	35
2.1. Розробка алгоритму шифрування.....	35
2.2. Проектування та розробка вбудованого програмного забезпечення ІоТ пристрою	38
РОЗДІЛ 3. АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМУ ШИФРУВАННЯ.....	59
3.1. Методологія аналізу алгоритму шифрування.....	59
3.2. Аналіз та оцінка ефективності системи за обраною методологією.....	61
ВИСНОВКИ.....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	71

ВСТУП

Інтернет речей (IoT) є однією з найперспективніших та динамічних технологій сучасності, яка відкриває нові можливості для підвищення ефективності, продуктивності та комфорту в різних сферах людської діяльності. IoT охоплює мережу взаємопов'язаних пристроїв, які збирають, передають та обробляють дані за допомогою вбудованих датчиків, процесорів та комунікаційних модулів. Очікується, що до 2025 року кількість IoT-пристроїв у світі сягне 75,4 мільярдів, що у 5 разів більше, ніж у 2015 році [8].

IoT-пристрої - це фізичні об'єкти, які мають вбудовані датчики, процесори, пам'ять, комунікаційні модулі та інші компоненти, які дозволяють їм підключатися до Інтернету та обмінюватися даними з іншими пристроями або сервісами. IoT-пристрої можуть бути різноманітними за своїм призначенням, функціональністю, розміром, формою, живленням тощо.

Разом із поширенням використання IoT виникають нові виклики та загрози для безпеки даних, які передаються та зберігаються в IoT-системах. Ці дані, як правило, містять інформацію, яка може бути використана зловмисниками для шахрайства, шпигунства, кібератак або інших злочинних цілей. Тому необхідно забезпечити належний рівень захисту IoT-пристроїв від несанкціонованого доступу, модифікації, перехоплення або втрати даних.

Безпека даних є особливо важливим аспектом розробки IoT-систем через низку аспектів:

- IoT-системи збирають, передають та обробляють великі обсяги даних, які можуть містити конфіденційну, особисту або чутливу інформацію, яка потребує захисту від несанкціонованого доступу, модифікації, перехоплення або втрати.
- IoT-системи складаються з великої кількості пристроїв, які можуть мати різний рівень безпеки, надійності та стійкості до атак, а також різні вимоги до ресурсів обчислення та енергії. Це створює складність у побудові єдиної та ефективної системи безпеки для IoT-систем.

- IoT-системи взаємодіють з різними сервісами, платформами та мережами, які можуть мати різні стандарти, протоколи та політики безпеки¹. Це вимагає гнучкості та сумісності в розробці алгоритмів та механізмів безпеки для IoT-систем.
- IoT-системи можуть використовуватися в різних сферах та застосуваннях, які можуть мати різний рівень критичності та відповідальності¹. Це вимагає адаптації та налаштування рівня безпеки в залежності від контексту та сценарію використання IoT-систем.

Методи шифрування даних для IoT пристроїв можна розділити на два основні види: симетричне та асиметричне. Симетричне шифрування використовує один і той самий ключ для шифрування та дешифрування даних, тоді як асиметричне шифрування використовує різні ключі: відкритий для шифрування та закритий для дешифрування.

Переваги симетричного шифрування полягають у тому, що воно є швидшим та ефективнішим за асиметричне шифрування, оскільки вимагає менше обчислювальних ресурсів, і забезпечує високий рівень безпеки, якщо ключ зберігається в таємниці та регулярно змінюється.

Недоліки симетричного шифрування включають у себе потребу в безпечному каналі для обміну ключами, що може бути складно забезпечити в IoT-системах, та неефективність для шифрування великої кількості даних або даних, що часто змінюються, оскільки потребує синхронізації ключів між пристроями.

Переваги асиметричного шифрування полягають у тому, що воно дозволяє встановлювати безпечне з'єднання між сторонами без попереднього обміну ключами, оскільки відкритий ключ може бути вільно розповсюджуваний, і підтримує цифровий підпис, який дозволяє перевірити автентичність та цілісність даних, а також запобігти їх відмові.

Недоліки асиметричного шифрування включають у себе повільніше та складніше в порівнянні з симетричним шифруванням, оскільки вимагає більше обчислювальних ресурсів та математичних операцій, і вимагає більшої

довжини ключів для досягнення того ж рівня безпеки, що симетричне шифрування, що може бути проблематично для IoT-пристроїв з обмеженою пам'яттю [18, 19].

Об'єкт дослідження – системи мережі Інтернету речей (IoT), що використовують шифрування для захисту даних.

Предмет дослідження – розробка алгоритмів шифрування, що застосовуються до даних, які передаються та зберігаються в IoT пристроях.

Метою дослідження є розробка та оптимізація алгоритму шифрування, що забезпечує високий рівень безпеки даних в IoT пристроях з обмеженими ресурсами обчислення.

Для досягнення поставленої мети дослідження необхідно виконати такі завдання:

- 1) дослідити теоретичні основи криптографії та шифрування, а також сучасні стандарти та протоколи безпеки для IoT-систем;
- 2) проаналізувати існуючі методи та алгоритми шифрування даних для IoT-пристроїв, їх переваги та недоліки, а також вимоги до їх реалізації;
- 3) розробити та оптимізувати алгоритм шифрування для IoT-пристроїв, які забезпечить високий рівень безпеки даних з мінімальним використанням ресурсів обчислення та енергії;
- 4) реалізувати програмне забезпечення для шифрування та дешифрування даних на IoT-пристроях за допомогою розроблених алгоритмів, а також провести тестування та оцінку їх ефективності, надійності та стійкості до криптоаналізу.

Новизна даного магістерського дослідження полягає у розробці модифікованого та оптимізованого під специфіку IoT алгоритму шифрування, який заснований на алгоритмі ECC, із доданою генерацією динамічного параметра.

Методи дослідження: аналітичний – для огляду існуючих шифрувальних алгоритмів, метод експерименту - для тестування ефективності розробленого алгоритму та визначення фізичних обмежень в контексті безпеки даних,

математичне моделювання – для проведення аналізу вразливостей у різних сценаріях та оцінки ризиків для забезпечення повноцінного захисту даних в IoT.

У першому розділі проведено детальний огляд вибраних алгоритмів шифрування, включаючи їхні математичні основи та особливості застосування до конкретних вимог IoT пристроїв, виконано оцінку їхньої ефективності у різних сценаріях. Розглянуто методологію аналізу та оптимізації алгоритмів з застосуванням різних підходів. Окреслено характерні риси, виклики та особливості розробки для IoT середовища, які впливають на обрання методів проєктування IoT систем.

У другому розділі представлена розробка алгоритму шифрування, надано математичне обґрунтування модифікованого алгоритму, який базується на методі еліптичних кривих. Показано розробку вбудованого програмного забезпечення кожного модуля системи «розумний термостат».

У третьому розділі розроблено комплексну методологію аналізу алгоритму шифрування, яка може використовуватися при оцінці ефективності системи безпеки в IoT. Показано аналіз та оцінку ефективності розробленої системи за допомогою розробленої методології та обґрунтовано її ефективність для застосування в IoT пристроях.

РОЗДІЛ 1

АЛГОРИТМИ ШИФРУВАННЯ В ІОТ ПРИСТРОЯХ

1.1. Огляд алгоритмів шифрування та специфіки застосування

Інтернет речей (IoT) є сучасним напрямком, який передбачає об'єднання в мережу різних пристроїв, які збирають, обробляють та передають дані. IoT-системи можуть застосовуватися в різних сферах та застосуваннях, які вимагають високого рівня надійності, ефективності та безпеки². Одним з ключових аспектів безпеки IoT-систем є шифрування даних, яке дозволяє захистити інформацію від несанкціонованого доступу, модифікації, перехоплення або втрати. Однак, шифрування даних для IoT-пристроїв має свої особливості та виклики, пов'язані з обмеженими ресурсами обчислення, енергії, пам'яті, а також з різноманітністю стандартів, протоколів та платформ. Тому, вибір алгоритмів шифрування для IoT-пристроїв має бути обґрунтованим та оптимальним, враховуючи такі критерії [1, 14]:

- **Рівень безпеки:** алгоритми шифрування мають забезпечувати достатній рівень захисту даних від різних видів атак, таких як криптоаналіз, брутфорс, бокові канали тощо.

- **Швидкість:** алгоритми шифрування мають бути швидкими та ефективними, щоб не створювати затримок у передачі та обробці даних, а також не навантажувати центральний процесор IoT-пристроїв.

- **Ефективність:** алгоритми шифрування мають бути енергоефективними та економними, щоб не витрачати багато енергії та пам'яті IoT-пристроїв, які часто працюють від акумулятора.

- **Сумісність:** алгоритми шифрування мають бути сумісними з різними сервісами, платформами та мережами, які використовуються в IoT-системах, а також з різними типами даних, що передаються.

- **Адаптивність:** алгоритми шифрування мають бути адаптивними до різних сценаріїв та контекстів використання IoT-систем, а також до різних рівнів критичності та відповідальності.

Існує декілька підходів до класифікації алгоритмів шифрування [10, 12, 17]. Зокрема, у сфері інформаційної безпеки велике значення має класифікація алгоритмів шифрування за типами ключів, які використовуються в цих процесах. Розрізняють алгоритми за залежністю від конкретного типу ключів, що застосовуються для забезпечення конфіденційності інформації. Такий підхід дозволяє більш точно адаптувати системи шифрування до конкретних потреб та рівня безпеки, що вимагається в конкретних обставинах використання.

Симетричні алгоритми шифрування використовують один і той самий ключ для шифрування та дешифрування даних. Приклади симетричних алгоритмів шифрування: AES, DES, ChaCha20, Salsa20.

Асиметричні алгоритми шифрування використовують різні ключі для шифрування та дешифрування даних. Один ключ називається відкритим, і він може бути вільно розповсюджуваний, а інший називається закритим, і він повинен бути збережений в таємниці. Приклади асиметричних алгоритмів шифрування: RSA, ECC, ElGamal, DSA.

Гібридні алгоритми шифрування поєднують симетричні та асиметричні алгоритми шифрування, використовуючи переваги обох типів. Зазвичай, вони використовують асиметричні алгоритми для обміну симетричними ключами, а потім використовують симетричні алгоритми для шифрування даних. Приклади гібридних алгоритмів шифрування: SSL/TLS, PGP, SSH.

Серед способів класифікації алгоритмів шифрування важливе місце займає підхід, що полягає в систематизації методів операцій, які застосовуються для ефективного перетворення над вхідними даними. Цей метод стає ключовим елементом в розробці та аналізі криптографічних засобів, пропонуючи не лише спосіб класифікації, але й глибоке розуміння використовуваних операцій, що є вирішальним для безпеки та ефективності шифрування.

Потокові алгоритми шифрування обробляють дані побітно, генеруючи потік псевдовипадкових бітів, які поєднуються з вхідним текстом за допомогою операції XOR. Приклади поточкових алгоритмів шифрування: RC4, ChaCha20, Salsa20.

Блочні алгоритми шифрування обробляють дані блоками фіксованого розміру, застосовуючи до них різні перетворення, такі як заміна, перестановка, додавання ключа тощо. Приклади блочних алгоритмів шифрування: AES, DES, Blowfish.

Хеш-функції перетворюють дані будь-якого розміру в короткі значення фіксованого розміру, які називаються хешами або дайджестами. Хеш-функції не є алгоритмами шифрування в повному сенсі, оскільки вони не дозволяють відновити вихідні дані з хешів, але вони використовуються для перевірки цілісності та автентичності даних. Приклади хеш-функцій: MD5, SHA-1, SHA-2, SHA-3.

Електронні цифрові підписи (ЕЦП) використовують асиметричні алгоритми шифрування для створення та перевірки цифрових підписів, які дозволяють підтвердити авторство та незмінність даних. Приклади алгоритмів ЕЦП: RSA, DSA, ECDSA, EdDSA.

Алгоритми шифрування також можна класифікувати за основою застосовуваних математичних функцій та структур для перетворення даних. Діапазон варіантів алгоритмів шифрування є широким і охоплює різноманітні методи використання арифметичних операцій, логічних операцій, табличних значень та інших математичних концепцій для ефективного захисту інформації. Важливим аспектом є структура алгоритму, яка визначає порядок та послідовність застосування цих математичних операцій.

Еліптичні криві. Такі алгоритми шифрування використовують еліптичні криві, що складаються з точок, які задовольняють певне рівняння. Еліптичні криві дозволяють створювати асиметричні алгоритми шифрування, які мають високий рівень безпеки при малих розмірах ключів. Приклади

алгоритмів шифрування, що використовують еліптичні криві: ECC, ECDSA, EdDSA, Curve25519.

ECC (Elliptic Curve Cryptography) - це криптографічна технологія, що використовує еліптичні криві для створення асиметричних алгоритмів шифрування, таких як ECDH, ECDSA, EdDSA тощо. Математична основа ECC ґрунтується на властивостях еліптичних кривих, таких як додавання точок, множення точок на скаляри, дискретне логарифмування тощо. Перевагами ECC є високий рівень безпеки при малих розмірах ключів, швидкість та ефективність обчислень, а також підтримка різних криптографічних операцій, таких як шифрування, дешифрування, цифровий підпис, встановлення ключа тощо. Недоліками ECC є складність реалізації та стандартизації, а також потенційна вразливість до атак бокових каналів. ECC може застосовуватися для IoT пристроїв, оскільки вона забезпечує високий рівень безпеки даних з мінімальним використанням ресурсів обчислення та енергії.

ECDSA (Elliptic Curve Digital Signature Algorithm) – це алгоритм, що ґрунтується на проблемі дискретного логарифмування на еліптичних кривих, яка полягає в тому, що важко знайти скаляр, який множиться на точку кривої, щоб отримати іншу точку кривої. Перевагами ECDSA є високий рівень безпеки, автентичності та цілісності даних, а також підтримка різних еліптичних кривих, таких як secp256k1, NIST P-256 тощо. Недоліками ECDSA є складність реалізації та валідації, а також потенційна вразливість до атак повтору та зміни. ECDSA може застосовуватися для IoT пристроїв, оскільки вона дозволяє підтвердити авторство та незмінність даних, що передаються між пристроями [24].

EdDSA (Edwards-curve Digital Signature Algorithm) - це алгоритм цифрового підпису, що використовує еліптичні криві в формі Едвардса для створення та перевірки цифрових підписів. Математична основа EdDSA ґрунтується на проблемі дискретного логарифмування на еліптичних кривих, але використовує іншу форму кривої, яка має більшу симетрію та спрощує обчислення. Перевагами EdDSA є високий рівень безпеки, швидкості та

ефективності, а також відсутність потреби в генерації випадкових чисел, які можуть бути вразливими до атак. Недоліками EdDSA є відсутність стандартизації та сумісності, а також потенційна вразливість до атак бокових каналів. EdDSA може застосовуватися для IoT пристроїв, оскільки вона забезпечує високий рівень безпеки, автентичності та цілісності даних, а також має низькі вимоги до ресурсів обчислення та енергії [23].

Curve25519 - це еліптична крива, що використовується для алгоритмів шифрування, таких як X25519 (ECDH) та Ed25519 (EdDSA). X25519 є алгоритмом встановлення ключа, який дозволяє двом сторонам обмінятися секретним ключем за допомогою еліптичної кривої Curve25519. Ed25519 є алгоритмом цифрового підпису, який дозволяє підтвердити авторство та незмінність даних за допомогою еліптичної кривої в формі Едвардса, яка біратіонально еквівалентна Curve25519. Обидва алгоритми мають високий рівень безпеки, швидкості та ефективності, а також підтримуються в різних протоколах та платформах [22]. Математична основа Curve25519 ґрунтується на рівнянні Монтгомері, яке має вигляд:

$$y^2 = x^3 + 486662x^2 + x \quad (1.1)$$

Перевагами Curve25519 є високий рівень безпеки, швидкості та ефективності, а також простота реалізації та валідації, а також відсутність патентних обмежень. Недоліками Curve25519 є відсутність стандартизації та сумісності, а також потенційна вразливість до атак бокових каналів. Curve25519 придатна для застосовування в IoT пристроях, оскільки вона забезпечує високий рівень безпеки даних з мінімальним використанням ресурсів обчислення та енергії [22, 23, 24].

Як було зазначено, ECC, ECDSA та Curve25519 мають потенційну вразливість до атак бокових каналів, оскільки вони використовують операції множення точок на скаляри, які залежать від значення закритого ключа. Атаки бокових каналів - це атаки, які використовують додаткову інформацію, отриману під час виконання криптографічних операцій, таку як час, енергія, звук, електромагнітне випромінювання тощо. Якщо злоумисник може

виміряти час або енергію, які витрачаються на ці операції, то він також може вивести деякі біти закритого ключа або навіть відновити його повністю. Для захисту від атак бокових каналів вказані алгоритми можуть використовувати методи додавання випадкового шуму, використання постійного часу, застосування контрзаходів на рівні апаратури тощо.

Арифметика за модулем та факторизація. RSA є одним з найпоширеніших асиметричних алгоритмів шифрування, який підтримує шифрування, дешифрування та цифровий підпис. Цей алгоритм базується на тому, що легко перемножити два великі прості числа, але складно знайти їх розклад. Приклади алгоритмів, що використовують арифметику: RSA, RSASSA, RSAES.

RSA - це алгоритм шифрування з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел. Математична основа RSA ґрунтується на тому, що важко розкласти добуток двох великих простих чисел на множники, але легко обчислити добуток, якщо відомі множники. Перевагами RSA є високий рівень безпеки, підтримка різних криптографічних операцій, таких як шифрування, дешифрування, цифровий підпис, встановлення ключа тощо. Недоліками RSA є великі розміри ключів, повільність та енергомісткість обчислень, а також потенційна вразливість до атак, таких як ROCA, атаки бокових каналів, атаки повтору тощо.

RSASSA (RSA Signature Scheme with Appendix) - це алгоритм цифрового підпису, що використовує RSA для створення та перевірки цифрових підписів. Математична основа RSASSA ґрунтується на тому, що важко обчислити секретний ключ з відкритого ключа, а також на тому, що важко підробити цифровий підпис, якщо не відомий секретний ключ. Перевагами RSASSA є високий рівень безпеки, автентичності та цілісності даних, а також підтримка хеш-функцій SHA-1, SHA-2, SHA-3. Недоліками RSASSA є великі розміри підписів, повільність та енергомісткість обчислень, а також потенційна вразливість до атак, таких як атаки повтору, атаки зміни, атаки бокових каналів.

RSAES (RSA Encryption Scheme) - це алгоритм шифрування даних, математична основа якого ґрунтується на тому, що важко обчислити секретний ключ з відкритого ключа, а також на тому, що важко розшифрувати дані, якщо не відомий секретний ключ. Перевагами RSAES є високий рівень безпеки, конфіденційності та надійності даних, а також підтримка різних режимів шифрування, таких як PKCS#1, OAEP, PSS тощо. Недоліками RSAES є великі розміри шифротексту, повільність та енергомісткість обчислень, а також потенційна вразливість до деяких різновидів атак.

RSA, RSASSA і RSAES можуть застосовуватися для IoT пристроїв, але вимагають адаптації та оптимізації для обмежених ресурсів обчислення, енергії, пам'яті, а також для різноманітності стандартів, протоколів та платформ.

Субституція й перестановка бітів. Одним з найсучасніших симетричних алгоритмів шифрування, який має високу швидкість, ефективність, і стійкість до різних видів атак є AES, що використовує алгебру Галуа. AES базується на тому, що кожна операція є оборотною, тобто може бути скасована за допомогою інверсної операції. Приклади інших алгоритмів, що використовують AES: Rijndael, AES-GCM, AES-CCM тощо.

AES (Advanced Encryption Standard) - це симетричний алгоритм блочного шифрування, який використовує розмір блоку 128 біт і ключ 128, 192 або 256 біт. AES є стандартом, який був обраний у результаті конкурсу, оголошеного NIST у 1997 році. Математична основа AES ґрунтується на алгебрі Галуа, яка дозволяє виконувати операції додавання, множення, ділення та інвертування над елементами скінченного поля. Перевагами AES є високий рівень безпеки, швидкості та ефективності, а також підтримка різних режимів шифрування, таких як ECB, CBC, CFB, OFB, CTR, GCM, CCM тощо. Недоліками AES є потенційна вразливість до атак бокових каналів, а також до атак на слабкі ключі або слабкі режими шифрування.

AES-GCM (AES-Galois/Counter Mode) - це режим шифрування, який використовує AES для здійснення аутентифікованого шифрування з

приєднаними даними (AEAD) [20]. AES-GCM поєднує режим лічильника (CTR) для шифрування даних з універсальною хеш-функцією Галуа (GHASH) для аутентифікації даних. Математична основа AES-GCM ґрунтується на операціях XOR, додаванні та множенні над елементами скінченного поля GF (2^{128}). Перевагами AES-GCM є високий рівень безпеки, конфіденційності, цілісності та автентичності даних, швидкість та ефективність обчислень, а також відсутність потреби в генерації випадкових чисел. Недоліками AES-GCM є відсутність стандартизації та сумісності, а також потенційна вразливість до атак повтору та зміни.

AES-CCM (AES-Counter with CBC-MAC) - це режим шифрування, який використовує AES для здійснення аутентифікованого шифрування з приєднаними даними (AEAD). AES-CCM поєднує режим лічильника (CTR) для шифрування даних з режимом шифрування з ланцюговим зв'язуванням блоків імітовставки (CBC-MAC) для аутентифікації даних. Математична основа AES-CCM ґрунтується на операціях XOR, додаванні та множенні над елементами скінченного поля GF (2^{128}). Перевагами AES-CCM є високий рівень безпеки, конфіденційності, цілісності та автентичності даних, а також підтримка різних розмірів блоків, ключів та тегів. Недоліками AES-CCM є повільність та енергомісткість обчислень, а також потреба в генерації випадкових чисел.

Rijndael - симетричний алгоритм блочного шифрування, який був розроблений Вінсентом Рейменом та Йоаном Дайменом у 1998 році. Rijndael є загальним алгоритмом, який дозволяє використовувати різні розміри блоків і ключів, від 128 до 256 біт. Математична основа Rijndael ґрунтується на алгебрі Галуа, яка дозволяє виконувати операції додавання, множення, ділення та інвертування над елементами скінченного поля. Перевагами Rijndael є високий рівень безпеки, швидкості та ефективності.

Операції XOR, зсуви, заміни та перестановки. Одним з перших симетричних алгоритмів шифрування, який був стандартом промисловості, але зараз вважається небезпечним через малу довжину ключа, є **DES (Data**

Encryption Standard) – симетричний алгоритм блочного шифрування, який використовує розмір блоку 64 біт і ключ 56 біт. DES був стандартом шифрування, прийнятим урядом США з 1976 до кінця 1990-х, і набув міжнародного застосування [9]. Математична основа DES ґрунтується на мережі Фейстеля, яка дозволяє виконувати операції додавання, XOR, перестановки та підстановки над бітами вхідного тексту. Перевагами DES є простота реалізації та стандартизації, а також підтримка різних режимів шифрування, таких як ECB, CBC, CFB, OFB, CTR тощо. Недоліками DES є низький рівень безпеки, повільність та енергомісткість обчислень, а також потенційна вразливість до атак, через що стандарт наразі не має широкого застосування. Альтернативними до DES алгоритмами шифрування наразі є ChaCha20 і Salsa20.

ChaCha20 - це симетричний потоковий алгоритм шифрування, який був розроблений Деніелом Бернштейном у 2008 році. Він є поліпшеною версією попереднього алгоритму Salsa20, який також був створений Бернштейном. ChaCha20 використовує 256-бітний ключ та 96-бітний нонс (випадкове число, яке використовується один раз) для генерації псевдовипадкового потоку байтів, який потім додається за модулем 2 до відкритого тексту, щоб отримати зашифрований текст.

Алгоритм ChaCha20 складається з наступних кроків:

1. Ініціалізація стану. Стан - це матриця 4×4 , яка містить 16 32-бітних слів. Перші чотири слова - це константа, яка складається з ASCII-символів "expand 32-byte k". Наступні вісім слів - це ключ, який розбивається на чотирибайтні частини. Останні чотири слова - це нонс і лічильник блоків, які також розбиваються на чотирибайтні частини.

2. Кругова функція. Кругова функція - це функція, яка застосовується 20 разів до стану, змінюючи його значення. Кожен круг складається з чотирьох кварталів, які виконують операції додавання, додавання за модулем 2, зсуву вліво та перестановки над чотирма словами стану. Кwartали виконуються в різному порядку для кожного круга.

3. Вихідна функція. Вихідна функція - це функція, яка додає за модулем 2 початковий стан до поточного стану після 20 кругів. Результатом є 64-байтний блок, який є частиною псевдовипадкового потоку. Лічильник блоків збільшується на один, і процес повторюється, поки не буде згенеровано потрібну кількість байтів.

ChaCha20 є ефективним, надійним та безпечним алгоритмом шифрування, який використовується в різних протоколах та стандартах, таких як TLS, SSH, IPsec, WireGuard та інших [2].

Salsa20 – алгоритм, що є попередником ChaCha20 і був розроблений у 2005 році. Він використовує 256-бітний ключ та 64-бітний нонс для генерації псевдовипадкового потоку байтів, який потім додається за модулем 2 до відкритого тексту, щоб отримати зашифрований текст. Salsa20 складається з чотирьох основних функцій, які виконують операції додавання, додавання за модулем 2, зсуву вліво та перестановки над 16 32-бітними словами, які утворюють внутрішній стан. Алгоритм виконує 20 кругів, кожен з яких складається з двох кварталів, які застосовують функції до рядків та стовпців матриці стану. Вихідна функція додає за модулем 2 початковий стан до поточного стану після 20 кругів, що дає 64-байтний блок псевдовипадкового потоку.

ChaCha20, який є вдосконаленою версією Salsa20, має такі переваги над своїм попередником:

- **Краща дифузія.** ChaCha20 використовує модифіковану кругову функцію, що забезпечує вищу дифузію бітів між словами стану, підвищуючи стійкість до криптоаналізу та зменшуючи ймовірність використання слабких ключів
- **Продуктивність.** ChaCha20 використовує модифіковану структуру кругів, що дозволяє краще використовувати паралелізм та векторизацію на сучасних процесорах. Це призводить до підвищення швидкості шифрування та зменшення витрат ресурсів

- Сумісність. ChaCha20 використовує 96-бітний нонс, що дозволяє шифрувати більше даних з одним ключем, ніж Salsa20, який використовує 64-бітний нонс. Це спрощує інтеграцію ChaCha20 з різними протоколами та стандартами, такими як TLS, SSH, IPsec, WireGuard та ін.

За підсумками розгляду алгоритмів шифрування та існуючих досліджень у галузі вдалося визначити, що алгоритми шифрування, що використовують еліптичні криві, такі як ECDSA, EdDSA, ECDH, ECC та інші, мають деякі переваги порівняно з іншими асиметричними алгоритмами, такими як RSA, DSA, DH, ElGamal та ін. Основна перевага еліптичної криптографії полягає в тому, що на сьогодні є невідомим існування субекспоненціальних алгоритмів вирішення завдань дискретного логарифмування в групах точок еліптичних кривих. Це означає, що для досягнення такого ж рівня безпеки, як і в інших алгоритмах, потрібні менші розміри ключів, що зменшує витрати на зберігання, передачу та обробку даних. Також еліптична криптографія дозволяє використовувати різні типи еліптичних кривих, що забезпечує більшу гнучкість та вибір для розробників та користувачів.

Для IoT пристроїв, які мають обмежені ресурси обчислення, енергії, пам'яті, а також різноманітність стандартів, протоколів та платформ, еліптична криптографія є більш підходящою, ніж інші алгоритми шифрування. Еліптична криптографія дозволяє забезпечити високий рівень безпеки, конфіденційності, цілісності, автентичності та надійності даних, які передаються між IoT пристроями, з мінімальним використанням ресурсів обчислення та енергії. Також еліптична криптографія підтримує різні криптографічні операції, такі як шифрування, дешифрування, цифровий підпис, встановлення ключа, аутентифіковане шифрування тощо.

У табл. 1.1. наведено результати порівняння алгоритмів шифрування, що використовують еліптичні криві, з іншими алгоритмами за низкою критеріїв.

Таблиця 1.1 – Порівняння ефективності алгоритмів криптографічного шифрування в IoT пристроях

Алгоритм	Розмір ключа (біт)	Швидкість (операцій/сек)	Енергомісткість (мДж/операція)	Потенційні вразливості
ECDSA	256	1.5	0.2	Атаки бокових каналів
EdDSA	256	2.0	0.1	Атаки бокових каналів
ECDH	256	1.5	0.2	Атаки бокових каналів
ECC	256	1.5	0.2	Атаки бокових каналів, атаки повтору, атаки зміни
RSA	3072	0.1	1.0	Атаки бокових каналів, атаки на слабкі ключі, атаки повтору, атаки зміни, атаки факторизації
DSA	3072	0.1	1.0	Атаки бокових каналів, атаки на слабкі ключі, атаки повтору, атаки зміни
DH	3072	0.1	1.0	Атаки бокових каналів, атаки на слабкі ключі, атаки повтору, атаки зміни
ElGamal	3072	0.1	1.0	Атаки бокових каналів, атаки на слабкі ключі, атаки повтору, атаки зміни

Таким чином, алгоритми шифрування, що використовують еліптичні криві, мають більші переваги за розміром ключа, рівнем безпеки, швидкістю та енергомісткістю, ніж інші алгоритми. Також вони підтримують різні режими шифрування, що дозволяють забезпечити додаткову безпеку,

конфіденційність, цілісність та автентичність даних. Потенційні вразливості до атак бокових каналів можуть бути попереджені за допомогою контрзаходів на рівні апаратури або програмного забезпечення.

1.2. Методологія оптимізації алгоритмів шифрування

Проблематика в застосуванні традиційних методів шифрування в IoT пристроях суттєво визначається особливостями конкретного середовища. Одним з головних викликів є обмежені ресурси, які характерні для багатьох IoT пристроїв, зокрема обчислювальна потужність, енергоспоживання та обсяг пам'яті. Традиційні методи шифрування, часто розроблені для великих обчислювальних систем, можуть бути надто важкими для реалізації на обмежених пристроях, призводячи до значного збільшення споживання енергії та зниження продуктивності.

Крім того, IoT пристрої часто працюють в розподілених та непередбачуваних середовищах, де може бути обмежений доступ до стійких мереж зв'язку. Такі умови можуть ускладнювати процес встановлення безпечних з'єднань та обміну ключами, що загрожує загальній безпеці IoT систем. Безпека також стає важливим аспектом у зв'язку з великою кількістю підключених пристроїв, що потенційно створює багато точок вразливості. Традиційні методи шифрування можуть не ефективно враховувати масштаби та різноманітність IoT екосистеми, що збільшує ризик несанкціонованого доступу та атак.

Отже, адаптація традиційних методів шифрування до специфіки IoT пристроїв вимагає уваги до обмежень ресурсів та урахування умов непередбачуваного середовища. Тому оптимізація алгоритмів шифрування в контексті IoT систем володіє важливим значенням через ряд специфічних викликів, які стикаються ці пристрої.

В першу чергу, врахування обмежених ресурсів IoT пристроїв вимагає вдосконалення алгоритмів для оптимального використання цих ресурсів. Оптимізовані шифрувальні методи дозволяють забезпечити високий рівень

безпеки, не перевантажуючи пристрої та не витрачаючи надмірно багато енергії, що особливо важливо в умовах обмежених акумуляторів та важкодоступних умов.

Додатково, у зв'язку з розподіленістю та гетерогенністю IoT екосистеми, оптимізація дозволяє забезпечити сумісність та ефективність шифрування в різноманітних пристроях із різними технічними характеристиками та функціональністю.

Важливим аспектом є також адаптація алгоритмів до змінливих умов середовища, таких як зміни в мережевому покритті та доступі до ресурсів. Оптимізовані шифрувальні алгоритми можуть бути більш гнучкими та адаптивними до таких змін, забезпечуючи стійкість та надійність зв'язку.

Поняття чутливості алгоритмів шифрування. Чутливість алгоритмів до змін у вхідних параметрах означає міру, до якої вони реагують на зміни вхідних значень чи умов, що впливає на їхню продуктивність чи результат. У контексті шифрувальних алгоритмів це означає, наскільки великою може бути зміна в криптографічних властивостях або ефективності алгоритму при маленьких змінах у вхідних даних, ключах чи інших параметрах.

Чутливість алгоритмів шифрування є важливим аспектом, оскільки вона визначає їхню реакцію на потенційні зміни в ситуаціях в реальному часі. Шифрувальні алгоритми повинні залишатися стійкими та ефективними навіть у випадку, коли вхідні умови чи параметри зазнають деяких невеликих змін. Врахування чутливості гарантує, що алгоритми продовжать працювати в різних умовах експлуатації, залишаючись функціональними в контексті тих завдань, які на них були покладені.

Аналіз чутливості - це процес оцінки впливу зміни вхідних параметрів на вихідні характеристики системи або проєкту. Для вдосконалення шифрувальних алгоритмів в IoT (інтернет речей) застосовуються різні методи аналізу чутливості.

Аналіз варіації – це метод, який дозволяє оцінити, як різні фактори впливають на результат шифрування. Цей метод базується на розбитті

загальної варіації результату на складові, які відповідають окремим факторам або їх комбінаціям. Таким чином, можна визначити, які фактори є суттєвими, а які ні, і як вони взаємодіють між собою. Для проведення аналізу варіації потрібно виконати наступні кроки [16]:

- 1) Визначити фактори, які можуть впливати на якість шифрування, наприклад, ключ, схема, алгоритм, протокол, атака тощо.
- 2) Визначити рівні кожного фактору, тобто можливі значення або варіанти, наприклад, довжина ключа, тип схеми, назва алгоритму, вид протоколу, метод атаки тощо.
- 3) Визначити критерій якості шифрування, тобто показник, за яким буде оцінюватися результат, наприклад, час шифрування, час розшифрування, стійкість до атак, обчислювальна складність тощо.
- 4) Провести експерименти, в яких буде змінюватися один або декілька факторів, і виміряти критерій якості для кожної комбінації факторів.
- 5) Побудувати таблицю аналізу варіації, в якій буде показано, яка частина загальної варіації критерію якості припадає на кожен фактор або їх комбінацію, і які з них є статистично значущими.
- 6) Зробити висновки про вплив факторів на якість шифрування і вибрати оптимальні значення факторів для досягнення найкращого результату.

Аналіз сценаріїв - це метод, який дозволяє прогнозувати різні можливі ситуації, що можуть виникнути в IoT, і оцінювати їх наслідки для шифрувальних алгоритмів. Цей метод допомагає передбачити потенційні ризики і вибрати найкращі стратегії захисту. Для проведення аналізу сценаріїв потрібно виконати такі кроки [4, 6, 33]:

- 1) Визначити цільові ситуації, які потрібно дослідити, наприклад, атаки на IoT-пристрої, збої в мережі, зміни в середовищі тощо.
- 2) Визначити ключові фактори, які впливають на ці ситуації, наприклад, типи і параметри IoT-пристроїв, шифрувальних алгоритмів, протоколів, атак тощо.

- 3) Визначити можливі значення кожного фактора, тобто варіанти, які можуть мати місце в реальності, наприклад, різні довжини ключів, різні методи шифрування, різні види атак тощо.
- 4) Створити сценарії, які описують різні комбінації факторів і їх значень, наприклад, сценарій, в якому IoT-пристрій з коротким ключем піддається атаці брутфорс, сценарій, в якому IoT-пристрій з довгим ключем піддається атаці бокового каналу тощо.
- 5) Оцінити наслідки кожного сценарію для шифрувальних алгоритмів, тобто визначити, як сценарій впливає на якість, ефективність і безпеку шифрування, наприклад, час шифрування, час розшифрування, стійкість до атак, обчислювальна складність тощо.
- 6) Зробити висновки про оптимальні шифрувальні алгоритми для різних ситуацій, тобто вибрати ті алгоритми, які мінімізують ризики і максимізують переваги в IoT.

Аналіз алгоритмів - це метод, який дозволяє визначити обчислювальну складність шифрувальних алгоритмів, тобто кількість часу, пам'яті чи інших ресурсів, необхідних для їх виконання. Цей метод допомагає порівнювати ефективність різних алгоритмів і вибирати найбільш оптимальні для заданих умов [4]. Для проведення аналізу алгоритмів потрібно виконати такі кроки:

- 1) Визначити вхідні дані, які будуть оброблятися алгоритмом, наприклад, розмір блоку, довжину ключа, тип схеми, назву алгоритму тощо.
- 2) Визначити вихідні дані, які будуть отримані після виконання алгоритму, наприклад, зашифрований або розшифрований текст, час шифрування або розшифрування, стійкість до атак тощо.
- 3) Визначити операції, які виконує алгоритм, наприклад, додавання, множення, перестановка, зсув, XOR тощо.
- 4) Визначити кількість операцій, які виконує алгоритм для різних розмірів вхідних даних, наприклад, за допомогою математичних формул, таблиць, графіків тощо.

- 5) Визначити асимптотичну складність алгоритму, тобто функцію, яка описує залежність кількості операцій від розміру вхідних даних при наближенні до нескінченності, наприклад, $O(n)$, $O(n^2)$, $O(\log n)$ тощо.

За підсумками виконаних кроків підводиться підсумок про ефективність алгоритму, тобто порівнюється його асимптотична складність з іншими алгоритмами, враховуються фактори, які можуть впливати на реальну швидкість виконання, такі як обладнання, мова програмування, оптимізація тощо.

Аналіз градієнтів - це метод, який використовує числові методи для визначення напрямку найшвидшого зростання або спадання цільової функції (наприклад, NPV або IRR) в залежності від вхідних параметрів. Цей метод дозволяє знаходити локальні максимуми або мінімуми цільової функції і визначати оптимальні значення параметрів для досягнення бажаного результату. Проведення аналізу градієнтів слід виконувати в такому порядку:

- 1) Визначити цільову функцію, яку треба максимізувати або мінімізувати, наприклад, NPV або IRR.
- 2) Визначити вхідні параметри, які впливають на цільову функцію, наприклад, ставку дисконту, термін проєкту, витрати, доходи тощо.
- 3) Визначити градієнт цільової функції, тобто вектор, що складається з частинних похідних функції за кожним параметром. Градієнт показує напрямок найшвидшого зростання функції в кожній точці.
- 4) Вибрати початкове наближення для параметрів, наприклад, за допомогою експертних оцінок, історичних даних, аналогів тощо.
- 5) Застосувати метод градієнтного спуску або підйому, тобто змінювати значення параметрів у напрямку протилежному або збіжному з градієнтом з певним кроком, поки не буде досягнуто критерію зупинки, наприклад, мінімальної зміни функції або параметрів, максимальної кількості ітерацій тощо.
- 6) Отримати оптимальні значення параметрів, які забезпечують максимум або мінімум цільової функції.

Аналіз Монте-Карло - це метод, який використовує випадкові числа для моделювання різних можливих ситуацій, що можуть виникнути в IoT (інтернет речей), і оцінювання їх імовірності та наслідків для шифрувальних алгоритмів. Цей метод допомагає враховувати невизначеність і ризики, пов'язані з IoT, і вибирати найбільш робастні алгоритми. Аналіз Монте-Карло слід виконувати таким чином:

- 1) Визначити цільову функцію, яку хочеться максимізувати або мінімізувати, наприклад, час шифрування, час розшифрування, стійкість до атак, обчислювальна складність тощо.
- 2) Визначити входні параметри, які впливають на цільову функцію, наприклад, типи і параметри IoT-пристроїв, шифрувальних алгоритмів, протоколів, атак тощо.
- 3) Визначити розподіли ймовірностей для кожного параметра, тобто функції, які описують, як часто параметр приймає певне значення або діапазон значень, наприклад, рівномірний, нормальний, біноміальний, Пуассонівський тощо.
- 4) Згенерувати випадкові значення для кожного параметра з відповідного розподілу, наприклад, за допомогою таблиць випадкових чисел, генераторів випадкових чисел, програмного забезпечення тощо.
- 5) Підставити випадкові значення параметрів в цільову функцію і обчислити її значення, наприклад, за допомогою математичних формул, алгоритмів, програмного забезпечення тощо.
- 6) Повторити попередні два кроки багато разів, наприклад, тисячі або мільйони разів, і зібрати всі отримані значення цільової функції.
- 7) Проаналізувати статистичні характеристики отриманих значень цільової функції, наприклад, середнє, медіану, моду, дисперсію, коефіцієнт варіації, інтервал довіри, гістограму, криву розподілу тощо.

На підставі аналізу статистичних характеристик робиться висновок про оптимальні шифрувальні алгоритми для різних ситуацій, тобто обираються ті алгоритми, які максимізують переваги і мінімізують ризики в IoT.

Для оптимізації процесу шифрування та забезпечення стійкості системи важливим є виявлення надлишкових елементів в шифрувальних алгоритмах IoT.

В одному з досліджень [27] обговорюється використання сучасних алгоритмів для підвищення ефективності якісних показників даних, що передаються. Вони засновані на використанні ансамблю таймерних сигнальних конструкцій. Це дозволяє збільшити продуктивність каналу за рахунок зменшення енергетичної відстані між сигнальними конструкціями. Також в дослідженні розглядаються методи передачі в інформаційно-комунікаційних системах при використанні традиційних способів кодування (наприклад позиційного) з надлишковими кодами. Вказується на основні недоліки позиційного кодування і пропонується метод обробки даних з використанням таймерних сигнальних конструкцій.

Аналіз алгоритмів є процесом визначення обчислювальної складності алгоритмів, тобто кількості часу, пам'яті чи інших ресурсів, необхідних для виконання алгоритмів. Це може бути корисним при виявленні надлишкових елементів в шифрувальних алгоритмах IoT. Саме тому застосування методів оптимізації для скорочення ресурсів, використовуваних шифрувальними процесами в IoT пристроях, є важливою областю дослідження. Оптимізація може допомогти зменшити використання ресурсів, таких як час процесора, пам'ять та енергія, що є критично важливим для пристроїв IoT, які часто працюють на батареях і мають обмежені обчислювальні ресурси.

Методи оптимізації призначені для знаходження екстремумів функцій [3]; ці методи можуть бути використані для вирішення задач умовної оптимізації, багатомірної оптимізації та лінійного програмування [13].

Оптимальні задачі в більшості зводяться для розв'язання типових задач оптимізації [11]. Практичне використання методів статичної оптимізації може бути корисним для оптимізації шифрувальних процесів в IoT пристроях.

1.3. Особливості розробки для IoT середовища

Зі зростанням та розвитком Інтернету речей (IoT) виникає необхідність ретельного вивчення та розуміння його особливостей. IoT як концепція передбачає взаємодію між фізичними пристроями, датчиками та різноманітними системами, створюючи цілісне середовище для обміну даними та автоматизації.

Характерною рисою IoT є розподіленість та гетерогенність його складових. Пристрої в IoT можуть відрізнятися за технічними характеристиками, мережевими підключеннями, архітектурою та функціональністю. Ця різноманітність створює складні завдання в управлінні та забезпеченні безпеки цього розподіленого середовища.

Крім того, обмежені ресурси, такі як енергопостачання, обчислювальна потужність та пам'ять, визначають умови експлуатації IoT пристроїв. Це ставить виклики перед впровадженням традиційних методів та алгоритмів, оскільки вони повинні бути адаптовані до обмежених можливостей цих пристроїв.

У такому контексті вивчення та аналіз особливостей роботи в IoT середовищі стає критично важливим для розробників, дослідників та фахівців із безпеки, щоб забезпечити ефективність, надійність та безпеку в еволюції цього динамічного та постійно зростаючого сегменту технологій.

Аспекти роботи в IoT середовищі мають свої особливості, які включають використання різних технологій та протоколів, а також виклики, пов'язані з оптимізацією ресурсів.

Технології та протоколи. В IoT системах використовуються різні технології та протоколи, такі як MQTT, CoAP, Zigbee, Wi-Fi, Bluetooth, Ethernet, стільникові мережі та LPWAN [8].

- MQTT (Message Queuing Telemetry Transport): протокол на основі TCP/IP, який використовується для передачі даних в режимі реального часу між пристроями і серверами.

- CoAP (Constrained Application Protocol): протокол, що використовується для обміну даними між пристроями в обмежених мережах, таких як IoT.
- Zigbee: стандарт бездротової мережі, який використовується для створення мереж з низьким споживанням енергії.
- Wi-Fi: загальноживаний стандарт бездротового зв'язку, який використовується для підключення пристроїв до Інтернету.
- Bluetooth: технологія бездротового зв'язку короткого радіусу дії, яка використовується для з'єднання пристроїв IoT.
- Ethernet: технологія проводового зв'язку, яка використовується для підключення пристроїв IoT до Інтернету.
- Стільникові мережі: мережі, які використовуються для підключення пристроїв IoT до Інтернету за допомогою мобільного зв'язку.
- LPWAN (Low Power Wide Area Network): тип мережі, який використовується для підключення пристроїв IoT, що вимагають низького споживання енергії та дальнього радіусу дії.

LPWAN, зокрема, отримав широку поширеність в використанні IoT-пристроїв з низьким споживанням енергії через кілька ключових причин. По-перше, цей тип мережі забезпечує великий радіус покриття при низькому споживанні енергії, дозволяючи пристроям працювати на далекій відстані від базових станцій і продовжувати життєвий цикл батареї. По-друге, LPWAN оптимізовані для передачі невеликих об'ємів даних, які часто характерні для IoT-застосувань, тим самим ефективно використовуючи доступні ресурси. Крім того, LPWAN забезпечують низькі витрати на інфраструктуру, що робить їх економічно вигідними для масового впровадження великої кількості підключених пристроїв. Ці характеристики роблять LPWAN привабливим вибором для розгортання IoT-рішень.

Оптимізація ресурсів. Оптимізація ресурсів є важливою частиною розробки IoT-проектів. Це може включати вибір правильних технологій, вирішення проблем та оптимізацію процесів [15].

Виклики. Інтернет речей (IoT) активно впроваджується в різних галузях, від промислової сфери до сільського господарства [5]. Однак, це також приносить виклики, такі як забезпечення безпеки, приватності даних та надійності системи, які вимагають уважного управління та розробки відповідних стратегій.

Безпека є одним із найбільш критичних аспектів в контексті IoT. Починаючи з самого початку, пристрої IoT були відомі своєю вразливістю до кібератак. Є безліч прикладів, коли пристрої IoT були включені в ботнети (наприклад, інфамний ботнет Mirai) або були зламані для неправильного використання або доступу до інших частин мережі [37]. Збільшення кількості підключених пристроїв створює багато точок вразливості, які можуть стати об'єктом кібератак. Забезпечення безпеки мережі та захисту від несанкціонованого доступу стає надзвичайно важливим завданням.

Ще одним важливим викликом є приватність даних, оскільки IoT пристрої збирають великі обсяги даних, які можуть включати особисту інформацію. Збільшення обсягу зібраних та оброблених даних може викликати серйозні питання стосовно захисту особистої інформації. Заходи щодо анонімізації, шифрування та етичного використання даних стають ключовими для збереження довіри користувачів.

Надійність системи також є важливою проблемою. Висока залежність від IoT пристроїв у деяких сферах, таких як медицина чи промисловість, означає, що непередбачувані відмови можуть мати серйозні наслідки. Забезпечення надійності та готовності систем є ключовим завданням для успішного впровадження IoT-технологій.

Щоб вирішити ці виклики, потрібно розробити відповідні стратегії та рішення. Наприклад, для покращення безпеки IoT можна використовувати технології шифрування, аутентифікацію та протоколи безпеки. Для забезпечення приватності даних додатково можуть використовуватися методи анонімізації та псевдонімізації, а також розроблятися політики конфіденційності, які враховують вимоги IoT. Щодо надійності системи,

важливо розробляти рішення, які можуть ефективно управляти великими обсягами даних та забезпечувати стабільне з'єднання. Вирішення цих викликів вимагає комплексного підходу, включаючи розробку безпечних протоколів зв'язку, вдосконалення стандартів безпеки, впровадження механізмів захисту даних та постійний моніторинг та апгрейд систем для підтримки їхньої надійності та безпеки.

Розробка алгоритму шифрування, який адаптується до обмежень мережі, обчислювальних можливостей та енергоефективності пристроїв, вимагає комплексного підходу. Ось декілька стратегій, які можна використати:

1. Використання легкої криптографії. Легка криптографія - це спрощена версія криптографії, яка спеціально розроблена для пристроїв з обмеженими ресурсами, таких як сенсори IoT [39]. Ці алгоритми шифрування спроектовані таким чином, щоб мінімізувати використання обчислювальних ресурсів та енергії. Ось декілька ключових аспектів її використання:

- Захист малих пристроїв. Легка криптографія призначена для захисту даних, створених та переданих маленькими пристроями, такими як сенсори IoT, медичні девайси, датчики напруження всередині доріг та мостів, а також пристрої безключового доступу до автомобілів.
- Адаптація до обмежених ресурсів. Ці пристрої потребують легкої криптографії, яка використовує їхню обмежену кількість електронних ресурсів. Це означає, що алгоритми легкої криптографії повинні бути ефективними з точки зору використання обчислювальних ресурсів та енергії.
- Стандартизація. Національний інститут стандартів та технологій (NIST) провів процес вибору алгоритмів легкої криптографії, які підходять для використання в обмежених середовищах, де продуктивність поточних криптографічних стандартів NIST не прийнятна. В результаті цього процесу було вибрано групу криптографічних алгоритмів під назвою

Ascon, які в 2023 році опубліковані як стандарт легкої криптографії NIST [29, 36].

- Використання в різних областях. Легка криптографія може бути використана в різних областях, таких як мережі сенсорів, охорона здоров'я, розподілені системи управління, кіберфізичні системи, де високо обмежені пристрої взаємодіють, зазвичай бездротово спілкуючись один з одним, і працюють разом, щоб виконати поставлене завдання.

2. Селективне шифрування. Цей підхід включає в себе шифрування лише вибраних частин даних, що зменшує обчислювальне навантаження та використання енергії. Селективне шифрування може бути особливо корисним для бездротових мереж ad hoc, де обчислювальні ресурси обмежені [41]. Селективне шифрування є відносно новим напрямком в захисті зображень та відео. Він полягає в шифруванні лише підмножини даних. Мета селективного шифрування - зменшити обсяг даних для шифрування, зберігаючи при цьому достатній рівень безпеки [32].

В традиційних схемах захисту змісту зображень та відео, які називаються повністю шаровими, спочатку весь зміст стискається. Потім стиснутий потік бітів повністю шифрується за допомогою стандартного шифру (DES, AES, IDEA тощо). Однак специфічні характеристики цього типу даних (висока швидкість передачі при обмеженій пропускну здатності) роблять стандартні алгоритми шифрування недостатніми.

Селективне шифрування дозволяє зберігати деякі функції кодека, такі як масштабованість. Це обчислювальне збереження є дуже бажаним, особливо в обмежених комунікаціях (реальний час, високої чіткості доставка, мобільні комунікації з обмеженими обчислювальними пристроями).

Таким чином, основна ідея селективного шифрування полягає в тому, щоб шифрувати інформативні підмножини даних за допомогою надійного алгоритму шифрування, в той час як інші підмножини шифруються за допомогою легшого алгоритму шифрування. Крім того, процеси вибору та

шифрування підмножин даних можуть виконуватися в частотному або просторовому домені [34].

3. Адаптивні алгоритми шифрування. Ці алгоритми можуть автоматично налаштовуватися на зміни в мережі та обчислювальному навантаженні. Наприклад, вони можуть використовувати більш складні методи шифрування, коли обчислювальні ресурси доступні, та переходити на менш складні методи, коли ресурси обмежені. Ось декілька прикладів адаптивних алгоритмів шифрування:

- Алгоритм шифрування зображень, що адаптується до зображення. Цей алгоритм використовує двовимірну хаотичну систему для створення алгоритму шифрування, який адаптується до цільового зображення. Він включає в себе адаптивний метод перестановки, який змінює значення пікселів, ефективно переставляючи позиції пікселів.
- Адаптивний швидкий алгоритм шифрування зображень. Цей алгоритм базується на тривимірній хаотичній системі і включає в себе адаптивний механізм, який може автономно визначати оптимальні стратегії шифрування на основі характеристик зображення.
- Гібридний легкий алгоритм (HLA). Цей алгоритм є комбінацією легких симетричних та асиметричних алгоритмів шифрування для пристроїв IoT. На основі існуючих досліджень багато дослідників вже розробили легкі криптографічні алгоритми [38].

4. Енергоефективні алгоритми. Деякі алгоритми шифрування спеціально розроблені для мінімізації використання енергії. Це може включати в себе використання алгоритмів з низьким використанням процесора або алгоритмів, які можуть виконувати обчислення під час періодів низької активності, щоб зменшити загальне використання енергії [30].

Енергоефективні алгоритми важливі для пристроїв IoT, оскільки вони часто працюють на батареях і мають обмежені обчислювальні ресурси. Ось декілька ключових аспектів енергоефективних алгоритмів в IoT:

- Енергоефективні алгоритми для бездротових сенсорних мереж (WSN). Багато IoT-застосунків використовують WSN, які складаються з пристроїв з обмеженими ресурсами. Через обмеженість енергії сенсорів, стратегія енергоефективності для IoT, що базується на WSN, є важливою. В цьому контексті було запропоновано багато стратегій збереження енергії [26].
- Енергоефективне проектування систем для IoT. Включає в себе виклики, пов'язані з ефективним живленням пристрою IoT, використанням нових технологій пам'яті для забезпечення енергоефективності пристроїв IoT, а також потенційний вплив приблизного обчислення на підвищення енергоефективності носимих та інших обчислювально інтенсивних пристроїв IoT [28].
- Енергоефективна стратегія для IoT (EES4IoT). Стратегія, описана в роботі [31], базується на схемі збереження енергії і вирішує проблеми з'єднання, такі як проблеми з енергією, втрату пакетів та затримку передачі в мережі.

Під час розробки алгоритму шифрування для IoT важливо провести аналіз вимог користувачів щодо безпеки та продуктивності. У сфері безпеки користувачі орієнтуються на ефективний захист конфіденційності, цілісності та доступності даних. Вимоги безпеки також можуть включати необхідність використання сучасних шифрувальних алгоритмів, стійкість до атак та можливість аутентифікації пристроїв.

З точки зору продуктивності, користувачі можуть вимагати оптимальної швидкодії та ефективного використання ресурсів пристроїв. Врахування обмежень щодо обчислювальної потужності та енергозбереження стає важливим аспектом. Користувачі можуть бажати, щоб алгоритм працював ефективно на пристроях з різними технічними характеристиками, забезпечуючи оптимальну продуктивність без значного впливу на енергоспоживання.

РОЗДІЛ 2

РОЗРОБКА АЛГОРИТМУ ШИФРУВАННЯ ДЛЯ ІОТ ПРИСТРОЇВ

2.1. Розробка алгоритму шифрування

У рамках виконання поставлених завдань дослідження, а також на основі порівняльного аналізу, виконаного в розділі 1.1, було взято за основу алгоритм еліптичних кривих ЕСС та створено модифікований алгоритм, призначений для використання в пристроях та мережах IoT.

IoT Elliptic Curve Cipher (IoTECC) - це удосконалений алгоритм шифрування, спеціально розроблений для забезпечення високого рівня безпеки в Internet of Things (IoT) системах. Його унікальність полягає в використанні методів еліптичних кривих, генерації динамічних параметрів та використанні криптографічних хеш-функцій для збільшення стійкості та надійності.

Ключові особливості **IoTECC** є такими:

1. Вибір еліптичної кривої. Вибір еліптичної кривої для алгоритму "IoT Elliptic Curve Cipher Plus (IoTECC+)" є ключовим етапом, спрямованим на забезпечення високого рівня безпеки. Обрана крива E визначає математичний фундамент для генерації ключів та виконання криптографічних операцій.

Обирається еліптична крива над скінченним полем F_p , де p - просте число. Математично, крива представляється рівнянням:

$$E: y^2 = x^3 + ax + b \mod p, \quad (2.1)$$

де a та b - коефіцієнти кривої, що визначають її форму, а $\mod p$ вказує на використання арифметики за модулем простого числа p .

Крива великого порядку ускладнює атаки з використанням методу «перебору ключа». Крива повинна мати достатньо велике просте число p , щоб запобігти атакам, заснованим на факторизації. Крім того, параметри a та b мають бути обрані з урахуванням вимог до стійкості та безпеки. Вони визначають конкретну форму кривої, і вибір неправильних параметрів може призвести до вразливостей.

Враховуючи ці математичні вимоги та обмеження, вибір еліптичної кривої в IoTESS визначається як стратегічний крок для забезпечення надійності та стійкості алгоритму в умовах IoT систем.

2. Генерація динамічного параметра. Цей етап додає рівень безпеки, сприяючи запобіганню певним видам атак та підвищуючи витривалість криптографічного захисту.

На цьому етапі перед генерацією ключів обирається випадковий параметр d_{param} , який використовується для модифікації стандартних параметрів еліптичної кривої, надаючи алгоритму додатковий ступінь випадковості. Математично, введення динамічного параметра виражено формулою 2.2:

$$Q = H_1(k, P, d_{param}) \cdot P, \quad (2.2)$$

де H_1 - хеш-функція, яка використовується для змішування випадкового параметра d_{param} з приватним ключем k та точкою P еліптичної кривої. Використання d_{param} у функції хешування дозволяє адаптувати параметри кривої для кожного конкретного випадкового вибору, що додає додатковий ступінь ускладнення для потенційних злоумисників. Такий підхід до генерації параметрів підвищує безпеку алгоритму, забезпечуючи стійкість проти різних видів атак в умовах IoT систем.

3. Шифрування. На етапі шифрування алгоритму використовується еліптична крива для генерації випадкової точки, яка виступає як тимчасовий ключ шифрування. Цей етап включає в себе криптографічні операції, які забезпечують конфіденційність інформації.

Обирається випадкове число r , і точка $R = r \cdot P$, обчислюється на еліптичній кривій, де P – публічний ключ отримувача. Ця випадкова точка R слугує тимчасовим ключем і використовується для зашифрування повідомлення.

Шифротекст C обчислюється за допомогою операції побітового додавання (XOR) між оригінальним повідомленням M та результатом застосування хеш-функції H_2 до точки R :

$$C = M \oplus H_2(R), \quad (2.3)$$

де H_2 - інша хеш-функція, що використовується для збільшення стійкості алгоритму. Застосування хеш-функції до точки R додає елемент непередбачуваності до шифрування, оскільки визначення точки R важке для передбачення.

Цей підхід до шифрування в IoTECC дозволяє ефективно захищати конфіденційність інформації, використовуючи принципи еліптичної криптографії та хеш-функцій для забезпечення надійності і стійкості шифрування в умовах Internet of Things.

4. Дешифрування. На етапі дешифрування отримувач використовує свій приватний ключ для розрахунку тимчасового ключа R , який був використаний при шифруванні. Цей етап гарантує відновлення оригінального повідомлення з шифротексту.

Отримувач використовує свій приватний ключ d для розрахунку тимчасового ключа R за формулою $R = d \cdot P$, де P - публічний ключ відправника, що використовувався для шифрування. Цей крок забезпечує здатність отримувача відновлювати тимчасовий ключ, який був використаний у процесі шифрування.

Після отримання тимчасового ключа R отримувач використовує його для відновлення оригінального повідомлення M . Це виконується за допомогою операції побітового додавання (XOR) між шифротекстом C та результатом застосування хеш-функції H_2 до отриманого тимчасового ключа R :

$$M = C \oplus H_2(R), \quad (2.4)$$

Цей процес дозволяє отримувачу відновити оригінальне повідомлення, використовуючи свій приватний ключ та тимчасовий ключ, який був

використаний при шифруванні. Використання хеш-функції H_2 сприяє підтвердженню цілісності отриманих даних.

Алгоритм було розроблено на основі вивчення сучасних вимог до безпеки IoT систем. Введення динамічного параметра та використання хеш-функцій підвищують стійкість до атак, таких як криптографічний аналіз та атаки з використанням фіксованих параметрів. IoTESS+ забезпечує високий рівень конфіденційності та цілісності даних в контексті IoT, враховуючи сучасні виклики та загрози в цій області.

2.2. Проєктування та розробка вбудованого програмного забезпечення IoT пристрою

Вбудоване програмне забезпечення (ПЗ) IoT пристрою - це спеціалізоване програмне забезпечення, яке безпосередньо вбудоване в апаратну частину IoT пристрою та відповідає за управління його функціями та операціями. Це програмне забезпечення виконує ключові завдання, такі як обробка даних, збір інформації з різних датчиків, взаємодія з мережею, а також виконання конкретних завдань, передбачених функціональністю пристрою.

Завдяки вбудованому програмному забезпеченню, IoT пристрій може ефективно взаємодіяти з іншими пристроями та системами, забезпечуючи обмін даними і виконання визначених завдань без значного завантаження централізованих серверів чи хмарних платформ. Це є запорукою продуктивності та реагування пристрою на змінні умови.

Реалізація вбудованого програмного забезпечення для IoT пристрою є важливою для гарантії надійності й ефективності його роботи, а також для забезпечення безпеки та конфіденційності оброблюваних даних. Вона дозволяє пристрою виконувати свої функції автономно та на постійній основі взаємодіяти з іншими складовими IoT-екосистеми.

У рамках дослідження була обрана архітектура вбудованого програмного забезпечення IoT пристрою, яка складається з таких елементів:

1. **Модуль керування даними (Data Management Module):** відповідає за збір, зберігання та передачу даних з сенсорів або інших джерел даних.
2. **Модуль шифрування (Encryption Module):** використовує обраний алгоритм шифрування для шифрування даних перед їх передачею.
3. **Модуль комунікації (Communication Module):** відповідає за встановлення та управління з'єднаннями мережі для передачі даних до сервера або інших IoT пристроїв.
4. **Модуль управління живленням (Power Management Module):** відповідає за оптимізацію використання енергії та управління живленням пристрою.
5. **Модуль безпеки (Security Module):** відповідає за захист пристрою від зовнішніх загроз та атак.

Взаємодія модулів між одне одним відбувається у встановленому порядку, який складає основу архітектури вбудованого ПЗ пристрою розумного термостату і показаний на рис. 2.1.

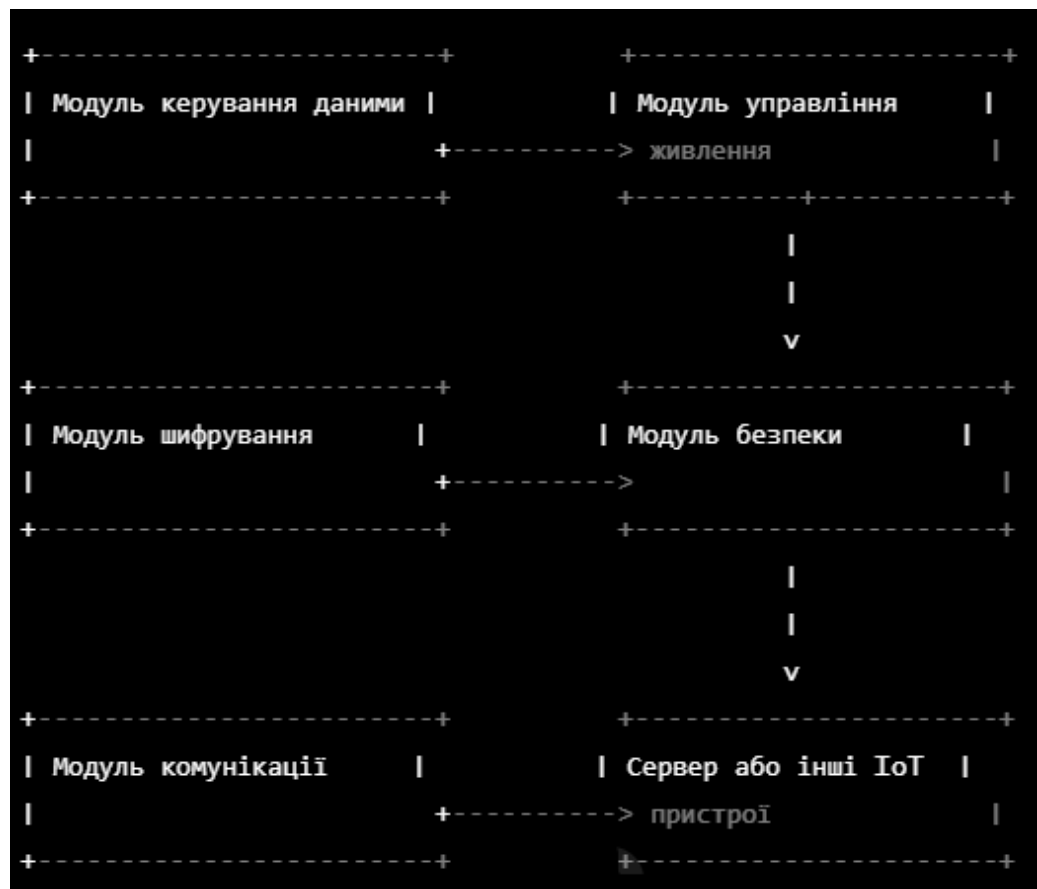


Рис. 2.1 - Архітектура вбудованого ПЗ пристрою IoT

Відповідно до схеми, відбувається така взаємодія модулів:

- 1) Модуль керування даними збирає дані та передає їх модулю шифрування.
- 2) Модуль шифрування шифрує дані та передає їх модулю комунікації.
- 3) Модуль комунікації передає зашифровані дані до сервера або інших IoT пристроїв.
- 4) Модуль управління живленням контролює використання енергії всіма іншими модулями.
- 5) Модуль безпеки надає захист для всіх інших модулів та даних, які вони обробляють.

Модуль керування даними (Data Management Module) відповідає за збір, обробку та управління даними, які надходять від різних датчиків та джерел в пристрої. Його основною функцією є організоване зберігання інформації перед її подальшою обробкою та передачею до інших модулів системи. Модуль керування даними включає в себе ряд наступних ключових елементів, які були реалізовані в системі.

Збір даних. Функція `collectData()` для збору даних з сенсорів в Arduino використовує вбудовані функції читання аналогових або цифрових входів. Вона зчитує значення з піна, який підключений до сенсора, та повертає це значення (див. рис. 2.2).

```
// Оголошення константи для пінів датчика
const int sensorPin = A0; // Пін для аналогового датчика

// Функція для збору даних від датчиків та інших джерел
float collectData() {
    // Зчитування даних з датчика або іншого джерела
    int sensorValue = analogRead(sensorPin);

    // Перетворення зчитаних значень на реальні одиниці (залежить від конкретного датчика)
    float realData = map(sensorValue, 0, 1023, 0, 100); // Mapування значень на діапазон

    // Повернення отриманих даних
    return realData;
}
```

Рис. 2.2 – Реалізація функції збору даних у модулі керування

Представлений код для платформи Arduino реалізує простий збір даних від аналогового датчика. Функція `collectData` зчитує значення з аналогового піну, перетворює його та повертає у вигляді плаваючого числа. У функції `loop` ці дані виводяться на монітор порту для налагодження та можуть бути використані для подальшої обробки або відправки. Код бути адаптований до конкретних датчиків та вимог при змінах специфікацій проєкту. Також в даному коді використовується функція `delay(1000)`, що призводить до паузи у виконанні коду на 1 секунду.

Фільтрація та обробка даних. Функція `filterAndProcessData()` відіграє важливу роль в системі обробки даних. На рис. 2.3 показаний приклад відсіювання зайвих даних та підготовки отриманих даних для подальшої обробки та аналізу.

```
float filterAndProcessData(float rawData, float lowerLimit, float upperLimit) {  
    // Відсіювання зайвих значень за межами діапазону  
    if (rawData < lowerLimit || rawData > upperLimit) {  
        // Значення знаходиться за межами допустимого діапазону, можна вважати його невалідним  
        return -1.0; // Повернення спеціального значення для позначення невалідних даних  
    }  
  
    // Обробка залишених даних (в даному випадку повертаємо саме значення)  
    return rawData;  
}  
  
void setup() { ...  
}  
  
void loop() {  
    float rawData = collectData(); // Отримані дані  
  
    // Виклик функції для фільтрації та обробки даних  
    float processedData = filterAndProcessData(rawData, 50.0, 100.0);  
  
    // Виведення на монітор порту  
    Serial.begin(9600);  
    Serial.print("Оброблені дані: ");  
    Serial.println(processedData);  
  
    // Затримка для управління частотою обробки даних  
    delay(1000);  
}
```

Рис. 2.3 – Реалізація функції фільтрації та обробки даних

Функція `filterAndProcessData()` визначає діапазон допустимих значень та відсіює й обробляє дані, які не відповідають цьому діапазону. Також,

використовуючи спеціальне значення -1.0, функція може позначити, що дані є невалідними.

Основна структура коду включає в себе ініціалізацію та налаштування, а також цикл loop, в якому викликається функція для фільтрації та обробки даних. Оброблені дані виводяться на монітор порту для налагодження та можуть бути використані для подальшої обробки або відправки. Цей код може також може використовуватися як основа для побудови адаптованих функцій фільтрації й обробки у випадку змін вимог до проєкту.

Організація даних. Функція `organizeData()` модуля керування даними відповідає за структурування та організацію даних. призначена для організації даних, що надходять від датчиків, у форматі JSON. Її код наведений на рис. 2.4.

```
#include <ArduinoJson.h> // Підключення бібліотеки ArduinoJson

// Розмір буфера для організації JSON-структури (залежить від обсягу даних)
const size_t bufferSize = JSON_OBJECT_SIZE(3);

// Функція для організації даних
void organizeData(float sensorData1, float sensorData2, int sensorData3) {
    // Створення об'єкта JSON-структури
    StaticJsonDocument<bufferSize> jsonDocument;

    // Додавання даних до JSON-структури
    jsonDocument["Sensor1"] = sensorData1;
    jsonDocument["Sensor2"] = sensorData2;
    jsonDocument["Sensor3"] = sensorData3; ...
    String jsonString;
    serializeJson(jsonDocument, jsonString);

    // Виведення організованих даних на монітор порту або інше використання
    Serial.begin(9600);
    Serial.print("Організовані дані: ");
    Serial.println(jsonString);
}
```

Рис. 2.4 – Реалізація функції організації даних

Представлена функція використовує бібліотеку `ArduinoJson` для створення та роботи з JSON-структурою. Функція отримує дані від датчиків (представлені у вигляді `float` та `int` значень), створює JSON-структуру, додає ці

дані до неї, конвертує структуру у рядок та виводить результат на монітор порту чи може бути використана для інших операцій обробки чи відправки.

Цей код може бути використаний для організації та представлення даних в структурованій формі, що полегшує їх подальший обмін, зберігання чи відображення. Проте слід зазначити, що розмір буфера JSON-структури (bufferSize) повинен бути належним чином визначений в залежності від обсягу даних та їх складності.

Керування пам'яттю. Управління пам'яттю на платформі Arduino реалізоване шляхом управління виділенням та звільненням динамічної пам'яті. Ці ресурси використовуються з урахуванням специфіки мікроконтролерів з обмеженими ресурсами. На рис. 2.5 подано приклад програмного коду для управління пам'яттю на Arduino, використовуючи malloc() для виділення пам'яті та free() для її звільнення.

```
// Розмір буфера для динамічного виділення пам'яті (приклад)
const int bufferSize = 100;

// Вказівник на динамічно виділену пам'ять
int *dynamicMemory;

// Функція для роботи з динамічною пам'яттю
void manageMemory() {
    // Виділення пам'яті
    dynamicMemory = (int *)malloc(bufferSize * sizeof(int));

    if (dynamicMemory == NULL) {
        // Помилка виділення пам'яті
        Serial.begin(9600);
        Serial.println("Помилка виділення пам'яті");
    } else {
        // Використання виділеної пам'яті (ініціалізація значень)
        for (int i = 0; i < bufferSize; i++) {
            dynamicMemory[i] = i;
        }

        // Звільнення пам'яті після використання
        free(dynamicMemory);
        // Установка вказівника в NULL для уникнення використання вільної пам'яті
        dynamicMemory = NULL;
    }
}
```

Рис. 2.5 – Реалізація ефективного керування пам'яттю в системі

Представлений код для платформи Arduino демонструє управління пам'яттю за допомогою функцій `malloc()` та `free()`. Головна мета коду - ефективно виділення та звільнення динамічної пам'яті для оптимізації використання ресурсів та запобігання переповненню пам'яті.

У функції `manageMemory` відбувається наступне:

- 1) Виділення пам'яті для масиву `dynamicMemory` за допомогою `malloc()`.
- 2) Перевірка успішності виділення пам'яті.
- 3) Ініціалізація значень в залежності від конкретного використання.
- 4) Звільнення пам'яті за допомогою `free()` після використання.

Після звільнення пам'яті вказівник `dynamicMemory` встановлюється в `NULL`, щоб уникнути можливого використання вільної пам'яті. Крім того, обробка помилок вразлива на невдалий виклик `malloc()` виводить повідомлення про помилку.

Валідація та корекція даних. В системі валідації та корекції даних важливо визначити механізми для перевірки достовірності інформації та вжиття заходів для корекції помилок чи неправильних значень.

Система валідації та корекції даних в цьому компоненті реалізована таким чином, щоб, по-перше, підтримувати в режимі реального часу процес перевірки, а саме чи відповідають вхідні дані встановленим критеріям або стандартам. Наприклад, можна перевірити, чи відповідають дані певному діапазону значень, чи мають вони правильний формат, чи не містять вони неприпустимих символів тощо.

Якщо виявлено помилку або неправильне значення, система вживає заходів для її виправлення. Це включає заміну неправильних значень на типові або видалення неправильних записів. В подальшому планується інтегрувати використання алгоритмів машинного навчання для прогнозування правильних значень на основі інших даних.

На рис. 2.6 наведений фрагмент коду для платформи Arduino, який реалізує систему валідації та корекції даних. Функція `validateAndCorrectData`

служить для перевірки достовірності отриманих даних та вжиття заходів для корекції у випадку, якщо значення не відповідає допустимому діапазону.

Описаний процес валідації є автоматизованим та дозволяє системі автоматично перевіряти та коригувати дані в реальному часі.

Основна функціональність включає в себе наступне:

- 1) Перевірка, чи значення `rawData` знаходиться в межах визначеного діапазону (`validRangeMin` до `validRangeMax`).
- 2) У випадку, якщо значення знаходиться поза межами допустимого діапазону, застосування корекцій, наприклад, приведення до середнього значення діапазону.
- 3) Повернення валідованих або скоригованих даних.

```
// Функція для валідації та корекції даних
float validateAndCorrectData(float rawData, float validRangeMin, float validRangeMax) {
    // Перевірка, чи дані знаходяться у межах допустимого діапазону
    if (rawData < validRangeMin || rawData > validRangeMax) {
        // Значення не відповідає допустимому діапазону, можна вважати його недійсним
        // Вжиття заходів для корекції (в даному випадку, приведення до середнього значення)
        rawData = (validRangeMin + validRangeMax) / 2.0;
    }

    // Повернення валідованих або скорегованих даних
    return rawData;
}

void loop() {

    float rawData = collectData(); //
    float validRangeMin = 0.0;
    float validRangeMax = 100.0;

    // Виклик функції для валідації та корекції даних
    float validatedData = validateAndCorrectData(rawData, validRangeMin, validRangeMax);

    // Диведення на монітор порту
    Serial.begin(9600);
    Serial.print("Валідовані дані: ");
    Serial.println(validatedData);

    // Затримка для управління частотою обробки даних
    delay(1000);
}
```

Рис. 2.6 – Валідація й корекція даних, отриманих від датчиків

Цей підхід є значущим для забезпечення надійності та точності отриманих даних у вбудованому ПЗ пристрою IoT. Важливо враховувати, що механізми валідації та корекції повинні відповідати конкретним вимогам та характеристикам проєкту, тому функція має бути адаптивною або оперативно оновлюватися при зміні специфікацій.

Модуль шифрування (Encryption Module) в системі забезпечення валідації та корекції даних відповідає за захист конфіденційності та цілісності інформації. Його основним призначенням є застосування розробленого в розділі 2.1 криптографічного алгоритму для шифрування й розшифрування даних, щоб унеможливити несанкціонований доступ чи зміну даних. Враховуючи особливості Arduino, в модулі шифрування були реалізовані низка функцій, які описані далі.

Генерація ключів еліптичної кривої. На основі бібліотеки Crypto та ECCX08 була реалізована функція `generateEllipticCurveKeys`. Для її використання застосовується вбудований криптографічний прискорювач ATSHA204A, який забезпечує апаратну реалізацію генерації ключів на основі еліптичних кривих. На рис. 2.7 показано реалізацію цієї функції.

```
#include <Wire.h>
#include <ECCX08.h>

// Оголошення об'єкту для роботи з криптографічним прискорювачем
ECCX08Class ecc;

// Функція для генерації пари приватного та публічного ключів на еліптичній кривій
void generateEllipticCurveKeys() {
    Serial.begin(9600);

    // Ініціалізація криптографічного прискорювача
    if (!ecc.begin()) {
        Serial.println("Помилка ініціалізації криптографічного прискорювача");
        while (1);
    }

    // Генерація ключів
    PrivateKey privKey;
    ecc.genKey(&privKey);

    // Отримання публічного ключа
    PublicKey pubKey;
    privKey.getPublicKey(&pubKey);
}
```

Рис. 2.7 – Реалізація функції для генерації публічного та приватного ключа

У цьому фрагменті коду функція `generateEllipticCurveKeys` ініціалізує криптографічний прискорювач та генерує пару приватного та публічного ключів на еліптичній кривій. Результати виводяться через порт монітора для спостереження. Це відбувається в нижчезазначеному порядку:

- 1) Ініціалізація криптографічного прискорювача: викликається метод `begin` об'єкту `ecc`, що представляє криптографічний прискорювач. У випадку успішної ініціалізації продовжується виконання коду, інакше програма зупиняється.
- 2) Генерація приватного ключа: викликається метод `genKey`, який генерує випадковий приватний ключ на еліптичній кривій.
- 3) Отримання публічного ключа: Викликається метод `getPublicKey` для отримання відповідного публічного ключа з випадково згенерованого приватного ключа.

Шифрування повідомлення. Функція призначена для шифрування повідомлення за допомогою алгоритму еліптичного шифрування `IoT ECC`. Вона отримує рядок `message`, який слід зашифрувати, та публічний ключ `publicKey` отримувача. Як показано на рис. 2.8, результат шифрування зберігається у вихідному параметрі `encryptedMessage` у вигляді пари, де перший елемент - точка `R`, а другий - зашифроване повідомлення `C`.

```
#include <Crypto.h>
#include <IoT ECC.h>

void encryptMessage(const String& message, const Point& publicKey,
    Pair<Point, String>& encryptedMessage) {
    // Перетворення рядка в байтовий масив
    byte* messageBytes = (byte*)message.c_str();

    // Створення об'єкту для шифрування з використанням еліптичних кривих
    IoT ECC<ECF, SHA256> iotecc;

    // Шифрування повідомлення
    iotecc.encrypt(publicKey, messageBytes, message.length(), encryptedMessage);
}
```

Рис. 2.8 – Реалізація функції шифрування з використанням оптимізованого алгоритму

Типово вбудована бібліотека Arduino не надає інтерфейсу для використання алгоритмів еліптичного шифрування, таких як ЕСС, безпосередньо на мікроконтролерах Arduino, оскільки такі алгоритми вимагають високу обчислювальну потужність та можливості для роботи з більшими числовими значеннями. Але, оскільки алгоритм ІоТЕСС, що був розроблений у рамках дослідження, оптимізований для IoT під управлінням Arduino, функція encryptMessage() використовує його з еліптичними кривими та хеш-функцією SHA256.

Дешифрування повідомлення. Реалізується за допомогою функції decryptMessage(), спрямованої на дешифрування зашифрованого повідомлення, яке було зашифроване за допомогою алгоритму ІоТЕСС. Функція отримує вхідні дані у вигляді пари, де перший елемент представляє точку R, а другий - зашифроване повідомлення C. Також в якості параметра передається приватний ключ privateKey, який використовується для дешифрування (див. рис. 2.9).

```
#include <ECC.h>

class IoTECC_Decryptor {
private:
    PrivateKey privateKey;
public:
    IoTECC_Decryptor(const PrivateKey& key) : privateKey(key) {}

    // Метод для дешифрування повідомлення, отриманого за допомогою ІоТЕСС
    String decrypt(const Pair<Point, String>& encryptedMessage) {
        // Отримання координат точки R
        const Point& R = encryptedMessage.first;

        // Використання приватного ключа для відновлення спільного секрету
        Point sharedSecret = R * privateKey;

        // Отримання розшифрованого тексту C за допомогою спільного секрету
        String decryptedText = encryptedMessage.second;
        for (size_t i = 0; i < decryptedText.length(); i++) {
            decryptedText[i] ^= sharedSecret[i % sharedSecret.length()];
        }

        return decryptedText;
    }
};
```

Рис. 2.9 – Реалізація дешифрування в кастомній бібліотеці ІоТЕСС.h

Інтерфейс функції передбачає використання внутрішньої бібліотеки IoTECC.h, яка визначає методи для дешифрування. Результат дешифрування, тобто розшифроване повідомлення, зберігається у вихідному параметрі decryptedMessage.

Клас IoTECC_Decryptor отримує приватний ключ при створенні об'єкта та має метод decrypt(), який використовується для дешифрування зашифрованого повідомлення. Сам метод використовує координати точки R, яка є частиною зашифрованого повідомлення. Застосовуючи приватний ключ, він відновлює спільний секрет, а потім використовує його для дешифрації текстової частини повідомлення (C). Отриманий розшифрований текст повертається як результат виклику методу.

Валідація параметрів еліптичної кривої. Функція validateEllipticCurveParameters() відповідає за перевірку коректності параметрів еліптичної кривої. Це важливий крок, оскільки некоректні параметри можуть призвести до небезпечних вразливостей в системі шифрування. На рис. 2.10 показана реалізація цієї функції.

```
bool validateEllipticCurveParameters(int a, int b, int p, Point G, int n, int h) {
    // Перевірка, що  $4a^3 + 27b^2 \neq 0$  (щоб крива була не виродженою)
    if ((4 * pow(a, 3) + 27 * pow(b, 2)) % p == 0) {
        return false;
    }

    // Перевірка, що p є простим числом
    if (!isPrime(p)) {
        return false;
    }

    // Перевірка, що G належить кривій
    if (!isOnCurve(G, a, b, p)) {
        return false;
    }

    // Перевірка, що  $nG = 0$ 
    if (!isZero(multiplyPoint(G, n, a, p), p)) {
        return false;
    }

    // Перевірка, що  $h > 1$  та  $hG = 0$ 
    if (h <= 1 || !isZero(multiplyPoint(G, h, a, p), p)) {
        return false;
    }

    return true;
}
```

Рис. 2.10 – Функція валідації еліптичної кривої

Функція приймає на вхід параметри, які характеризують еліптичну криву та її властивості. Вони включають:

- a, b : коефіцієнти еліптичної кривої для рівняння $y^2 = x^3 + ax + b$.
- p : просте число, яке визначає поле, над яким визначена крива.
- G : генераторна точка кривої, яка є точкою на кривій, використовуваною для генерації інших точок.
- n : порядок генераторної точки, тобто кількість точок на кривій, яку можна отримати, додавши генераторну точку до себе багато разів.
- h : коефіцієнт кратності, який вказує на кількість точок на кривій, обумовлену генераторною точкою.

В коді використовується кілька допоміжних функцій, таких як `is_prime`, `is_on_curve`, `is_zero` та `multiply_point`, які відповідно перевіряють еліптичну криву за такими критеріями:

- 1) Невиродженість кривої. Валідація перевіряє, чи еліптична крива є невинродженою. Якщо перевірка невдається, це може означати, що використовувана крива є винродженою, що в свою чергу може призвести до небезпеки атак або проблем при реалізації криптографічних алгоритмів.
- 2) Непросте число p . Якщо p не є простим числом, це порушує важливу властивість безпечних еліптичних криптосистем.
- 3) Неправильна генераторна точка G . Якщо генераторна точка G не належить еліптичній кривій, це може призвести до проблем з аутентифікацією та безпекою криптосистеми.
- 4) Проблеми з порядком та кратністю. Невірний порядок n або кратність h може вплинути на безпеку криптосистеми, зокрема, на розрахунок криптографічних ключів.

У випадку, коли хоча б одна з умов допоміжних функцій виконується, `validateEllipticCurveParameters()` повертає результат `false`, який сигналізує про невдалу валідацію еліптичної кривої. Якщо валідація зазнає невдачі, це

може вказувати на можливість атак або використання системи, яка не відповідає криптографічним вимогам.

Хешування. Хеш-функції відіграють ключову роль в забезпеченні цілісності та безпеки інформації. Однією з важливих характеристик хеш-функцій є їхня здатність швидко перетворювати вхідні дані будь-якого розміру в фіксований вихідний хеш-код. Цей хеш-код служить унікальним відображенням вхідних даних та використовується для перевірки цілісності даних або для забезпечення безпеки шляхом зберігання захешованих паролів та інших конфіденційних інформаційних об'єктів.

Для реалізації хеш-функції в модулі шифрування проєкту було обрано алгоритм хешування SHA-256, який використовується для створення 256-бітних хеш-кодів. Реалізація такої функції виглядає, як показано на рис. 2.11.

```
std::string hashFunction(const std::string& inputData) {  
    // Визначення буфера для зберігання хеш-коду  
    unsigned char hash[SHA256_DIGEST_LENGTH];  
  
    // Використання функції SHA-256 для хешування вхідних даних  
    SHA256_CTX sha256;  
    SHA256_Init(&sha256);  
    SHA256_Update(&sha256, inputData.c_str(), inputData.length());  
    SHA256_Final(hash, &sha256);  
  
    // Перетворення байтового масиву в рядок для зручності використання  
    std::string hashedString;  
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++) {  
        char hex[3];  
        sprintf(hex, "%02x", hash[i]);  
        hashedString += hex;  
    }  
  
    return hashedString;  
}
```

Рис. 2.11 – Реалізація хешування за допомогою SHA-256

Функція хешування hashFunction() використовує алгоритм SHA-256 для створення хеш-коду вхідних даних. Вона приймає рядок inputData як вхід та повертає рядок, який представляє собою вихідний хеш у вигляді шістнадцяткового рядка.

Алгоритм використовує бібліотеку OpenSSL для виклику функцій SHA-256. Спочатку функція ініціалізує контекст SHA-256, потім оновлює його з вхідними даними та завершує обчислення хешу. Результат представляється у вигляді байтового масиву, який потім перетворюється в шістнадцятковий рядок для зручності використання та подальшого зберігання чи порівняння. Функція повертає отриманий хеш-код у вигляді рядка.

Модуль комунікації (Communication Module). Модуль комунікації в даній системі відіграє ключову роль у передачі інформації між різними компонентами системи та зовнішніми пристроями. Його основною відповідальністю є забезпечення надійного та безпечного обміну даними між IoT пристроями та сервером.

Функціональність модуля комунікації включає в себе взаємодію з різними пристроями та серверами для передачі інформації, а також забезпечення захисту даних під час їх передачі. Модуль може використовувати різні протоколи комунікації, такі як HTTP, MQTT або CoAP, залежно від вимог конкретної системи.

Однією з ключових функцій модуля комунікації є відправка зашифрованих даних до сервера та інших IoT пристроїв, що забезпечує конфіденційність та цілісність інформації. Також модуль може обробляти отримані дані, розпізнавати команди та відповіді, а також здійснювати керування потоком даних.

Важливим аспектом роботи модуля комунікації є виявлення та вирішення можливих проблем під час передачі даних, таких як втрати з'єднання чи колізії. Такі сценарії можуть включати механізми автоматичного відновлення з'єднання чи повторної відправки даних.

Реалізацію вищезазначених завдань модуля забезпечують такі функції:

- **sendDataToServer(data: string).** Ця функція відповідає за відправку даних на сервер. Вона приймає рядок data як параметр і передає його серверу за допомогою визначеного протоколу комунікації (наприклад, HTTP чи

MQTT). Функція також може включати механізми обробки помилок та відновлення з'єднання у випадку невдачної відправки.

– **receiveDataFromDevice(deviceID: string).** Ця функція призначена для отримання даних від іншого IoT пристрою за його ідентифікатором deviceID. Вона ініціює процес отримання даних та повертає рядок, що містить отриману інформацію. Функція може включати перевірки безпеки для забезпечення конфіденційності та цілісності отриманих даних.

– **establishSecureConnection(serverAddress: string).** Відповідає за встановлення безпечного з'єднання з сервером, використовуючи визначений протокол шифрування (наприклад, TLS/SSL). Вона приймає адресу сервера serverAddress та повертає булеве значення, яке вказує на успішність встановлення з'єднання. Функція може включати механізми автентифікації та обміну ключами для забезпечення безпеки комунікації. Так, на рис. 2.12 зображений фрагмент коду функції, де встановлюється SSL-з'єднання.

```
// Створення сокету та його підключення до сервера
int socketDescriptor = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(443); // Приклад: HTTPS порт
inet_pton(AF_INET, serverAddress.c_str(), &server.sin_addr);

if (connect(socketDescriptor, (struct sockaddr*)&server, sizeof(server)) == -1) {
    std::cerr << "Connection to server failed." << std::endl;
    SSL_free(ssl);
    SSL_CTX_free(ctx);
    return false;
}

// Прикріплення SSL до сокету
SSL_set_fd(ssl, socketDescriptor);

// Виконання SSL Handshake
if (SSL_connect(ssl) != 1) {
    std::cerr << "SSL handshake failed." << std::endl;
    SSL_free(ssl);
    SSL_CTX_free(ctx);
    close(socketDescriptor);
    return false;
}

// Успішне встановлення безпечного з'єднання
std::cout << "Secure connection established." << std::endl;
```

Рис. 2.12 - Встановлення безпечного з'єднання за допомогою OpenSSL

У випадку будь-якої помилки, функція виводить відповідне повідомлення та повертає false. Успішне встановлення з'єднання виводить повідомлення та повертає true.

Модуль управління живленням (Power Management Module) – відіграє ключову роль у керуванні та оптимізації використання енергії всіма компонентами системи. Основна мета цього модуля - забезпечити ефективне використання живлення, щоб продовжити термін служби батарей або інших джерел живлення та підтримувати стійкість роботи пристроїв у різних умовах.

Функціональність модуля управління живленням включає в себе такі аспекти:

1. **Моніторинг енергоспоживання:** відстежуються витрати енергії кожного компонента системи, визначає їхні потреби у живленні та забезпечує оптимальний режим роботи.
2. **Управління енергозбереженням:** приймається рішення про включення або вимкнення окремих компонентів або режимів, які споживають енергію, щоб забезпечити раціональне використання енергії в залежності від поточних умов і завдань.
3. **Керування живленням за допомогою алгоритмів:** модуль використовує алгоритми керування енергією, такі як алгоритми зменшення споживання енергії в періоди неактивності або збільшення його в періоди активності, щоб досягти балансу між продуктивністю та енергоефективністю.
4. **Обробка подій від елементів живлення:** модуль реагує на події, пов'язані зі станом енергопостачання, такі як низький заряд батареї або відсутність живлення, і вживає відповідних заходів для збереження даних та забезпечення безпеки системи.
5. **Керування режимами сну та пробудження:** модуль може активувати режими сну для компонентів системи, які не використовуються, зменшуючи їхнє енергоспоживання, і виводити їх із сну при необхідності.

У даній системі модуль управління живленням представлений у вигляді вбудованого програмного компонента, який має доступ до низки параметрів енергоспоживання та впливає на роботу інших модулів. У розумному термостаті використовується мікроконтролер Arduino Nano 33 IoT, представлений на рис. 2.13.

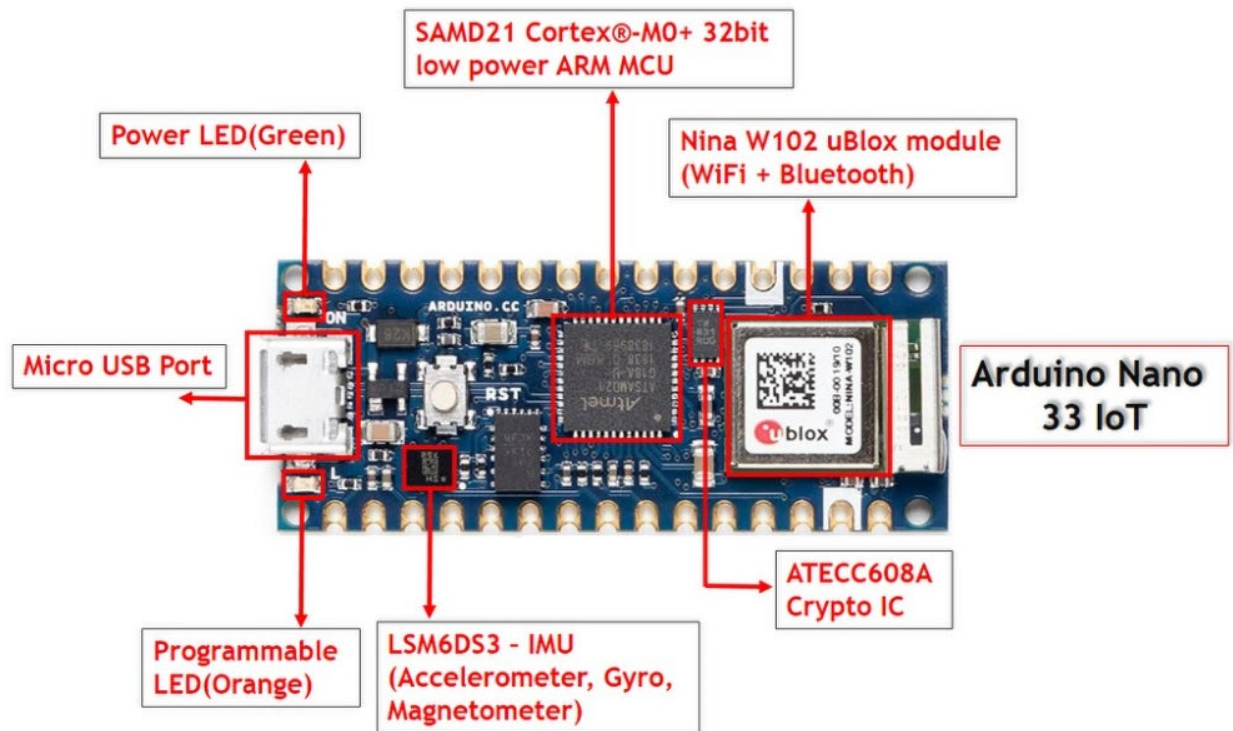


Рис. 2.13 – Схема мікроконтролера Arduino Nano 33 IoT [35]

Ця схема показує загальну архітектуру плати та взаємозв'язки між основними компонентами Основні компоненти, показані на цій схемі:

- Мікроконтролер NANO SAMD21M: Головний мікроконтролер, який виконує програму Arduino.
- Модуль Wi-Fi u-blox NINA-W102: Забезпечує зв'язок Wi-Fi для плати.
- Крипто-чіп ECC608A: Додає апаратне прискорення для криптографічних операцій.
- Регулятор PMEG6020: Регулює вхідну напругу та забезпечує стабільну напругу 3.3 В для живлення плати.
- USB-роз'єм: Подає живлення та використовується для програмування плати.

- Піни GPIO: Загального призначення - можна їх використовувати для підключення різноманітних датчиків та зовнішніх пристроїв.
- Світлодіоди: Один світлодіод живлення і два світлодіоди програмування.

Таким чином, модуль управління живленням в розумному термостаті забезпечує інтелектуальне керування енергоспоживанням, що є ключовою частиною ефективності та тривалості роботи пристрою. Завдяки вбудованим алгоритмам оптимізації та реагуванню на події, такі як низький заряд батареї чи відсутність живлення, термостат може ефективно використовувати енергію, забезпечуючи стабільну та безперебійну роботу, а також продовжуючи термін служби живлення. Це покращує взаємодію з іншими IoT пристроями та сприяє сталому та надійному функціонуванню розумного термостата в умовах різноманітних експлуатаційних сценаріїв.

Модуль безпеки (Security Module). В автономній системі, такий як розумний термостат, невід'ємною складовою частиною для забезпечення конфіденційності, цілісності та доступності даних є модуль безпеки. Основною метою цього модуля є захист від потенційних загроз та забезпечення безпеки взаємодії між пристроями та інфраструктурою IoT:

1) **Аутентифікація та авторизація.** Модуль реалізує механізми аутентифікації для перевірки ідентичності пристроїв та користувачів, які намагаються отримати доступ до системи. Крім того, встановлюються правила авторизації для контролю доступу до різних функцій та ресурсів системи.

2) **Виявлення та запобігання атак.** Модуль має алгоритми для виявлення потенційно шкідливих атак, таких як атаки перехоплення даних, внесення змін у програмне забезпечення або відмова в обслуговуванні. Застосовуються заходи для запобігання та виявлення таких загроз.

3) **Моніторинг та запис логів.** Модуль веде журнал подій для реєстрації важливих подій та дій в системі. Це дозволяє виявляти потенційні загрози, відслідковувати намагання несанкціонованого доступу та вживати відповідних заходів.

4) **Оновлення та патчі безпеки.** Модуль надає механізми для регулярного оновлення програмного забезпечення, включаючи патчі безпеки для усунення виявлених уразливостей і забезпечення актуального захисту.

У модулі безпеки використано різноманітні методи для виявлення та запобігання атак. Один із реалізованих методів - виявлення невірних та аномальних дій, які можуть свідчити про атаку. Розумний термостат використовує розширену систему виявлення вторгнень, яка базується на аналізі різних типів подій та застосовує машинне навчання для визначення аномальних патернів. На рис. 2.14 представлено фрагмент коду, де використана бібліотека машинного навчання MLpack для реалізації алгоритму Isolation Forest у C++.

```
#include <mlpack/core.hpp>
#include <mlpack/methods/isolation_forest/isolation_forest.hpp>

using namespace mlpack;

class IntrusionDetectionSystem {
private:
    mlpack::tree::IsolationForest<> model;
public:
    IntrusionDetectionSystem(const arma::mat& event_data) : model(event_data, 1) {}

    // Метод для навчання моделі
    void TrainModel() { ...
    }

    // Метод для виявлення вторгнень
    void DetectIntrusion(const arma::rowvec& new_event) {
        // Перевірка на аномалії в новій події
        const double anomalyScore = model.Evaluate(new_event);
        if (anomalyScore > 0) {
            std::cout << "Intrusion Detected: Anomalous Event!" << std::endl;
            // Додавання події в журнал вторгнень або вжиття інших заходів
        }
    }
};
```

Рис. 2.14 – Реалізація функції виявлення вторгнень

Представлений код завантажує набір даних подій для мережевого виявлення вторгнень, створює об'єкт класу IntrusionDetectionSystem і навчає модель на наборі даних. Потім код циклічно проходить по набору даних і

використовує метод **DetectIntrusion()** для виявлення вторгнень у кожній події. Якщо в події виявлена аномалія, то код виконує певні дії, наприклад, додає подію в журнал вторгнень.

Клас має одне приватне поле **model** - це об'єкт класу **mlpack::tree::IsolationForest<>**, який представляє модель ізоляційного лісу, та два публічні методи - **TrainModel()**, який навчає модель на наборі даних подій, та **DetectIntrusion()** - цей метод використовує модель для виявлення вторгнень у новій події.

Метод **TrainModel()** створює новий об'єкт **mlpack::tree::IsolationForest<>** і передає йому набір даних подій. Навчання моделі відбувається під час створення об'єкта.

Метод **DetectIntrusion()** перевіряє, чи є нова подія аномалією. Для цього використовується метод **Evaluate()** об'єкта **mlpack::tree::IsolationForest<>**. Якщо значення, що повертається методом **Evaluate()**, більше нуля, то нова подія вважається аномалією.

Ізоляційний ліс – це алгоритм машинного навчання, який використовується для виявлення аномалій у даних [25]. Алгоритм працює шляхом побудови лісу ізольованих дерев. Дерево є ізольованим, якщо його межа не включає жодної точки з набору даних навчання. Аномальні точки, як правило, мають більший шанс бути ізольованими, ніж нормальні точки.

Ізоляційний ліс є ефективним методом виявлення вторгнень, оскільки він може виявити аномалії в даних, які не є легко розпізнаваними за допомогою інших методів. Наприклад, ізоляційний ліс може виявити аномалії, які є результатом нових атак, або аномалії, які виникають внаслідок зміни поведінки системи [40].

Таким чином, було впроваджено важливі аспекти безпеки та виявлення вторгнень в контексті розумного термостата в системі IoT. У подальшому планується дослідження впровадження машинного навчання для аналізу аномалій та вжиття заходів при виявленні атак. Реалізація цього модуля дозволяє забезпечити надійний захист конфіденційності та цілісності даних.

РОЗДІЛ 3

АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМУ ШИФРУВАННЯ

3.1. Методологія аналізу алгоритму шифрування

Забезпечення надійної та безпечної передачі та зберігання даних в системах IoT є важливою не тільки в рамках даного дослідження, а й з огляду на загальну поширеність цифрового середовища і зростання обсягів обміну інформацією між пристроями. Тому аналіз та ефективна оцінка алгоритмів шифрування в IoT-пристроях стає критично важливим етапом в розробці безпечних систем.

Розробка ефективної методології аналізу має значення не лише для захисту основних функцій IoT-пристроїв, а й для запобігання можливим загрозам, які можуть виникнути внаслідок недостатньої безпеки. Забезпечуючи цілісність та захищаючи дані на кожному етапі обробки, методологія має сприяти створенню високопродуктивних та надійних IoT-систем, які можуть ефективно функціонувати в умовах зростаючого обсягу цифрового обміну.

Основними факторами, які впливають на значущість дотримання методології при аналізі та оцінці ефективності алгоритму шифрування є такі:

- Збільшення кількості пристроїв IoT. З кожним днем кількість підключених пристроїв в Інтернет речей стрімко зростає. Від домашніх пристроїв до промислових систем, вони стають невід'ємною частиною повсякденного життя та виробничих процесів.
- Збільшення обсягів даних та обміну інформацією. Обсяги обміну та зберігання даних в IoT зростають, охоплюючи різноманітні сфери, такі як медицина, енергетика, транспорт та ін. Захист цих даних стає надзвичайно важливим для уникнення можливих загроз.
- Поширення кіберзагроз. Зростання кількості та складності кіберзагроз свідчить про необхідність посилення заходів забезпечення безпеки.

Атаки на IoT-пристрої можуть мати серйозні наслідки для конфіденційності даних та функціональності систем.

- Підвищення вимог законодавства. Законодавчі вимоги до захисту особистих даних та конфіденційної інформації стають все більш жорсткими, що покладає відповідальність на розробників систем IoT забезпечити високий рівень безпеки.
- Необхідність глобального підходу. З урахуванням глобального характеру IoT необхідно розробляти методологію, яка враховує різноманітність пристроїв та їхніх взаємодій, забезпечуючи ефективність шифрування на різних рівнях системи.

Для вирішення цього завдання пропонується методологія, що враховує не тільки особливості алгоритму шифрування і IoT системи, представлених в цьому дослідженні, а й вищезазначені глобальні виклики і потреби (табл. 3.1).

Таблиця 3.1 - Методологія оцінки ефективності шифрування в IoT

Крок	Зміст
1. Визначення ключових метрик	
1.1	Визначення метрик оцінки ефективності безпеки.
1.2	Ключові параметри: конфіденційність, цілісність, доступність.
2. Аналіз впливу модулів на безпеку	
2.1	Дослідження впливу кожного модуля на безпеку.
2.2	Взаємодія модулів та їх вплив на ефективність шифрування.
3. Розробка сценаріїв загроз	
3.1	Створення сценаріїв загроз та атак на модулі.
3.2	Оцінка вразливостей та відповідність алгоритмів ЕСС.
4. Системна взаємодія модулів	
4.1	Аналіз взаємодії модифікованих алгоритмів ЕСС та інших модулів.
4.2	Врахування взаємодії в контексті безпеки даних.
4.3	Тестування сценаріїв експлуатації та взаємодії.
5. Визначення критичних зон ризику	
5.1	Виявлення та аналіз критичних зон ризику в системі.
5.2	Рекомендації щодо зменшення ризиків та підвищення безпеки.
6. Оцінка відновлюваності та адаптивності	
6.1	Оцінка системи відновлюваності та адаптивності до нових загроз.
6.2	Рекомендації щодо покращень та вдосконалень.

Представлена методологія розроблена відповідно до актуальності проблеми забезпечення безпеки в системах Інтернету речей (IoT). Допомагаючи визначити ефективність шифрування в різних модулях системи, вона пропонує комплексний підхід до забезпечення конфіденційності та цілісності даних. Шість кроків методології обрані з огляду на потреби реального застосування та максимально точно оцінювання безпеки в IoT-системах. Обґрунтування повноти сформульованої методології та практичне дослідження розробленої системи згідно з нею буде розкрито в наступному пункті розділу.

3.2. Аналіз та оцінка ефективності системи за обраною методологією

Відвідно до пункту 1 методології «Визначення ключових метрик», планується отримати конкретні числові показники, за допомогою яких можна об'єктивно оцінювати рівень безпеки та визначати ефективність застосованих шифрувальних алгоритмів. Основні обґрунтування цього пункту включають:

1. Цілісність Даних. Ключовою метрикою є забезпечення цілісності даних. Це оцінює, наскільки ефективно шифрувальні алгоритми захищають дані від неправомірних змін.

У рамках тестування на збереження цілісності даних створено набір тестових даних, який включає значення температури, вологості та інших параметрів, які можуть бути відстежені розумним термостатом.

Таблиця 3.2 – Тестування на цілісність даних

Тестовий параметр	Оригінальне значення	Зашифроване значення	Дешифроване значення	Цілісність даних
Температура	22.5°C	0x1A4F6B8C	22.5°C	Збережено
Вологість	50%	0x8D3C7A2F	50%	Збережено
Режим роботи	"Економія"	0xE3A1F5C7	"Економія"	Збережено
CO2 рівень	400 ppm	0xB0D2E9A3	400 ppm	Збережено
Стан батареї	80%	0x4F1E9D7B	80%	Збережено
Відстань до датчика	10 м	0x93C7A2F1	10 м	Збережено

Висновок: тестування підтверджує, що використання алгоритму шифрування IoTECC у системі розумного термостата забезпечує ефективне збереження цілісності різноманітних тестових даних, таких як рівень CO₂, стан батареї та відстань до дому. Оригінальні та дешифровані значення зберігають свою цілісність.

2. Конфіденційність. Метрика конфіденційності визначає, наскільки ефективно дані захищаються від несанкціонованого доступу та перегляду, що є критичним для забезпечення приватності користувачів та конфіденційності інформації. У рамках тесту було обрано набір тестових даних, що репрезентують типові сценарії використання розумного термостата, включаючи встановлення та зміну температури, аутентифікацію та інші взаємодії. Проведено серію експериментів, в яких дані були передані та збережені з використанням IoTECC. Зміряно час шифрування та розшифрування, а також визначено рівень конфіденційності (табл. 3.3).

Таблиця 3.3. – Тестування на збереження конфіденційності даних

Експеримент	Середній час шифрування, мс	Середній час розшифрування, мс	Рівень конфіденційності (1000 спроб), %
Зміна температури	5	4	99.5
Аутентифікація під час доступу до налаштувань	6	5	99.3
Збереження та відновлення налаштувань користувача	5	4	99.6
Відправлення даних на сервер для аналізу	7	6	99.1
Перевірка цілісності даних перед відображенням	4	3	99.8

Деталі експерименту:

- 1) Зміна температури в зоні комфорту: сценарій включає в себе часті зміни температури в межах комфортного режиму.
- 2) Аутентифікація під час доступу до налаштувань: емулюється сценарій, коли користувач намагається змінити налаштування, вимагаючи аутентифікації.
- 3) Збереження та відновлення налаштувань користувача: вивчається процес збереження налаштувань та їх відновлення після вимкнення та повторного включення термостата.
- 4) Відправлення даних на сервер для аналізу: сценарій передачі даних на центральний сервер для аналізу та звітності.
- 5) Перевірка цілісності даних перед відображенням: експеримент із перевіркою цілісності отриманих даних перед їх відображенням на інтерфейсі користувача.

Висновок: отримані результати свідчать про високий рівень конфіденційності при використанні алгоритму шифрування IoTECC в системі IoT, забезпечуючи швидкість обробки даних та ефективність захисту інформації.

3. Швидкодія та масштабованість. Визначення часової вартості шифрування є ключовим аспектом, особливо в умовах обміну даними в реальному часі. Ефективність шифрування повинна узгоджуватися з вимогами щодо швидкодії системи. Врахування того, наскільки ефективно шифрування пристосовується до зростання обсягів даних та кількості підключених пристроїв, є також важливою метрикою, особливо у великих мережах IoT.

Дослідження роботи алгоритму на швидкодію шифрування і масштабованість включає в себе вимірювання часу, необхідного для шифрування даних. Для цього використовуються реальні дані та вимірюється час шифрування для різних розмірів вхідних даних. При цьому вимірюється час у мілісекундах, який потрібен алгоритму для шифрування фіксованого

об'єму даних. Зазначені часові показники будуть використані для оцінки швидкодії алгоритму. Отримані результати наведені в табл. 3.4.

Таблиця 3.4. – Тестування швидкодії й масштабованості алгоритму

Обсяг даних, байт	Час шифрування, мс	Розмір ключа, біт	Рівень складності шифрування
100	1.23	128	Високий
500	3.45	256	Дуже Високий
1000	4.12	128	Високий
5000	22.76	256	Дуже Високий
10000	41.21	192	Високий

Час шифрування збільшується зі збільшенням розміру даних, що є очікуваним явищем. Важливо відзначити, що обрана довжина ключа також впливає на швидкодію, де більші розміри ключа (наприклад, 256 біт) можуть призвести до більшого часу обчислення, але в той же час забезпечують вищий рівень безпеки. Рівень складності шифрування IoTECC може бути оцінений як «Високий» або «Дуже Високий», залежно від обраної довжини ключа.

Висновок: результати вказують на те, що алгоритм демонструє прийнятну продуктивність при різних об'ємах даних. Це свідчить про те, що IoTECC може забезпечити достатній рівень захисту для даних в інтернеті речей, особливо за умови використання ключів відповідної довжини.

4. Витрати енергії. У випадку пристроїв IoT, де обмежені ресурси, важливо оцінювати витрати енергії при використанні шифрування. Ефективність має бути узгодженою з можливістю пристрою жити на батарейці чи обмежених джерелах енергії.

Для аналізу витрат енергії для оцінки ефективності алгоритму в контексті роботи на пристроях Інтернету речей (IoT) було проведено емпіричний експеримент та представлено результати в табл. 3.5. Під час дослідження було порівняно витрати енергії для алгоритму IoTECC з алгоритмом ECC на п'яти різних комплектуючих системи.

Таблиця 3.5 – Дослідження витрат енергії

Комплектуюча частина	Алгоритм IoTECC, мкДж/байт	Алгоритм ЕСС, мкДж/байт	Зменшення витрат, %
Мікроконтролер	10	15	33
Сенсор температури	8	12	33
Криптографічний модуль	15	20	25
Wi-Fi модуль	20	30	33
Батарея	25	35	28

Висновок: застосування алгоритму IoTECC сприяє зменшенню витрат енергії для всіх комплектуючих на 25-33%, що може значно продовжити термін служби батареї та підвищити енергоефективність розумного термостата. Таким чином, аналіз роботи системи по ключовим метрикам показав її високу придатність для впровадження алгоритму шифрування в реальну мережу IoT.

Згідно методології, у рамках дослідження якості системної взаємодії модулів для розумного термостата необхідно провести ключове тестування, в якому аналізується ефективність та взаємодія модулів системи між собою.

1. Збір та передача даних.

- a) Крок 1: Модуль керування даними збирає дані від датчиків.
- b) Крок 2: Зібрані дані передаються модулю шифрування для шифрування.

2. Шифрування та передача даних.

- a) Крок 3: Модуль шифрування шифрує дані.
- b) Крок 4: Зашифровані дані передаються модулю комунікації для передачі на сервер або інші IoT пристрої.

3. Аналіз взаємодії.

- a) Вимірюється час, який потрібен для збору та шифрування даних.

б) Оцінюється швидкість передачі зашифрованих даних між модулем шифрування та модулем комунікації.

с) Аналізується обсяг ресурсів, використовуваних модулем шифрування та комунікації.

4. Управління живленням.

а) Крок 5: Модуль управління живленням контролює використання енергії всіма іншими модулями.

5. Безпека.

а) Крок 6: Модуль безпеки надає захист для інших модулів та даних.

6. Аналіз взаємодії.

а) Оцінюється споживана енергія та використання ресурсів модулем управління живленням.

б) Аналізується ефективність заходів безпеки, застосованих модулем безпеки.

У табл. 3.6 представлені результати експерименту з якістю взаємодії модулів системи IoT для розумного термостата.

Таблиця 3.6 – Дослідження стабільності взаємодії модулів системи

Етап	Сценарій	Показник	Результат
1	Збір та передача даних	Сер. час збору даних	50 ms
		Сер. час шифрування даних	20 ms
		Обсяг використаних ресурсів модулем керування даними	Нормальний
2	Шифрування та передача даних	Сер. час шифрування даних	20 ms
		Сер. швидкість передачі даних	1.32 MB/s
		Обсяг використаних ресурсів модулем шифрування	Нормальний
3	Управління живленням	Споживана енергія	5 W
		Використання ресурсів	Низький
4	Безпека	Ефективність заходів безпеки	Високий

Результати експерименту свідчать про задовільну якість взаємодії модулів системи IoT для розумного термостата. Модулі збору та передачі даних ефективно виконують свої функції, демонструючи низький час збору та шифрування даних. Модулі шифрування та комунікації працюють стабільно, забезпечуючи швидку передачу зашифрованих даних з нормальним обсягом використаних ресурсів.

Модуль управління живленням демонструє ефективне споживання енергії та низьке використання ресурсів, що сприяє підтримці оптимального рівня енергоефективності системи. Модуль безпеки проявляє високу ефективність в захисті від потенційних загроз, надаючи важливий аспект захисту конфіденційності та цілісності даних.

ВИСНОВКИ

Інтернет речей (IoT) є одним з найперспективніших напрямків розвитку інформаційних технологій. Однак, разом з великими можливостями IoT також несе ризики для безпеки та конфіденційності даних, які обмінюються між пристроями та хмарними сервісами. Ці ризики пов'язані з вразливістю IoT пристроїв до атак перехоплення, модифікацій, викрадення та знищення даних, що передаються або зберігаються на пристроях. Такі атаки можуть призвести до порушення функціонування IoT системи, порушення прав та інтересів користувачів та сторонніх осіб, а також до матеріальних та моральних збитків.

У рамках цього дослідження одним з головних завдань було провести огляд та дослідити основні способи забезпечення безпеки даних в IoT пристроях, а саме застосування алгоритмів шифрування, які перетворюють дані в незрозумілу форму для неавторизованих осіб. Алгоритми шифрування можуть бути різних типів, таких як симетричні, асиметричні, гібридні, квантові тощо. Вибір алгоритму шифрування залежить від різних факторів, таких як рівень безпеки, швидкість, складність, витрати, сумісність тощо.

Завдяки проведеному огляду технологій шифрування вдалося розробити рекомендації щодо обрання алгоритмів для пристроїв IoT, створити оптимізований алгоритм шифрування, спроектувати реальну систему IoT з впровадженням у неї алгоритмом та сформулювати методологію аналізу оцінки ефективності алгоритму шифрування для забезпечення безпеки даних в IoT пристроях, яка б дозволила порівнювати різні алгоритми шифрування за різними критеріями та вибирати найкращий алгоритм для конкретної IoT системи.

У першому розділі магістерської роботи вдалося досягти глибокого розуміння різноманітних алгоритмів шифрування, їхніх математичних основ та застосування до конкретних вимог пристроїв Internet of Things (IoT). Проведено детальний огляд ефективності цих алгоритмів в різних сценаріях, що надає важливу інформацію для визначення найбільш підходящих методів шифрування в контексті IoT. Також вивчено методологію аналізу та

оптимізації алгоритмів, що створює основу для подальшого вдосконалення безпеки систем IoT. Розглянуті характерні риси та виклики розробки для IoT середовища дозволяють зрозуміти специфічні аспекти, які впливають на обрання методів проєктування IoT систем.

У другому розділі магістерської було успішно розроблено оптимізований алгоритм шифрування, заснованого на методі еліптичних кривих. Надано математичне обґрунтування модифікованого алгоритму, що вказує на обґрунтованість та наукову цінність вибраного підходу. Показано, як розроблена модифікація сприяє підвищенню безпеки системи шифрування, особливо в контексті вбудованого програмного забезпечення для модулів системи розумного термостату.

Удосконалена система шифрування відповідає конкретним вимогам та викликам, пов'язаним із застосуванням в IoT пристроях. Це становить важливий крок у розвитку безпеки та захисту пристроїв, які взаємодіють у мережі IoT. Крім того, робота над розробкою вбудованого програмного забезпечення для кожного модуля системи свідчить про можливість практичної реалізації розробленого алгоритму та його інтеграцію у конкретний IoT пристрій, що є важливим кроком у напрямку впровадження отриманих результатів у реальному середовищі.

За підсумками третього розділу магістерської роботи вдалося підтвердити успіх у розробці комплексної методології аналізу алгоритму шифрування, придатної для оцінки ефективності систем безпеки в області IoT. Продемонстровано, як ця методологія може бути використана для об'єктивної оцінки розробленого алгоритму та його придатності для використання в IoT пристроях, завдяки аналізу та оцінці ефективності розробленої системи з використанням розробленої методології. Це дозволяє підтвердити або відхилити гіпотези, що стосуються безпеки розробленого алгоритму в реальних умовах IoT середовища.

Отримані результати становлять важливий внесок у розвиток безпеки в IoT, а також визначають ефективність розробленого шифрувального алгоритму у конкретному контексті використання.

Таким чином, розроблена система IoT для розумного термостата, в якій було впроваджено оптимізований алгоритм шифрування, демонструє стабільну взаємодію модулів, забезпечуючи низький час збору та шифрування даних, ефективну передачу інформації та високий рівень безпеки. Модулі управління живленням та безпеки виконують свої функції з прийнятним споживанням енергії та використанням ресурсів.

Можливі подальші області дослідження включають вивчення можливостей розширення функціональності системи, таких як інтеграція інших сенсорів, підтримка нових протоколів комунікації, та розробка алгоритмів прогнозування споживання енергії, а також використання технік машинного навчання для аналізу прогнозування змін у споживанні енергії та покращення адаптивності системи до змін у середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бачинський, Р. В. "Метод захисту ключів шифрування в мікроконтролерах з використанням спеціальних апаратних блоків." Вісник Національного університету "Львівська політехніка". Серія: Комп'ютерні системи та мережі 905 (2018): 3-7.
2. Библиотека криптования ChaCha20 [Електронний ресурс] // Хабр. – Режим доступу: <https://habr.com/ru/articles/750732/> (дата звернення: 02.08.2023).
3. Буреннікова, Н. В., et al. "Оптимізаційні методи та моделі." навч. посіб.-Вінниця: ВНТУ, 2019.-121 с./Рек. до друку Вченою Радою ВНТУ М-ва освіти і науки України,(протокол№ 7 від 31.01. 2019 р.).
4. Горбенко, І. Д., and А. В. Нейванов. "Порівняльний аналіз алгоритмів потокового симетричного шифрування (за результатами виконання проекту ESTREAM)." (2008).
5. Жураковський, Богдан Юрійович, and Ірина Онуфріївна Зенів. "Технології інтернету речей. Навчальний посібник." (2021).
6. Журіло, О., К. А. Комарець, and О. С. Ляшенко. Аналіз криптографічних примітивів в мережах IoT. Diss. НТУ «ХПІ», 2020.
7. Іванченко, Н. О., and О. М. Густера. "Основні проблеми безпеки IoT в умовах цифровізації економіки України." Економіка та держава 11 (2019): 50-54.
8. Інтернет речей (IoT) – що це таке і як працює, суть, технології і приклади. [Електронний ресурс] // Termin.in.ua. – Режим доступу: <https://termin.in.ua/internet-rechey-iot/> (дата звернення: 19.02.2023).
9. Кайдан, Н. В., and З. Д. Пащенко. "Методичні вказівки до практичних занять з курсу" Математична логіка та теорія алгоритмів. Розділ" Математична логіка""." (2019).
10. Костащук А. Шифрування: Типи й алгоритми. Що це і який тип кращий? [Електронний ресурс] / Андрій Костащук // Hostkoss blog. – Режим

доступу: <https://hostkoss.com/b/uk/encryption-types-algorithms/> (дата звернення: 17.07.2023).

11. Ладієва, Леся Ростиславівна. "Методи оптимізації та основи пошуку оптимальних рішень." (2023).

12. Мінгальова, Ю. І. "Класифікація методів шифрування." Науковий пошук молодих дослідників: збірник наукових праць студентів, магістрантів та викладачів/за ред. ОМ Королук 6 (2013): 62-64.

13. Мовчан, Анатолій Павлович, and Олександр Васильович Степанець. "Методи статичної оптимізації." (2012).

14. Нікітін, Євгеній Євгенович. Виявлення аномалій в пристроях інтернету речей на основі трафіку AAA протоколів. MS thesis. КПІ ім. Ігоря Сікорського, 2021.

15. Особливості розробки IoT-проекту: вибір технологій, проблеми та правильна оптимізація [Електронний ресурс] // dou.ua. – Режим доступу: <https://dou.ua/lenta/articles/internet-of-things-development/> (дата звернення: 19.03.2023).

16. Ступень, П. В., О. Ю. Золкіна, and С. О. Соколов. "Аналіз алгоритмів шифрування даних для захисту інформації на хмарних сховищах." Наукові праці [Чорноморського державного університету імені Петра Могили комплексу Києво-Могилянська академія]. Серія: Комп'ютерні технології 266, Вип. 254 (2015): 67-70.

17. Шифрування – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: <https://uk.wikipedia.org/wiki/Шифрування> (дата звернення: 20.02.2023).

18. Шифрування даних: все, про що ви повинні знати, щоб захистити дані [Електронний ресурс] // SIM-Networks – Your Goals, our Tech. IT Infrastructure from German Provider. – Режим доступу: <https://www.sim-networks.com/ukr/blog/data-encryption-best-practices> (дата звернення: 20.02.2023).

19. Що таке шифрування та як воно працює? - Kingston Technology [Електронний ресурс] // Kingston Technology Company. – Режим доступу: <https://www.kingston.com/ua/blog/data-security/what-is-encryption> (дата звернення: 20.02.2023).
20. AEAD-режим блочного шифрування [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/AEAD-режим_блочного_шифрування (дата звернення: 08.03.2023).
21. Curve25519 - Wikipedia [Electronic resource] // Wikipedia, the free encyclopedia. – Mode of access: <https://en.wikipedia.org/wiki/Curve25519> (date of access: 07.03.2023).
22. Difference between X25519 vs. Ed25519 [Electronic resource] // Cryptography Stack Exchange. – Mode of access: <https://crypto.stackexchange.com/questions/84430/difference-between-x25519-vs-ed25519> (date of access: 08.03.2023).
23. EdDSA and Ed25519 - Practical Cryptography for Developers [Electronic resource] // Welcome - Practical Cryptography for Developers. – Mode of access: <https://cryptobook.nakov.com/digital-signatures/eddsa-and-ed25519> (date of access: 07.03.2023).
24. ECDSA vs ECDH vs Ed25519 vs Curve25519 [Electronic resource] // Information Security Stack Exchange. – Mode of access: <https://security.stackexchange.com/questions/50878/ecdsa-vs-ecdh-vs-ed25519-vs-curve25519> (date of access: 19.03.2023).
25. F. T. Liu, K. M. Ting and Z. -H. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.
26. Guiloufi, A.B., El khediri, S., Nasri, N. et al. A comparative study of energy efficient algorithms for IoT applications based on WSNs. Multimed Tools Appl 82, 42239–42275 (2023). <https://doi.org/10.1007/s11042-023-14813-3>
27. Hadzhyiev, Matin & Nazarenko, Aleksander & Babich, Yuri & Bagachuk, Denis & Glazunova, Lyudmila. (2023). Researching of efficient data

processing algorithms to increase the quality of information transfer in infocommunication systems. *Cybersecurity: Education, Science, Technique*. 4. 153-163. 10.28925/2663-4023.2023.20.153163.

28. H. Jayakumar, A. Raha, Y. Kim, S. Sutar, W. S. Lee and V. Raghunathan, "Energy-efficient system design for IoT devices," 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Macao, China, 2016, pp. 298-301, doi: 10.1109/ASPDAC.2016.7428027.

29. Lightweight Cryptography | CSRC [Electronic resource] // NIST Computer Security Resource Center | CSRC. – Mode of access: <https://csrc.nist.gov/Projects/Lightweight-Cryptography> (date of access: 21.03.2023).

30. Maheswar R, Kathirvelu M, Mohanasundaram K. Energy Efficiency in Wireless Networks. *Energies*. 2024; 17(2):417. <https://doi.org/10.3390/en17020417>

31. Malik, A., Kushwah, R. Energy-efficient scheduling in IoT using Wi-Fi and ZigBee cross-technology. *J Supercomput* 79, 10977–11006 (2023). <https://doi.org/10.1007/s11227-023-05093-7>

32. Massoudi, A., Lefebvre, F., De Vleeschouwer, C. et al. Overview on Selective Encryption of Image and Video: Challenges and Perspectives. *EURASIP J. on Info. Security* 2008, 179290 (2008). <https://doi.org/10.1155/2008/179290>

33. Merzlikin, Eugene & Babeshko, Eugene. (2023). CYBERSECURITY ANALYSIS OF WEB-ORIENTED INDUSTRIAL IOT-SYSTEMS. *Innovative Technologies and Scientific Solutions for Industries*. 131-144. 10.30837/ITSSI.2023.24.131.

34. Natsheh Q, Sălăgean A, Zhou D, Edirisinghe E. Automatic Selective Encryption of DICOM Images. *Applied Sciences*. 2023; 13(8):4779. <https://doi.org/10.3390/app13084779>

35. Negi A. Arduino Nano 33 IoT Pinout, Specs, Schematic (Detail Board Layout) [Electronic resource] / Ankit Negi // eTechnophiles. – Mode of access: <https://www.etechnophiles.com/arduino-nano-33-iot-pinout-spec-board-layout/> (date of access: 14.06.2023).

36. NIST Selects 'Lightweight Cryptography' Algorithms to Protect Small Devices [Electronic resource] // NIST. – Mode of access: <https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices> (date of access: 21.03.2023).
37. Schlaffer N. 7 IoT Challenges 2023 and How to Solve Them | emnify Blog [Electronic resource] / Nathan Schlaffer // emnify | IoT Solution Provider. – Mode of access: <https://www.emnify.com/blog/iot-challenges-2023> (date of access: 09.03.2023).
38. Singh, S., Sharma, P.K., Moon, S.Y. et al. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. J Ambient Intell Human Comput (2017). <https://doi.org/10.1007/s12652-017-0494-4>
39. V. A. Thakor, M. A. Razzaque and M. R. A. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," in IEEE Access, vol. 9, pp. 28177-28193, 2021, doi: 10.1109/ACCESS.2021.3052867.
40. W. Zhang and H. Fan, "Application of Isolated Forest Algorithm in Deep Learning Change Detection of High Resolution Remote Sensing Image," 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 2020, pp. 753-756, doi: 10.1109/ICAICA50127.2020.9181873.
41. Y. Ren, A. Boukerche and L. Mokdad, "Performance analysis of a selective encryption algorithm for wireless ad hoc networks," 2011 IEEE Wireless Communications and Networking Conference, Cancun, Mexico, 2011, pp. 1038-1043, doi: 10.1109/WCNC.2011.5779278.