Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав.кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Дослідження застосування алгоритмів малоресурсної криптографії у децентралізованих середовищах»

оцінка « »

Голова ЕК

Доценко С.І. _____

Керівник Доцент ЗВО, к.т.н

Полуяненко М. О.

Рецензент Професор ЗВО, д.т.н.

Кузнецов О. О.

Виконавець: студент групи КБ-61

Деменко Є. Є.

Харків – 2022

РЕФЕРАТ

Дипломна робота складається з 78 сторінок, 15 рисунків, 3 таблиць та 91 джерел посилань.

Об’єктом дослідження є визначення актуальності сучасних алгоритмів малоресурсної криптографії, які використовуються в технологіях Інтернету речей.

Предметом дослідження є програмна реалізація алгоритму малоресурсної криптографії та можлива реалізація на платформі Інтернету речей.

Метою роботи є дослідити концепцію Інтернету речей та визначити та порівняти алгоритми малоресурсної криптографії для подальшої реалізації на платформах Інтернету речей.

Основні методи досліджень є узагальнення отриманих результатів та комп’ютерне моделювання.

В першому та другому розділі проаналізовано концепцію Інтернету речей, виклики безпеки та шляхи децентралізації. Наданий загальний огляд малоресурсної криптографії, їх властивості та структурні елементи. У третьому проведено дослідження та порівняння алгоритмів малоресурсної криптографії. Зроблено обґрунтування вибору алгоритму, розкриття його роботи та безпосередня його реалізація.

Результати досліджень дозволяють підготувати підґрунтя для використання реалізації на платформах Інтернету речей та в особливості на апаратній частині.

Демонструється, що малоресурсна криптографія на сьогоднішні день є актуальною технологію для задоволення великого спектру потреб, котрі відносяться до збереження цілісності даних на платформах Інтернету речей.

Ключові слова: МАЛОРЕСУРСНА КРИПТОГРАФІЯ, ЛЕГКА КРИПТОГРАФІЯ, ІНТЕРНЕТ РЕЧЕЙ, БЛОКОВІ ШИФРИ, МАЛОРЕСУРСНА КРИПТОГРАФІЯ, IOT, PRESENT, CLEFIA, AES, ДЕЦЕНТРАЛІЗОВАНІ СИСТЕМИ.

ABSTRACT

The work consists of 78 pages, 15 figures, 3 tables and 91 references.

The aim of the research is to determine the relevance of modern low-resource encryption algorithms used in the Internet of Things.

The topic of the research is the software implementation of a low-resource encryption algorithm and possible implementation on the Internet of Things platform.

The aim of the work is to explore the concept of the Internet of Things and to identify and compare low-resource encryption algorithms for further implementation on the Internet of Things platform.

The main research methods are the generalization of the obtained results and computer modeling.

The first and second parts analyze the concept of the Internet of Things, security challenges and forms of decentralization. An overview of low-function cryptography, its properties and structural elements is presented. The third part studies and compares low-resource encryption algorithms. The justification of the choice of the algorithm, the publication of its work and its direct implementation are presented.

The results of the research allow us to prepare the ground for the use of the implementation of Internet of Things platforms, especially in hardware.

It shows that low-resource cryptography is now a relevant technology to meet a wide range of data integrity needs in IoT platforms.

Keywords: LOW SOURCE ENCRYPTION, LIGHTWEIGHT ENCRYPTION, INTERNET OF THINGS, BLOCK CIFRES, LOW SOURCE ENCRYPTION, IOT, PRESENT, CLEFIA, AES, DECENTRALIZED SYSTEMS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ СКОРОЧЕНЬ І ТЕРМІНІВ

5G	–	технологічний стандарт п'ятого покоління для широкосмугових стільникових мереж.
API	–	Application Programming Interface
AES	–	Advanced Encryption Standard
CBC	–	Cipher Block Chaining
DDoS	–	(distributed) denial-of-service attack
DTLS	–	Datagram Transport Layer Security
FSM	–	Finite-state machine
FPGA	–	field-programmable gate array
FN	–	Feistel network
EPC	–	Electronic Power Control
GE	–	Gate Equivalents
GFN	–	General Feistel network
GSN	–	Global Sensor Networks
HIP	–	Host Identity Protocol
HTTP	–	HyperText Transfer Protocol
IaaS	–	Infrastructure as a Service
IoT	–	Internet of Things,
IPSec	–	IP Security
LFSR	–	Linear-feedback shift register
LWC	–	Lightweight cryptography
MAC	–	Media Access Control
MANET	–	Mobile Ad hoc Network
MQTT	–	Message queuing telemetry transport
NIST	–	National Institute of Standards and Technology

OWASP	–	Open Web Application Security Project.
RFID	–	Radio Frequency IDentification;
RSSI	–	Received Signal Strength Indication
SaaS	–	Software as a Service
SPN	–	Substitution–permutation network
SEAPOL	–	Slim Extensible Authentication Protocol Over Local Area Networks
TCP	–	Transmission Control Protocol
TLS	–	Transport Layer Security
UDP	–	User Datagram Protoco
VANET	–	Vehicular ad hoc networks
WSN	–	Wireless sensor network
ІБ	–	Інформаційна безпека
МН	–	Машинне навчання
ІШІ	–	Штучний інтелект

ВСТУП

Останні роки Інтернет речей став однієї із «найтрендовіших» технологій цього століття. Сьогодні широкосмуговий Інтернет є загальнодоступним, а вартість підключення постійно знижується. Як наслідок, до всесвітньої мережі Інтернет підключається все більше гаджетів та різних датчиків [1-2]. Сьогодні людство досягло високого рівня технологічного розвитку, що дозволяє налаштовувати взаємодію між пристроями та створювати безшовний зв’язок між процесами, речами та людьми.

З появою 5G технології, IoT стали центром розвитку майже для всіх сучасних галузей. Ці тенденції створюють сприятливе підґрунтя для розвитку Інтернету речей (IoT). Однак, наразі навколо технології Інтернету речей існує багато складнощів [2]. Перш за все це зв’язано зі складністю, як елементної бази, так і спеціальних алгоритмів обробки даних, що реалізуються в IoT пристроях. Пристрої в цій архітектурі значно менші та мають низьке енергоспоживання. Звичайні алгоритми шифрування, як правило, дорогі в обчислювальному плані через їх складність і вимагають багато раундів, однак це може поставити під загрозу бажану цілісність. Процес обміну такою великою кількістю даних починається з самих пристроїв, які повинні безпечно взаємодіяти із платформою.

Сутність Інтернету речей - зв'язок та обмін інформацією [2]. Однак, всі ці дані не генеруються лише для того, щоб їх десь зберігати і «забути про їх існування». Основна ціль їх використання, це автоматизація. IoT практично «стирає» розрив між цифровим та фізичним світом, однак має і зворотний бік процесу - компрометація IoT пристроїв, може мати небезпечні наслідки в «реальному» світі.

Як правило, система IoT включає в себе проміжний, апаратний та представницький рівні[2]. Апаратний рівень забезпечує роботу виконавчих механізмів і датчиків, проміжний рівень забезпечує обчислення зі збереженням

даних, а рівень представлення, включає в себе засоби інтерпретації для отримання інформації з різних базових форм.

Безпека даних в Інтернеті речей формуються на фізичному, семантичному або обчислювальному рівні. Більш популярні новітні атаки направлені на фізичний рівень, а основні атаки досі виконуються на програмному рівні [3].

Система IoT включає в себе один або декілька датчиків, які застосовуються для збору даних та підтримки мережевих інтерфейсів [3]. Тип та номенклатура даних, що збирають ці датчики, залежать від конкретного пристрою і їх функціонального завдання. При цьому, всі накопичені дані, та дані телеметрії (технологічна інформація) інтенсивно циркулюють та взаємодіять в межах відповідної мережі пристроїв що обумовлює актуальність питань забезпечення безпеки обміну інформацією.

Однією з найпоширеніших особливих рис, яка є найбільш поширеною серед застарілих апаратних засобів, є серіалізація, наприклад, замість того, щоб виконувати одні й ті ж завдання паралельно є можливість використовувати одне із завдань і спробувати застосувати його до входів в різних тактових циклах послідовно, яка і називається серіалізацією. Ця особливість може зменшити площу реалізації за рахунок збільшення часу виконання. Для досягнення того ж результату, критичні за площею функції розбиваються на підфункції, які можуть застосовуватися повторно.

Gartner [4] прогнозує, що IoT досягне 26 мільярдів одиниць до 2020 року, порівняно з 0,9 мільярда у 2009 році, і вплине на інформацію, яка доступна для партнерів ланцюга поставок, і на те, як працює ланцюг поставок[4]. Від виробничої лінії і складування до роздрібної доставки і розміщення товарів на полицях магазинів - Інтернет речей трансформує бізнес-процеси, забезпечуючи більш точну видимість потоку матеріалів і продуктів в режимі реального часу. В перспективі фірми та підприємства будуть інвестувати в технологію, щоб перепроектувати робочі процеси на заводах, поліпшити відстеження матеріалів та оптимізувати витрати на дистрибуцію.

Передові технології моніторингу та управління, такі як «розумні мережі» та «розумний облік», дозволяють виявити закономірності роботи, визначити області потенційного поліпшення або спрогнозувати майбутні результати і оптимізувати операції, що призводить до зниження витрат і підвищення продуктивності. Ці дані використовуються для виявлення і вирішення бізнес-проблем, таких як зміни в поведінці клієнтів і ринкових умовах, для підвищення задоволеності клієнтів і надання їм послуг з доданою вартістю. Інструменти бізнес-аналітики можуть бути вбудовані в пристрої IoT, а саме - як носимі датчики моніторингу стану здоров'я. Прийняття рішень в реальному часі може відбуватися в джерелі даних.

Існуючі в зараз платформи IoT використовують переважно централізовану модель, згідно з якою вони виступають в якості «брокерів» або концентраторів для управління обміном даними між пристроями IoT [1,5]. Однак, багато досліджень свідчать, що IoT повинен використовувати насамперед децентралізовану модель для забезпечення безпечного обміну даними. При цьому ключовими проблемами реалізації традиційної криптографії в пристроях IoT вважаються наступні [1,5]:

- низький рівень наявної обчислювальної потужності (або відсутність батареї у випадку пасивних RFID-міток);
- невелика фізична площа для реалізації збірки;
- реакція в реальному часі.
- обмеженість ресурсів наявної пам'яті IoT пристроів ;
- низький заряд батареї (або навпаки її відсутність);

Різні зусилля по криптографічній стандартизації розглядають як програмний, так і апаратний аспект безпеки. Програмні та апаратні рішення безпеки мають різні метрики. Програмні метрики включають цикли, пам'ять або цикл на байт, в той ж час як апаратні метрики включають пропускну здатність, площу, співвідношення по всій площі. Важко отримати пряме порівняння між цими двома показниками.

Словосполучення «малоресурсне програмне забезпечення» було створено шляхом поєднання обмежень швидкості, використання пам'яті та споживання енергії.

Датчики, наприклад, класифікуються як пристрої Інтернету речей і вочевидь мають обмежені ресурси. Як наслідок, концепція легкої ваги є критично важливою для їх роботи. Як наслідок, програмне забезпечення, алгоритми, програми, протоколи та додатки повинні розроблятися на основі малоресурсних стандартів, щоб ефективно працювати при впровадженні в пристроях IoT, оскільки вони отримують більшу продуктивність, коли вони засновані на малоресурсній вазі.

Малоресурсна або ж легка криптографія є компромісом між такими категоріями, як вартість реалізації, швидкість, безпека, продуктивність та енергоспоживання на пристроях з обмеженими ресурсами. При цьому, мотивація для використання малоресурсної криптографії полягає у використанні меншого обсягу пам'яті, менших обчислювальних ресурсів та меншого енергоспоживання заради забезпечення безпеки [6].

Малоресурсна криптографія дає рішення для реалізації безпеки на апаратному рівні. Ці криптографічні методи спеціально створені для вбудованих систем. Тому реалізація малоресурсної криптографії спрямована на оптимізацію певних критеріїв, таких як енергоспоживання, час виконання, обсяг пам'яті та розмір мікросхеми.

Блокові шифри мають фіксовану довжину бітів і різні кроки перетворення, які визначаються симетричним ключем[6]. Блокові шифри дуже універсальні, що дуже корисно з точки зору Інтернету речей. Ще одна перевага полягає в тому, що процес перетворення має близькі за схожістю методи шифрування та дешифрування. Тому його можна реалізувати з меншими ресурсами.

Об’єкт дослідження – малоресурсна криптографія.

Предмет дослідження – реалізація та формування рекомендацій щодо застосування алгоритмів малоресурсної криптографії в системах Інтернету речей.

Метою дослідження - ознайомлення з областю досліджень застосування малоресурсних алгоритмів криптографії для систем Інтернету речей (IoT) та можливості прямого впровадження в децентралізованих системах.

Для вирішення проблем та обмежень безпеки в основному розглядалися стандартизовані моделі шифрів з ваговими коефіцієнтами. В даній роботі представлена реалізація моделі шифру PRESENT, яка включає в себе як процес шифрування, так і процес дешифрування з використанням 80-бітових та 128-бітових ключів для захисту 64-бітових вхідних даних на програмному рівні.

1 ОГЛЯД СИСТЕМИ ІНТЕРНЕТУ РЕЧЕЙ

1.1 Поняття Інтернету речей

Інтернет речей(IoT) перетворює звичне відношення до речей чи пристроїв, створюючи розумні гаджети та розумні міста. Він підключає до мережі інструменти, які віддалені від Інтернету і надає їм нові можливості. Це нова технологічна парадигма, що представляє собою глобальну мережу машин і пристроїв, здатних взаємодіяти один з одним[2].

Термін «Інтернет речей» ввів Пітер Т. Льюїс у вересні 1985 року[7]. Це концепція комунікаційних об'єктів(«речей»), які використовують технології для зв'язку один з одним і з навколишнім середовищем. Ця концепція передбачає дії певних пристроїв без втручання людини. За допомогою IoT кожен реальний об'єкт може бути перетворений на віртуальний: кожна людина і «річ» має локалізований, адресований і зчитуваний аналог в Всесвітній мережі[1,3]. Наприклад, будь-які пристрої в будинках або автомобілях можуть обмінюватися даними між собою, виконувати обробку даних, приймати рішення і виконувати певні дії в залежності від аналізу результатів [3]. Для того, щоб користуватися цим новим середовищем, необхідно забезпечити відповідний рівень безпеки.

1.1.1 Основні технології Інтернету речей.

З точки зору високого рівня, IoT складається з трьох компонентів, а саме: апаратного забезпечення, проміжного програмного забезпечення та представлення [2]. Апаратне забезпечення складається з датчиків та виконавчих механізмів, проміжне програмне забезпечення забезпечує зберігання та обчислювальні засоби, а представлення забезпечує засоби інтерпретації, доступні на різних платформах. Неможливо обробити дані, зібрані з мільярдів датчиків, тому пропонуються контекстно-орієнтовані рішення проміжного програмного забезпечення, щоб допомогти датчику вирішити, які дані є найбільш важливими для обробки [2,3]. За своєю суттю архітектура IoT не пропонує достатнього запасу для виконання необхідних дій, пов'язаних з процесом автентифікації та забезпеченням цілісності

даних. Пристрої в IoT, такі як RFID[8], є сумнівними для досягнення фундаментальних вимог процесу автентифікації, що включає в себе постійний зв'язок з серверами та обмін повідомленнями між вузлами.

П'ять технологій Інтернету речей широко використовуються для розгортання успішних продуктів та послуг на основі Інтернету речей[3,4]:

1) радіочастотна ідентифікація (RFID);

2) бездротові сенсорні мережі (WSN);

3) проміжне програмне забезпечення;

4) хмарні обчислення;

5) прикладне програмне забезпечення IoT.

1.1.1.1 Радіочастотна ідентифікація (RFID)

Радіочастотна ідентифікація (RFID)[8] дозволяє здійснювати автоматичну ідентифікацію та збір даних за допомогою радіохвиль мітки та зчитувача. Мітка має можливість зберігати більше даних, ніж традиційні штрих-коди. Мітка містить дані у вигляді електронного коду товару (EPC), глобальної системи ідентифікації товарів на основі радіочастотної ідентифікації, розробленої Центром Auto-ID. Використовуються три типи міток. Пасивні RFID-мітки покладаються на радіочастотну енергію, яка передається від зчитувача до мітки для живлення мітки. Зазвичай вони не працюють від батареї. Їх можна знайти в паспортах, електронних платежах за проїзд, ланцюгах поставок, та відстеженні на рівні товару. Активні RFID-мітки зазвичай мають власне джерело живлення і можуть ініціювати зв'язок зі зчитувачем. Активні мітки можуть містити зовнішні датчики для моніторингу температури, тиску, хімічних речовин та інших умов. Активні RFID-мітки використовуються на виробництві, в лікарняних лабораторіях і для дистанційного управління IT-активами. Напівпасивні RFID-мітки застосовують батареї для живлення мікрочіпа під час зв'язку. Активні та напівпасивні RFID-мітки коштують дорожче, ніж пасивні[8].

1.1.1.2 Бездротові сенсорні мережі (WSN)

Бездротові сенсорні мережі (WSN)[3] складаються з просторово розподілених автономних пристроїв, оснащених датчиками, які відстежують

фізичні умови або умови навколишнього середовища і можуть взаємодіяти з системами RFID для кращого відстеження умов, таких як їх місцезнаходження, температура чи рух. WSN допускає різноманітні мережеві топології. Останні технологічні досягнення в галузі малопотужних інтегральних схем і бездротового зв'язку зробили ефективні, недорогі й малопотужні мініатюрні пристрої доступними для використання в додатках WSN.

WSN в основному використовуються в логістиці холодного ланцюга, де для транспортування чутливих до температури продуктів застосовуються методи термічного старіння і охолодження упаковки. WSN також використовуються в системах технічного обслуговування та відстеження. Наприклад, General Electric[9] використовує датчики у своїх реактивних двигунах, турбінах та вітроелектростанціях. Аналізуючи дані в режимі реального часу, General Electric економить час і гроші на профілактичному обслуговуванні. Аналогічно, American Airlines[10] використовує датчики, здатні збирати 30 терабайт даних за один рейс для таких послуг, як прогнозоване технічне обслуговування[10].

1.1.1.3 Проміжне програмне забезпечення

Проміжне програмне забезпечення є програмним рівнем, який знаходиться між додатками для полегшення комунікації та введення/виведення для розробників програмного забезпечення [3]. Його здатність приховувати різні технічні деталі має важливе значення для звільнення розробників IoT від програмних сервісів, які не мають прямого відношення до конкретного додатку IoT. Проміжне програмне забезпечення стало популярним у 1980-х роках через його важливу роль у сприянні інтеграції старих і нових технологій. Це також сприяло розвитку нових послуг в розподілених обчислювальних середовищах.

Складні розподілені інфраструктури IoT з великою кількістю різнорідних пристроїв потребують спрощення розробки нових додатків і сервісів, тому використання проміжного програмного забезпечення ідеально підходить для розробки додатків IoT[3]. Наприклад, Global Sensor Network (GSN)[4] - це платформа проміжного програмного забезпечення з відкритим вихідним кодом, яка дозволяє розробляти і розгортати сенсорні сервіси практично з нульовим рівнем

програмування. Більшість архітектур проміжного програмного забезпечення IoT використовують сервіс-орієнтований підхід для підтримки невідомих динамічних топологій мережі.

1.1.1.4 Хмарні обчислення

Хмарні обчислення є моделлю доступу до спільного пулу конфігурованих ресурсів, які можуть бути надані як інфраструктура як послуга (IaaS) або програмне забезпечення як послуга (SaaS). Одним з найважливіших результатів Інтернету речей є величезна кількість даних, що генеруються пристроями, підключеними до Інтернету (Gubbi et al., 2013)[2]. Багато додатків інтернету речей вимагають великого обсягу зберігання даних, величезної швидкості обробки для прийняття рішень в режимі реального часу, а також високошвидкісних широкосмугових мереж для передачі даних, аудіо або відео. Хмарні обчислення забезпечують ідеальне внутрішнє рішення для обробки величезних потоків даних та їх обробки для великої кількості пристроїв Інтернету речей та людей в режимі реального часу.

1.1.1.5 IoT-додатки

IoT сприяє розробці безлічі галузевих і користувацьких додатків IoT[2]. У той час як пристрої та мережі забезпечують фізичний зв'язок, в свою чергу додатки IoT надають надійну взаємодію між пристроями та людьми. Додатки IoT на пристроях повинні гарантувати, що дані чи повідомлення були отримані і оброблені належним чином і вчасно. Наприклад, транспортні чи логістичні додатки відстежують стан товарів, що перевозяться, таких як свіжі продукти, м'ясо та молочні продукти. Під час транспортування постійно контролюється стан збереження (наприклад, температура, вологість) і автоматично вживаються відповідні заходи, щоб уникнути псування, коли з'єднання знаходиться поза зоною досяжності.

У той час як додатки між пристроями не обов'язково вимагають візуалізації даних, все більше і більше додатків Інтернету речей, орієнтованих на суспільне використання, забезпечуючи візуалізацію, щоб представити інформацію кінцевим користувачам в інтуїтивно зрозумілому і простому для сприйняття вигляді і забезпечити взаємодію з навколишнім середовищем. Також важливо акцентувати

на тому, щоб додатки IoT були побудовані зі штучним інтелектом, щоб пристрої могли контролювати навколишнє середовище, виявляти проблеми, спілкуватися один з одним і потенційно вирішувати проблеми без необхідності втручання людини.

1.2 Тенденція розробки системи безпеки Інтернету речей

Пристрої Інтернету речей дозволяють зберігати, аналізувати, контролювати та обмінюватися величезними обсягами даних з іншими мережевими пристроями та користувачами. Крім того, конфіденційність користувачів знаходиться під загрозою через їх обмежений контроль над збором та розповсюдженням своїх даних. Тому питання безпеки та конфіденційності є ключовими викликами для IoT.[3]

Безпека IoT ґрунтується на забезпеченні автентифікації та авторизації вузлів IoT (речей, користувачів, серверів, об'єктів), автентичності, конфіденційності, цілісності та свіжості даних. Рішення з безпеки зазвичай реалізуються на трьох рівнях: мережевому, транспортному та прикладному[3]. Для захисту перших двох рівнів можуть використовуватися наступні протоколи безпеки: IPSec, Host Identity Protocol (HIP), протокол Datagram Transport Layer Security (DTLS), протокол Transport Layer Security (TLS) та Slim Extensible Authentication Protocol Over Local Area Networks (SEAPOL) [11]. В основі рішень для збереження конфіденційності лежать криптографічні примітиви [11].

Загалом, коли говориться про безпеку в системах IoT, це означає, що говориться про безпеку і у вбудованих системах. Вбудовані обчислювальні системи - це системи, які мають виконувати певну функцію в межах більш масштабної механічної або електричної системи. Система вбудовується як частина конкретного пристрою, зазвичай включаючи апаратні та механічні компоненти [3,5]. Ці системи часто базуються на мікроконтролерах (тобто процесорах з інтегрованою пам'яттю та/або периферійними інтерфейсами), але і традиційні мікропроцесори все ще поширені.

Оскільки система управління є частиною більш складної системи, слід враховувати наступні фактори[5]:

- мінімальне енергоспоживання;
- мінімальні габарити і вага;
- захисний кожух, який забезпечує міцність і жорсткість застосованих елементів;
- функція охолодження;
- поєднання мікропроцесора та системної логіки на одному кристалі.

Вбудованим пристроям притаманні обмеження щодо обчислювальної потужності, пам'яті, сховища та енергоспоживання, тому реалізувати достатній обсяг криптографічних функцій нелегко.

1.2.1 Виклики безпеки Інтернету речей

Вразливості є великою проблемою, яка постійно турбує користувачів і організації. Коли мільярди розумних пристроїв, що працюють на різноманітних платформах, підключаються до Інтернету, особливо при переході від настільних комп'ютерів до невеликих пристроїв, вони приносять широкий спектр нових і безпрецедентних проблем для їх власників або користувачів, таких як безпека і конфіденційність, інтероперабельність, довговічність та підтримка, технології та багато іншого [12]. Крім того, пристрої Інтернету речей є легкодоступними та вразливими до багатьох атак на безпеку [3,12], оскільки вони безпосередньо взаємодіють з фізичним світом для збору конфіденційних даних або контролю змінних фізичного середовища, що робить їх привабливою мішенню для зловмисників [12].

Багато організацій та дослідницьких агентств наголошують на необхідності забезпечення безпеки для IoT [13]. Проект Open Web Application Security Project (OWASP) [13] визначив конфіденційність, недостатню автентифікацію/авторизацію, відсутність транспортного шифрування та низький рівень безпеки фізичного рівня серед десяти основних вразливостей для IoT.

Відповідно до еталонної архітектури IoT, безпека IoT має п'ять функціональних компонентів [3,12]:

- управління ідентифікацією,
- автентифікація, авторизація,

- обмін та управління ключами,
- довіра та репутація.

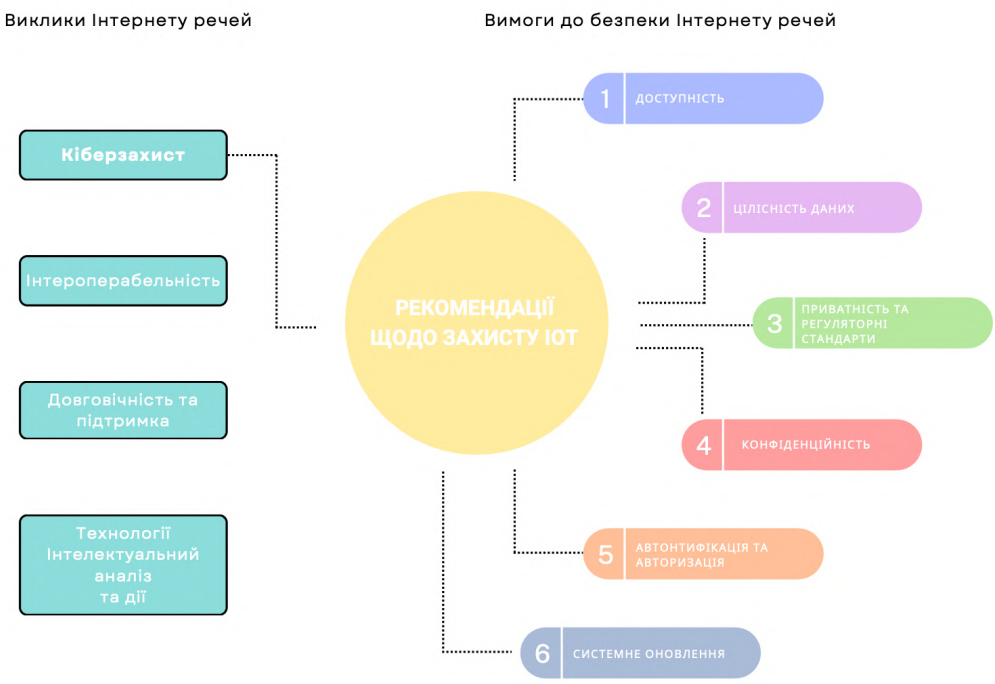


Рисунок 1.1 - напрямок у сфері безпеки Інтернету речей

На рисунку 1.1 зображені основні напрямки у безпеки Інтернету речей. Крім конфіденційності, цілісності та доступності, він включає в себе автентифікацію, не повторюваність контролю доступу, а також конфіденційність, цілісність та доступність. За допомогою криптографічних примітивів всі ці завдання можуть бути виконані.

Для впровадження технології Інтернету речей необхідно зміцнити впевненість користувачів у її безпеці та конфіденційності, що вона не спричинить жодної серйозної загрози цілісності, конфіденційності та повноваженням їхніх даних. За своєю суттю IoT є вразливою до різних видів загроз безпеці, якщо не вжити необхідних заходів безпеки, то є вирогідній виникнення загрози витоку інформації або може бути завдана шкода економіці [3,5]. Такі загрози можуть розглядатися як одна з основних перешкод в IoT.

Датчики збирають у багатьох випадках надзвичайно конфіденційні дані [8]. Захист цих даних є життєво важливим для довіри споживачів, але до цього часу

показники безпеки Інтернету речей були вкрай низькими. Це стосується, як шифрування даних при передачі так і функцінування в стані спокою.

Підключення промислового обладнання до мереж IoT може збільшувати ризик атаки хакерів на ці пристрої. Це викликає велике занепокоєння з огляду на готовність хакерів втручатися в промислові системи, які були підключені до Всесвітньої мережі, але залишилися незахищеними.

Недоліки в програмному забезпеченні - навіть у старому і добре використовуваному коді - виявляються на регулярній основі, але багато пристроїв Інтернету речей не мають можливості бути виправленими, що означає, що вони постійно перебувають у зоні ризику [14]. Хакери зараз активно націлені на пристрої Інтернету речей, такі як роутери і веб-камери, тому що властивий їм брак безпеки дозволяє легко скомпрометувати їх і об'єднати в гігантські ботнети[14].

Хоча пристрої Інтернету речей відіграють величезну роль в обговоренні безпеки Інтернету речей, зосередження всієї уваги на цьому аспекті Інтернету речей не дає повного уявлення про те, чому безпека необхідна і що вона за собою тягне. Існує багато факторів, які роблять безпеку IoT критично важливою сьогодні.

1.2.1.1 Класифікація викликів для безпеки

Централізація. На даний момент архітектури інтелектуальних транспортних засобів базуються на централізованих, брокерських моделях зв'язку [15]. Точніше, центральні хмарні сервери ідентифікують, автентифікують, авторизують і з'єднують всі транспортні засоби. Тим не менш, малоймовірно, що ця модель буде масштабована. Вихід з ладу хмарних серверів може поставити під загрозу всю мережу.

Відсутність конфіденційності. Як правило, конфіденційність користувачів не захищена в нинішніх комунікаційних архітектурах. Іншими словами, обмін даними, що стосуються транспортного засобу, відбувається без дозволу власника. Більше того, запитувачу розкриваються зашумлені або узагальнені дані.

Неоднорідність[5]. Використання з'єднувальних пристроїв в IoT є дуже різноманітним, оскільки вони розгортаються різними організаціями, органами влади та приватними особами. Крім того, їх роздільна здатність, функціональні

можливості та умови експлуатації відрізняються один від одного. Таким чином, забезпечення безперешкодної інтеграції численних пристроїв одночасно є складним завданням. Зокрема, об'єднання таких пристроїв у складну мережу збільшує ступінь складності.

Масштабованість. Використання мініатюрних пристроїв, таких як виконавчі механізми та датчики, зростає через стрімкий розвиток вбудованих технологій. Одночасно з цим необмежено зростають обсяги даних, що створюються такими пристроями. Таким чином, ще одним значним викликом, пов'язаним з IoT - управління кількістю пристроїв і даними, які вони створюють.

Інтероперабельність. Як людські, так і нелюдські об'єкти представляють користувачами(клієнтами) в екосистемі IoT. Кожен користувач, в залежності від середовища і конкретної ситуації, може грати кілька ролей, наприклад, постачальника послуг, споживача даних, постачальника даних і доступного ресурсу в додатках IoT. Для матеріалізації бачення IoT важливо забезпечити безперебійну взаємодію всіх учасників. Якщо кожен учасник управляється по-різному, їх взаємодія посилюється.

Мобільність. Виклики безпеки з точки зору мобільності пов'язані з ефективністю протоколу і мережі IoT. В даний час використання сенсорних мереж, мобільних спеціальних мереж (MANET) і протоколів мобільності транспортних спеціальних мереж (VANET) недостатньо пристосовані для роботи зі стандартними пристроями IoT через значні обчислювальні та енергетичні обмеження. Крім того, необхідна ефективна автентифікація в режимі реального часу замість одноразової початкової конфігурації, враховуючи, що транспортний засіб повинен безперервно автентифікувати інші транспортні засоби, присутні на дорогах.

Загрози безпеці. Можна розглянути приклад продовження зростання кількості функцій автономного водіння в «розумних» транспортних засобах. Отже, порушення безпеки, яке відбувається через несправність, що виникла внаслідок встановлення шкідливого програмного забезпечення, може призвести до автомобільних аварій та наражати на небезпеку учасників дорожнього руху.

В IoT багато взаємопов'язаних пристроїв з обмеженими ресурсами не призначені для виконання дорогих звичайних криптографічних обчислень, що ускладнює реалізацію достатніх криптографічних функцій. Гарантування безпеки та захисту конфіденційності в IoT стає серйозною проблемою при безпечній інтеграції пристроїв з обмеженими ресурсами в IoT, оскільки вони не здатні виконувати достатні криптографічні алгоритми [12].

Оскільки системи Інтернету речей використовують дані в реальному світі, збір даних з пристроїв також може стати об'єктом кібератак. Саме через це контрзаходи, засновані на шифруванні, наразі набувають все більшого значення.

Прогнозується, що під час використання IoT для моніторингу буде генеруватися значний обсяг даних, і життєво важливо зберегти уніфікацію даних [16]. Саме цілісність та автентифікація даних є предметом занепокоєння.

У захищених системах зберігається конфіденційність даних і гарантується, що в процесі обміну повідомленнями дані зберігають свою оригінальність і жодні зміни не залишаються невидимими для системи. IoT складається з безлічі невеликих пристроїв, таких як RFID-мітки, які залишаються без нагляду протягом тривалого часу, тому противнику легше отримати доступ до даних, що зберігаються в пам'яті [8]. Для забезпечення стійкості до атак Sybil в RFID-мітках використовуються методології, засновані на індикації рівня прийнятого сигналу (RSSI) [17].

Для бездротових сенсорних мереж запропоновано багато рішень, які розглядають сенсор як частину мережі Інтернет, підключену через вузли [2]. Однак, в IoT самі сенсорні вузли розглядаються як вузли Інтернету, що робить процес автентифікації ще більш важливим. Цілісність даних також стає життєво важливою і вимагає особливої уваги до збереження їх достовірності.

Найбільша загроза, пов'язана з безпекою, систем IoT від традиційних IT-систем полягає в тому, що навіть використання пристроїв для збору даних з реального світу може стати об'єктом кібератак. Наприклад, метою застосування IoT на заводі є значне підвищення продуктивності та ремонтопридатності за рахунок збору даних з великої кількості датчиків, встановлених на виробничому

обладнанні, їх аналізу та виконання автономного управління в режимі реального часу. Якщо під час цього процесу дані з датчиків будуть фальсифіковані, це призведе до неправильних результатів аналізу та помилкового керування, що може призвести до значних збитків. Крім того, оскільки дані вимірювань і команди управління є комерційною таємницею, пов'язаною з «ноу-хау» виробництва і управління, запобігання витокам також важливо з точки зору конкурентоспроможності. Навіть якщо на даний момент проблеми немає, необхідно враховувати вплив загроз, які можуть стати очевидними в майбутньому.

Всі ці обставини роблять кібербезпеку основним викликом в пристроях IoT з вимогами конфіденційності, цілісності даних, автентифікації та авторизації, доступності, конфіденційності та стандартів регулювання і регулярного оновлення системи [18]. У цьому сценарії криптографія може бути одним з ефективних заходів для гарантування конфіденційності, цілісності, автентифікації та авторизації даних, що проходять через пристрої IoT [18,19]. Криптографія також може бути рішенням для захисту даних, що зберігаються або передаються через мережу. Однак традиційні криптографічні рішення на базі ПК не підходять для більшості пристроїв IoT, включаючи датчики і RFID-мітки, через їх обмеженість в ресурсах. Полегшені версії цих рішень можуть вирішити цю проблему. Як правило, полегшені версії обчислювальної криптографії відомі як малоресурсна криптографія (LWC).

1.2.2 Вразливості

Вразливість - це слабке місце в системі або її конструкції, яке дозволяє зловмиснику виконувати команди, отримувати доступ до несанкціонованих даних та/або здійснювати атаки на відмову в обслуговуванні [20].

Вразливостями системи IoT можуть бути слабкі місця в апаратному або програмному забезпеченні системи, політиках і процедурах, що використовуються в системі, а також в самих користувачах системи [3,5,11,12].

Вразливості програмного забезпечення можна знайти в операційних системах, додатках та керуючому програмному забезпеченні, такому як протоколи зв'язку та пам'ять пристрою. Існує багато факторів, які сприяють виникненню

помилок при розробці програмного забезпечення, включаючи людські помилки та складність програмного забезпечення. Наслідками нерозуміння вимог є запуск проектів без плану, погана комунікація між розробниками та користувачами, нестача ресурсів, навичок і знань, а також нездатність керувати та контролювати систему [11].

Інтернет речей є надзвичайно вразливим [21], оскільки його компоненти можуть піддаватися фізичним атакам, коли залишаються без нагляду протягом тривалого часу. По-друге, бездротове середовище зв'язку надзвичайно полегшує підслуховування. Як зазначалося, вимоги до енергоспоживання компонентів IoT є низькими, як з точки зору споживаної ними енергії, так і з точки зору їх обчислювальних потужностей. Реалізація традиційних, обчислювально дорогих алгоритмів безпеки призведе до зниження продуктивності пристроїв з обмеженими потужністями [14].

1.2.2.1 Категорії вразливостей

Стандартні компоненти вразливості також впливають на мільйони пристроїв, як показано в Ripple20[22] та URGENT/11[23,24]. Крім самих пристроїв, до атак на системи також можуть призводити вразливості у веб-додатках та пов'язаному з ними програмному забезпеченні на пристроях IoT. Оператори шкідливих програм шукають такі можливості, навіть коли їм відомо про попередні вразливості.

Шкідливе програмне забезпечення. Хоча більшість пристроїв IoT мають обмежену обчислювальну потужність, вони все одно можуть бути заражені шкідливим програмним забезпеченням [14]. Це те, що кіберзлочинці дуже ефективно використовують в останні роки. Шкідливе програмне забезпечення для бот-мереж IoT є одним з найпоширеніших варіантів, оскільки воно є універсальним та економічно вигідним для кіберзлочинців. Найвідоміша атака сталася в 2016 році, коли Mirai вивела з ладу основні веб-сайти та сервіси, використовуючи велику кількість популярних пристроїв IoT [25]. Також застосовуються інші сімейства шкідливих програм включають майнери криптовалют та програми-вимагачі.

Ескалація кібератак. Заражені пристрої часто використовуються в розподілених атаках на відмову в обслуговуванні (DDoS) [24]. Викрадені пристрої також можуть використовуватися в якості бази атаки для зараження декількох комп'ютерів і маскування шкідливої активності, або в якості точки входу для стороннього трафіку в корпоративну мережу. Хоча організації можуть здаватися більш прибутковими цілями, розумні будинки також піддаються приголомшливій кількості непередбачених кібератак [24].

Крадіжка і невідоме розголошення інформації. Як і все, що пов'язано з Інтернетом, підключені пристрої збільшують шанси бути знайденими в мережі. Важлива технічна інформація і навіть персональні дані можуть несвідомо зберігатися на цих пристроях і ставати цілями для атак [20].

Пристроями погано керують і неправильно їх налаштовують. Відсутність обізнаності про безпеку, погана гігієна паролів і загальне погане управління пристроями можуть сприяти успіху цих загроз. Користувачам також може не вистачати знань і навичок для впровадження відповідних заходів безпеки, а постачальникам послуг і виробникам може знадобитися допомогти своїм клієнтам досягти кращого захисту.

Галузі не вистачає далекоглядності і у неї мало часу на розробку стратегій і засобів захисту від відомих загроз в екосистемі IoT, що розвивається [14]. Передбачення нових викликів є однією з причин, чому дослідження в області безпеки IoT повинні бути безперервними. Ось деякі нові виклики, які потребують моніторингу.

Складні умови. У 2020 році більшість домогосподарств США мали в середньому 10 підключених пристроїв. У цьому дослідженні складне середовище IoT визначається як взаємопов'язана мережа з 10 або більше IoT-пристроїв [26]. Таке середовище практично неможливо контролювати через складну мережу взаємопов'язаних функцій. Ігнорування неправильної конфігурації в такій ситуації може мати плачевні наслідки і навіть загрожувати фізичній безпеці будинку.

Поширення віддаленої роботи. Пандемія Covid-19 перекреслила багато очікувань від 2020 року [27]. Вона призвела до широкого впровадження механізмів

роботи з дому в організаціях по всьому світу та сприяла більшій залежності від домашніх мереж [27]. Пристрої Інтернету речей також виявилися успішними для багатьох користувачів, які працюють вдома. Ці події підкреслюють необхідність перегляду практики безпеки IoT.

Зв'язок 5G. Перехід до 5G пов'язаний з великими очікуваннями і надіями. Цей розвиток також призведе до розвитку інших технологій. На сьогоднішній день більшість досліджень, присвячених 5G, зосереджені на тому, як він вплине на бізнес і як вони можуть розгорнути його безпечно [28].

1.2.2.2 Атаки на систему Інтернету речей

Атака - це дія, спрямована на пошкодження системи або порушення нормальної роботи шляхом використання вразливостей з використанням різноманітних методів та інструментів. Зловмисники здійснюють атаки для особистого задоволення або винагороди [29] для досягнення своїх цілей. Зусилля зловмисника вимірюються його досвідом, ресурсами та мотивацією, що називається вартістю атаки [30]. Зловмисник - це той, хто становить загрозу для цифрового світу. Це можуть бути хакери, злочинці або навіть уряди [31].

Сама атака може приймати багато форм, включаючи активні мережеві атаки для моніторингу незашифрованого трафіку на предмет отримання конфіденційної інформації, пасивні атаки (такі як моніторинг незахищеного мережевого трафіку для розшифровки слабко зашифрованого трафіку та отримання автентифікаційної інформації), атаки зближення, інсайдерська торгівля тощо. Види кібератак включають[31]:

- Фізичні атаки:

Цей вид атак спрямований на втручання в роботу апаратних компонентів. Через розподілений характер Інтернету речей, більшість пристроїв, як правило, працюють у зовнішньому середовищі, яке є дуже вразливим до фізичних атак.

- Розвідувальні атаки

Несанкціоноване виявлення та картографування систем, сервісів або вразливостей. Прикладами розвідувальних атак є сканування мережевих портів

[29,31], перехоплення пакетів, аналіз трафіку та надсилання запитів щодо інформації про IP-адреси.

- Відмова в обслуговуванні (DoS):

Цей вид атаки є спробою зробити машину або мережевий ресурс недоступним для його цільових користувачів. Через низькі можливості пам'яті та обмежені обчислювальні ресурси більшість пристроїв в IoT вразливі до атак на збереження ресурсів.

- Атаки доступу

Неавторизовані особи отримують доступ до мереж або пристроїв, до яких вони не мають права доступу. Існує два різних типи атак на доступ: перший - фізичний доступ, коли зловмисник може отримати доступ до фізичного пристрою. Другий - це віддалений доступ, який здійснюється до пристроїв, підключених до IP-мережі.

- Атаки на конфіденційність

Захист конфіденційності в IoT стає все більш складним завданням через великі обсяги інформації, які легко доступні через механізми віддаленого доступу. Найбільш поширеними атаками на конфіденційність користувачів є наступні:

- Видобуток даних

Дозволяє зловмисникам виявити інформацію, яка не передбачається в певних базах даних.

- Кібершпигунство

Використання техніки злому та шкідливого програмного забезпечення для шпигунства або отримання секретної інформації приватних осіб, організацій або уряду.

- Підслуховування:

Прослуховування розмови між двома партнерами [31].

- Відстеження

Пересування користувача може бути відстежено за унікальним ідентифікаційним номером (UID) пристрою. Відстеження місцезнаходження

користувача полегшує його ідентифікацію в ситуаціях, коли він бажає залишитися анонімним.

- Атаки на основі паролів:

Зловмисники намагаються скопіювати дійсний пароль користувача. Ця спроба може бути здійснена двома різними способами:

1) словникова атака - перебір можливих комбінацій букв і цифр для вгадування паролів користувачів;

2) атака грубої сили - використання інструментів злому для перебору всіх можливих комбінацій паролів з метою розкриття дійсних паролів.

1.2.3 Проблеми традиційної криптографії

Ключовими проблемами реалізації традиційної криптографії в пристроях IoT є наступні [32]:

- Обмеженість пам'яті (регістри, ОЗП, ПЗП);
- Зниження обчислювальної потужності;
- Невелика фізична площа для реалізації збірки;
- Низький заряд батареї (або відсутність батареї);
- Реакція в реальному часі.

Більшість пристроїв IoT (таких як RFID-мітки і датчики) мають невеликі розміри і оснащені обмеженими ресурсами, такими як невелика пам'ять (RAM, ROM) для зберігання і запуску додатків, низька обчислювальна потужність для обробки даних, обмежена потужність батареї (або відсутність батареї у випадку пасивних RFID-міток) [8], невелика фізична площа для розміщення в збірці [2,14] . Крім того, більшість пристроїв IoT працюють в режимі реального часу, де швидке і точне реагування з необхідним рівнем безпеки з використанням наявних ресурсів є складним завданням [33]. Розробники пристроїв IoT стикаються з декількома ризиками і проблемами, включаючи енергетичну потужність [11] і безпеку даних [14].

Якщо примітив реалізовано апаратно, наступні метрики описують ефективність реалізації.

Споживання пам'яті і, отже, розмір реалізації об'єднуються в площу воріт, яка вимірюється в еквівалентах воріт (Gate Equivalents, GE)[34]. Чим вона нижча, тим краще.

Пропускна здатність, що вимірюється в бітах або байтах в секунду, відповідає кількості простого тексту, що обробляється за одиницю часу. Чим вона вища, тим краще. Затримка, що вимірюється в секундах, відповідає часу, необхідному для отримання виходу схеми після подачі на її вхід. Чим вона менша, тим краще. Споживана потужність, що вимірюється у Ватах, характеризує кількість енергії, необхідної для роботи схеми. Чим воно нижче, тим краще.

При апаратній реалізації малоресурсної криптографії важливими показниками є розмір коду, споживання оперативної пам'яті (ОЗП) і енергоспоживання[1, 6]. Отже, не існує стандартного порівняння щодо апаратної реалізації різних алгоритмів. З точки зору споживання пам'яті, для малоресурсних примітивів краще використовувати менші блоки з меншими ключами[6]. Однак, для "запису" ключів у пристрої використовуються структури тільки для читання, а не пам'ять для читання/запису. Енергоефективність лежить в основі апаратної реалізації, поряд з латентністю, час виконання заданої операції також береться як один з критеріїв при проектуванні легких блокових шифрів.

Відповідно, у табл. 1.1 наведено порівняння, яке дозволяє продемонструвати різницю між асиметричною та симетричною криптографією [34].

Таблиця 1.1 – Порівняння методів криптографії [34].

Параметри	Особливості різновидів реалізації	
	Криптографія з симетричним ключем	Криптографія з асиметричним ключем
Ключі	Один загальний приватний ключ	Унікальна пара приватного та публічного ключів. Генерація відкритих ключів залежить від криптографічних алгоритмів, заснованих на односторонніх математичних функціях.

Продовження таблиці 1.1 – Порівняння методів криптографії

Кількість ключів	Експоненціально пропорційні кількості користувачів	Лінійно пропорційні кількості користувачів
Швидкість та складність	Це прості алгоритми, і завдяки цьому процес шифрування може бути здійснений швидко.	Це набагато складніший процес, ніж шифрування з симетричним ключем, і він відбувається повільніше через те, що для використання різних ключів потрібно більше часу.
Апаратна складність	Менш комплексна реалізація апаратного забезпечення, оскільки вона реалізує алгоритми з простих операції які потребують відносно недорогого апаратного забезпечення.	Більш складна реалізація апаратного забезпечення, яка обчислює важкі алгоритми які потребують більш потужне апаратне забезпечення.
Використання	Здебільшого використовується, для передачі великих обсягів даних.	Використовується в невеликих транзакціях, в першу чергу для автентифікації та встановлення безпечного каналу зв'язку перед фактичною передачею даних.
Алгоритми	RSA, ECC, DSA	Stream cipher: Chacha, Trivium, WG-8, Grain 128, Espresso, Block Ciphers: AES, 3DES, DES Blowfish, Twofish, Curupira, TEA, RECTANGLE, SIMON PRESENT.

Шифрування стандартно застосовується на канальному рівні системи зв'язку, як приклад - мобільний телефон. Навіть у такому випадку шифрування на прикладному рівні є ефективним для забезпечення наскрізного захисту даних від пристрою до сервера та забезпечення безпеки незалежно від системи зв'язку. Тоді шифрування повинно застосовуватися на процесорі, що обробляє додаток, і на невикористовуваних ресурсах, а отже, бажано, щоб воно було якомога легшим[34].

1.3 Децентралізація Інтернету речей

Централізовані архітектури часто працюють неефективно в сучасних середовищах Інтернету речей[36]. Ефективна обробка даних, прийняття рішень і використання ресурсів вимагають нових підходів, таких як децентралізована платформа IoT. Очевидно, що для забезпечення її стійкості необхідний відповідний дизайн, який сам по собі є складним завданням, не кажучи вже про оптимальну розробку. В якості кроку вперед було визначено характеристики мережевого трафіку серед найбільш поширених послуг IoT, згрупованих у вісім областей застосування відповідно до Mocnej et al. (2018)[37, 38, 39].

Оскільки Блокчейн та Інтернет речей набувають все більшої популярності серед суспільства протягом цих років, компанії та команди стартапів розробляють нові системи або програмне забезпечення шляхом їх інтеграції. Деякі з найвідоміших команд розробників будуть показані нижче, і вони прагнуть вирішити різні види попиту на ринку. Slock.it [40] намагається зробити процес оренди та повернення квартири Airbnb повністю автоматизованим за допомогою смарт-контракту для дистанційного керування дверним замком.

1.3.1 Блокчейн

Блокчейн - це розподілений реєстр, в якому стан його транзакцій підтримується розподіленим консенсусом між недовіреними суб'єктами без необхідності централізованого довіреного органу третьої сторони. Ці децентралізовані суб'єкти називаються майнерами [40, 41]. Отримаючи доказ роботи, майнери об'єднують підтверджену транзакцію в блоки, таким чином генеруючи хеш поточного блоку, який включає хеш попереднього блоку.

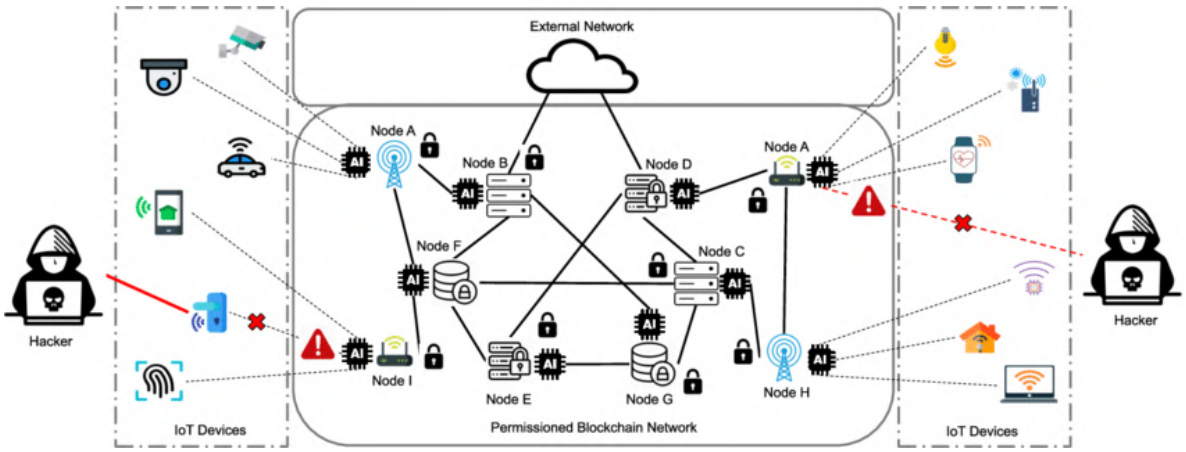


Рисунок 1.2 - Архітектура на основі штучного інтелекту та блокчейну для гетерогенних систем IoT [15, 39]

Суцільними лініями позначені з'єднання і лінії зв'язку між вузлами, що беруть участь в мережі блокчейн. Пунктирними лініями позначені лінії зв'язку між пристроями IoT і периферійними вузлами. Червоні лінії (суцільні і пунктирні) вказують на вразливі з'єднання і підозрілі комунікації IoT [15, 39].

Ці докази роботи вимагають високої обчислювальної потужності процесора, таким чином захищаючи блокчейн від зловмисників.

Блокчейн може зберігати дані і виконувати обчислення, які можуть бути виконані цими децентралізованими блоками для визначення стану блокчейна поза ланцюжком. Ці обчислення поза ланцюжком називаються смарт-контрактами. При використанні смарт-контрактів забезпечується система, яка забезпечує децентралізований контроль політик доступу до даних, не покладаючись на зовнішніх постачальників послуг, таким чином забезпечуючи безперебійне надання послуг користувачам IoT [41].

1.3.2 Набір характистик децентралізованої системи

Набір характеристики, яким повинна відповідати децентралізована архітектура Інтернету речей[39-41].

1.3.2.1 Мультимережевий підхід

Платформи IoT призначені для підключення пристроїв незалежно від використовуваної мережевої технології. IoT є гетерогенним, і оскільки різні обмеження можуть вимагати різних рішень, існує безліч мережевих технологій,

розроблених спеціально для різних частин IoT. Всі різнорідні пристрої приховують базову мережеву технологію в стандартизованій формі[40].

1.3.2.2 Масштабована та інтероперабельна реалізація

Перехід від рішень, що залежать від додатків, до рішень, що не залежать від додатків, є дуже бажаним. Оскільки умови і очікування прикладних областей IoT можуть динамічно змінюватися з часом, адаптивна архітектура IoT повинна розпізнавати і своєчасно реагувати на ці зміни. Таким чином, платформа IoT в майбутньому повинна бути масштабованою і неінвазійною, а також готовою до інтеграції нових додатків за будь-яких обставин.

1.3.2.3 Низьке енергоспоживання

Ефективне використання ресурсів та низьке енергоспоживання повинні бути пріоритетами заради досягнення бажаної стійкості рішення. Платформа Інтернету речей повинна підтримувати такі механізми цього роду, та інші методи оптимізації для досягнення поліпшень у використанні енергії[39].

1.3.2.4 Інтуїтивно зрозуміле управління даними та пристроями

Щоб йти в ногу зі швидким розвитком та інноваціями, ефективне управління пристроями та збір даних повинні бути присутніми в IoT за замовчуванням. Платформа повинна підтримувати як індивідуальне, так і централізоване управління для забезпечення доступності, контролю та модифікацій. Крім того, багато рутинних дій повинні бути автоматизовані, щоб зробити процес конфігурації простішим та інтуїтивно зрозумілішим.

1.3.2.5 Штучний інтелект на периферії

Штучний інтелект (ШІ) та машинне навчання (МН) з кожним роком набувають все більшої популярності. Сьогодні вони використовуються в переважній більшості технологій та послуг. Інтернет речей не є винятком, де ШІ найчастіше використовується в хмарі для забезпечення низки покращень. Однак, у випадку децентралізованої архітектури, частина послуг повинна бути перенесена з хмари на кордон мережі, включаючи ШІ. Найбільш підходящим пристроєм для розміщення ШІ є необмежений шлюз, який пропонує розумні обчислювальні

ресурси і стабільне джерело енергії. Децентралізована платформа IoT також повинна бути розроблена для підтримки контекстної обізнаності.

Ці п'ять ключових особливостей були ретельно включені в архітектуру запропонованого децентралізованого рішення[41].

2 МАЛОРЕСУРСНА КРИПТОГРАФІЯ

2.1 Визначення малоресурсної криптографії

Якщо один з вузлів буде скомпрометований, мережа може серйозно постраждати. Однак реалізувати достатню кількість криптографічних функцій на обмежених пристроях нелегко через обмеженість їхніх ресурсів.

Криптографічні технології розвиваються. Інтенсивно вивчаються нові методи атаки, проектування та реалізації. Однією з найсучасніших методик є «Малоресурсна криптографія» (Lightweight Cryptogrphy, LWC)[1,6].

В умовах обмежених ресурсів дуже важко реалізувати якісні криптографічні алгоритми через розмір реалізації, швидкість або пропускну здатність і споживання енергії.

За останнє десятиліття було запропоновано низку малоресурсних криптопримітивів (включаючи блокові шифри, хеш-функції, коди автентифікації повідомлень та потокові шифри), які мають переваги у продуктивності порівняно з традиційними криптографічними стандартами. Ці примітиви відрізняються від звичайних алгоритмів припущеннями, що легкі примітиви не призначені для широкого кола застосувань, і можуть накладати обмеження на потужність зловмисника.

Малоресурсна криптографія - сукупність криптографічних примітивів, методів та шифрів, які можуть бути застосовані в пристроях, що не здатні забезпечити більшість існуючих шифрів і мають обмежені ресурси (пам'ять, потужність, розміри) для роботи. Малоресурсна криптографія націлена на дуже широкий спектр пристроїв з обмеженими ресурсами, таких як кінцеві вузли IoT і RFID-мітки [8], які можуть бути реалізовані як на апаратному, так і на програмному забезпеченні з різними технологіями зв'язку. Для середовища з обмеженими ресурсами дуже важко реалізувати стандартні криптографічні алгоритми через розмір реалізації, швидкість або пропускну здатність та споживання енергії. Малоресурсна криптографія забезпечує компроміс між вартістю реалізації,

швидкістю, безпекою, продуктивністю та енергоспоживанням на пристроях з обмеженими ресурсами.

2.1.1 Стандартизація

В ISO/IEC 29192[42] обговорювались властивості криптографії з низькими ресурсами в ISO/IEC JTC 1/SC 27[43]. Новий проект зі стандартизації малоресурсної криптографії ISO/IEC 29192[42] знаходиться в процесі стандартизації. У стандарті ISO/IEC 29192 атрибути малоресурсної криптографії описуються на основі цільової платформи. В апаратних реалізаціях розмір матриці та/або енергоспоживання є важливими показниками для оцінки характеристик низьких ресурсів. У програмних реалізаціях менший обсяг коду та/або оперативної пам'яті краще підходить для малоресурсних додатків. З точки зору властивостей реалізації, малоресурсні примітиви перевершують традиційні криптографічні примітиви, які в даний час використовуються в протоколах безпеки в Інтернеті, таких як IPsec, TLS. Загальна класифікація наведена на рисунку 2.1.

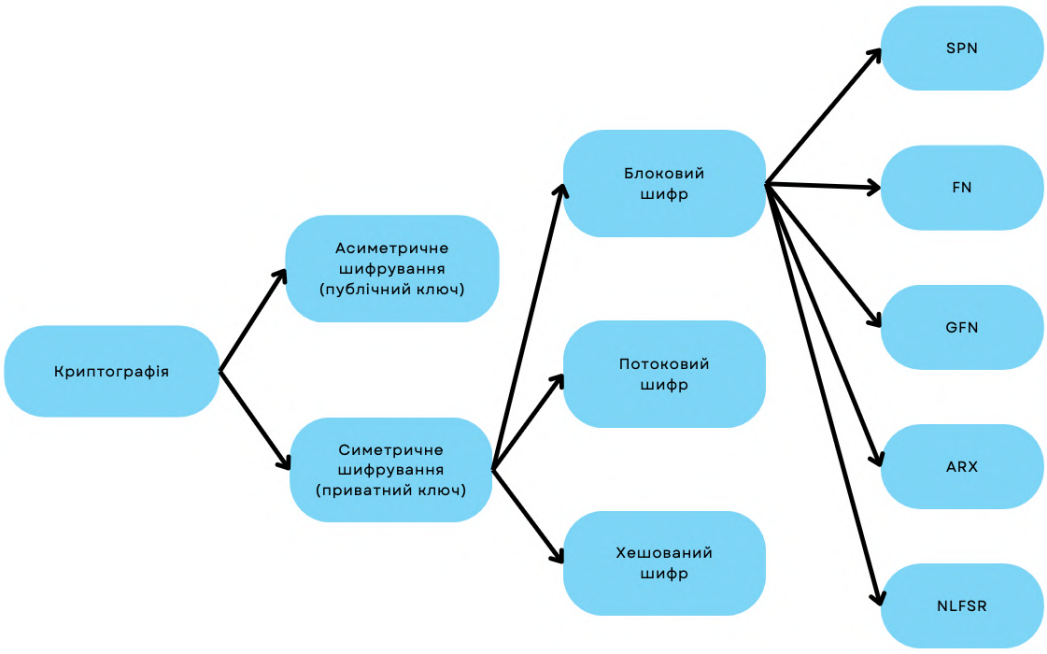


Рисунок 2.1 - Структурна класифікація малоресурсної криптографії[6,16,42]

У світі передачі даних зазвичай використовується AES [44], оскільки він є безпечним стандартом. Хоча AES є безпечним стандартом, він не підходить для пристроїв з апаратними обмеженнями. Відомо, що нове рішення, відмінне від AES,

необхідне особливо для додатків, які повинні працювати з низьким енергоспоживанням.

Для досягнення цієї мети було створено багато малоресурсних алгоритмів. Більшість розроблених алгоритмів є алгоритмами блокового шифрування. В ході дослідження були проаналізовані малоресурсні криптографічні алгоритми PRESENT [45], CLEFIA [46], PICCOLO [47], PRINCE [48] та LBLOCK [49]. Причину вибору саме цих алгоритмів можна пояснити наступним чином. PRESENT та CLEFIA стандартизовані як алгоритми малоресурсного блокового шифрування документом ISO/IEC 29192-2:2012 [42], який визначає вимоги до малоресурсної криптографії.

Для правильної оцінки малоресурсної криптографії необхідно враховувати точний тип схеми (наприклад, годинник), пам'ять, зберігання внутрішніх станів і станів ключів. Однак, це не означає, що менший розмір блоку та ключа є кращим, оскільки це може призвести до незахищеності від атак з використанням пов'язаного ключа [18,32].

2.2 Апаратна реалізація

Специфічні вимоги до апаратного забезпечення будь-якого цифрового пристрою зазвичай вимірюються з точки зору вимог до пам'яті, площі затвору, затримки, пропускної здатності, а також потужності та енергоспоживання наступним чином.

2.2.1 Вимоги до пам'яті

Як правило, вимірюється в GE [9]. ОЗП (оперативна пам'ять), необхідна для зберігання проміжних значень, які можуть бути використані в обчисленнях, і ПЗП (пам'ять тільки для читання), необхідна для зберігання програмного коду (алгоритму), і фіксованих даних, таких як S-box або жорстко закодований круглий ключ [42].

Розмір фізичної області, необхідної для реалізації/запуску алгоритму на схемі, може бути визначений в термінах логічних блоків для польових програмованих вентильних матриць (FPGA), або в еквівалентах вентилів (GE) для реалізації ASIC (один GE еквівалентний двох вхідному вентилю NAND) [9]. Вона

вимірюється в мкм^2. Зазвичай, недорога RFID-мітка здатна вмістити від 1000 до 10 000 воріт, з яких з міркувань безпеки може бути доступно лише від 200 до 2000 [9].

2.2.2 Затримка

Це міра часу між початковим запитом на виконання операції та отриманням результату (час між отриманням зашифрованого тексту з відкритого тексту) [42].

2.2.3 Пропускна здатність

Пропускна здатність в апаратному забезпеченні може бути виміряна в одиницях простого тексту, що обробляється за одиницю часу (біти в секунду) при частоті 100 КГц, тоді як в програмному забезпеченні - це середній обсяг простого тексту, що обробляється за один такт процесора при частоті 4 МГц [42].

2.3 Програмна реалізація

Малоресурсні примітиви також можуть бути реалізовані в програмному забезпеченні, як правило, для мікроконтролерів. У цьому випадку відповідними метриками є споживання оперативної пам'яті, розмір коду та пропускна здатність [50].

Споживання оперативної пам'яті відповідає обсягу даних, що записуються в пам'ять при кожному обчисленні функції - розміру коду, фіксованого обсягу даних, необхідний для обчислення функції, незалежно від її вхідних даних.

Як і раніше, пропускна здатність вимірює середній обсяг даних, що обробляються за один такт [42].

Перші дві величини вимірюються в байтах і повинні бути мінімізовані, тоді як остання величина вимірюється в байтах за цикл і повинна бути максимізована. Знову ж таки, ці три величини не є незалежними. Тому зменшення кількості таких операцій призводить до зменшення споживання пам'яті та збільшення пропускної здатності.

Для малоресурсні реалізацій прикладного програмного забезпечення найкращими метриками є розмір реалізації та споживання пам'яті, а також пропускна здатність (байти за цикл) [50]. Чим менше, тим краще.

2.4 Асиметричне шифрування

Криптографія з асиметричним ключем називається криптографією з відкритим ключем, оскільки для її роботи необхідна пара відкритих і закритих ключів [35]. В основі генерації відкритого ключа лежить криптографічний алгоритм, що базується на односторонніх математичних функціях. Таким чином, відкритий ключ може поширюватися публічно без шкоди для безпеки, оскільки збереження приватного ключа в таємниці має вирішальне значення для ефективної безпеки [35].

Зовсім недавно фокус малоресурсної криптографії змістився в бік криптографії з асиметричним ключем, але результати ще не є стабільними і плідними, як у криптографії з симетричним ключем. Малоресурсні асиметричні алгоритми складні в роботі і не є ефективними в часі. Розмір операндів та невпинний розвиток моделей атак також роблять ці алгоритми вразливими [35,42,43].

У системах такого типу будь-яка особа може зашифрувати повідомлення за допомогою відкритого ключа одержувача, але зашифроване повідомлення може бути розшифроване тільки за допомогою приватного ключа одержувача. Асиметричні шифри є набагато більш вимогливими до обчислювальних ресурсів, ніж їх симетричні аналоги.

Деякі з дуже важливих асиметричних алгоритмів є наступними[51]:

Rivest-Shamir Adleman (RSA) - зворотна процедура дуже складна для зловмисника, а також складна до виготовлення приватного ключа з відкритого ключа. Таким чином, цей метод є високо захищеним, але генерація ключів є складною, і процес є дуже повільним.

Diffie-Hellman - закритий ключ дуже короткий, як наслідок, процес відбувається швидше. Через короткий закритий ключ він піддається більшій кількості атак, а також процес вразливий до атак типу "людина посередині".

Алгоритм цифрового підпису (DSA) - цей процес швидший і вигідніший, ніж інші асиметричні алгоритми, але цифрові підписи мають короткий термін служби, а їх спільне використання ускладнене [51].

Криптографія еліптичних кривих (ECC) - хоча вона є більш складною і важкою для реалізації, але споживає менше енергії. Серед різних типів асиметричних алгоритмів ECC є найбільш сприятливим для реалізації в пристроях з обмеженим доступом [51].

(Метод, запропонований ISO/IEC 29192-4:2013[52] включає в себе ELLI (Elliptic Light), cryptoGPS і ALIKE. ELLI використовує еліптичні криві разом з рукостисканням Діффі-Хеллмана між RFID-міткою та RFID-зчитувачем [8].

2.5 Симетричне шифрування

Зосередимось на криптографії з симетричним ключем, яка може широко застосовуватися на пристроях, які піддаються жорстким ресурсним обмеженням. Криптографія з симетричним ключем складається з основних функцій, таких як блокові або потокові шифри (криптографічні примітиви) і методів застосування основної функції до пакету, які називаються режимом роботи блокового шифру для шифрування та/або автентифікації [53].

2.5.1 Визначення

В симетричному шифруванні використовується однаковий ключ як для шифрування, так і для розшифрування даних. Цей метод шифрування є безпечним і відносно швидким [54]. Але головним недоліком шифрування з симетричним ключем є спільне використання ключа двома сторонами, які між собою спілкуються. Зловмисник може розшифрувати дані, якщо має доступ до ключа. Алгоритми з симетричним ключем забезпечують конфіденційність і цілісність даних, але не гарантують автентифікацію. Цей тип шифрування використовує три типи алгоритмів, заснованих на хешуванні, потоковому та блоковому шифрах[54].

2.5.2 Хешування

Хеш-функція приймає на вхід повідомлення довільного розміру і видає на виході повідомлення з фіксованим розміром [56]. Відповідні обчислення включають код автентифікації повідомлення (MAC), цілісність даних і цифрові підписи[55]. Хеш-функція не є безколізійною, якщо вона відображає повідомлення будь-якої довжини в рядки фіксованої довжини. Результат відомий як відбиток

[56]. Для n-бітної ідеальної хеш-функції зловмисник виконує $2n/2$ та $2n$ обчислень, щоб отримати колізію та (друге) попереднє зображення.[56].

Дві основні частини хеш-функції - це функція побудови та функція стиснення[56]. Функція побудови призначена для ітерації функції стиснення. Ідеальна хеш-функція повинна мати властивості випадкового оракула. Навіть якщо випадковий оракул не існує, конструкція хеш-функції повинна відповідати критерію безпеки.

2.5.3 Потоковий шифр

Потоковий шифр - це шифр з симетричним ключем, в якому цифри відкритого тексту поєднуються з псевдовипадковим потоком цифр шифру [57]. У цьому типі кожна цифра відкритого тексту шифрується по черзі з аналогічною цифрою ключового потоку, щоб в результаті отримати цифру потоку шифрованого тексту [57]. Mickey V2 є малоресурсним потоковим шифром[58]. Він створює ключовий потік з 80-бітового ключа і вектора ініціалізації змінної довжини (до 80 біт). Ключовий потік має максимальну довжину 240 біт. Trivium[59] також є одним з малоресурсних потокових шифрів має низьку вимогливість до апаратного забезпечення. Він використовує 80-бітний ключ і генерує до 264 біт вихідних даних. Grain та Enocoro є малоресурсних потоковими шифрами, які мають 80-бітний та 128-бітний ключ відповідно. Grain має відносно низьке енергоспоживання та об'єм пам'яті [60]. Enocoro визначений компанією Hitachi і включений в міжнародний стандарт ISO/IEC 29192[42] для методу малоресурсних потокових шифрів.

2.5.4 Блоковий шифр

Блокові шифри - різновид симетричних шифрів, в яких обробляється відразу весь блок. Блокові шифри використовуються для побудови хеш-функцій та кодів автентифікації повідомлень (MAC) [61]. Полегшені блокові шифри базуються на структурі Мережа підстановки-перестановки (SPN) та Структурі Фейстеля. Структура Фейстеля використовує свою кругову функцію лише на половині стану. PRESENT та CLEFIA для блокових методів визначені як стандарти малоресурсної криптографії в рамках ISO/IEC 29192-2:2012[42].

AES [44, 62] є класичним прикладом алгоритму на основі SPN, стандартизований NIST, працює на 128-бітному блоці з 128, 192 та 256-бітними варіантами ключів [62]. Мінімальна вимога GE, зафіксована для AES, становить близько 2400 GE (на 23% менше, ніж звичайна) [62], що все ще є важким для деяких невеликих додатків у реальному часі [63]. Це показує порівняно ефективну продуктивність при забезпеченні додатковими ресурсами.

Одним з основних, що вказують на перспективність заміни AES для малоресурсної криптографії, є PRESENT [45, 62]. Він працює з 64-бітними блоками і використовує метод підстановки-пермутації. Відомим малоресурсним блоковим шифром може бути CLEFIA, який був розроблений компанією Sony і має 128, 192 і 256-розрядні ключі та 128-розрядні розміри блоків. Було розроблено багато алгоритмів малоресурсної криптографії, серед яких декілька симетричних алгоритмів використовують AES (Advanced encryption standards).

Симетрична шифрування допомагає при проектуванні однієї і тієї ж схеми для шифрування і дешифрування з мінімальними накладними витратами. Таким чином, основною перевагою структури Фейстеля є використання одного і того ж програмного коду для процесу шифрування і дешифрування. Це призводить до низького використання пам'яті. Вона може бути реалізована на апаратних засобах з низькою середньою потужністю. Фейстелівська структура не підходить для конструкцій з малою затримкою. SPN буде швидше, але без розкладу ключів. Відсутність ключового розкладу зробить її вразливою до атак. При однаковій величині запасу стійкості та однакових витратах енергії, структура SPN є більш придатною, оскільки вона вимагає меншого раунду виконання. За аналогічних умов SPN матиме менші енерговитрати.

Основними параметрами для оцінки малоресурсного блокового шифру є розмір ключа, розмір блоку, тип структури та кількість раундів. В роботі [19] автори вважають, що малоресурсний шифр повинен відповідати трьом вимогам: мінімальна площа кремнію або обсяг пам'яті, низьке енергоспоживання, достатній рівень безпеки.

З наведених вище причин блоковому шифру надається перевага в пристроях IoT з обмеженими ресурсами над потоковим шифром. Симетричний малоресурсний блоковий шифр використовує одну з наступних структур[35]:

- Мережа підстановок і перестановок (SPN)
- Мережа Фейстеля (FN)
- Загальна мережа Фейстеля (GFN)
- Add-Rotate-XOR (ARX)
- Регістр зсуву зі зворотним зв'язком без лінійного зворотного зв'язку (NLFSR)
- Гібридний

Мережа підстановок і перестановок (SPN) обробляє дані за допомогою серії підстановок (S-box) і перестановок (таблиця), змінюючи дані і остаточно формулюючи їх для наступного раунду.

Мережа Фейстеля (FN) - це багатораундовий шифр, який ділить вхідний блок на дві частини і працює тільки над половиною (дифузія) в кожному раунді шифрування або дешифрування. Між раундами ліва і права половини блоку міняються місцями. Узагальнена мережа Фейстеля (GFN) є узагальненою формою класичного шифру Фейстеля. У GFN вхідний блок розбивається на два або більше підблоків і застосовується (класичне) перетворення Фейстеля для кожних двох підблоків, а потім виконується циклічний зсув, що відповідає кількості підблоків [35]. ARX виконує шифрування-розшифрування з використанням функцій додавання, обертання та XOR без використання S-box. Реалізація ARX є швидкою та компактною, але має обмеження по захисним властивостям у порівнянні з шифрами SPN та Фейстеля. Нелінійний регістр зсуву зі зворотним зв'язком (NLFSR) може бути застосований як в потоковому, так і в блоковому шифрі. Він використовує будівельні блоки потокових шифрів, поточний стан яких є функцією нелінійного зворотного зв'язку від його попереднього стану [60]. Гібридний шифр поєднує в собі будь-які три типи (SPN, FN, GFN, ARX, NLFSR) для поліпшення специфічних характеристик (наприклад, пропускної здатності, енергії, GE і т.д.),

виходячи з вимог його застосування, або навіть може змішувати блоковий і потоковий шифр.

З цих структур SPN і FN є найбільш популярним вибором завдяки їх гнучкості в реалізації структури на основі вимог застосування [42]. Структури Фейстеля можуть бути реалізовані в апаратних засобах з низькою середньою потужністю, оскільки кругова функція застосовується тільки до однієї половини стану [42]. Структури Фейстеля застосовують нелінійність тільки в одній половині стану в кожному раунді, підтримка запасів міцності зазвичай вимагає більшої кількості функцій в порівнянні зі структурами SPN. Коли є вибір між меншою кількістю раундів функції SPN та більшою кількістю раундів функції Фейстеля з однаковим рівнем безпеки та аналогічними витратами енергії, функція SPN може бути більш розумним вибором [19].

2.5.5 S-box

S-box є важливим компонентом блокових шифрів, які зазвичай використовуються для введення нелінійності. У програмному забезпеченні вони реалізуються за допомогою таблиць пошуку (LUT). Однак, ці таблиці при апаратній реалізації можуть займати велику площу або створювати інші технічні ускладнення. Комбінаторні рішення [45] зазвичай є більш ефективними, де вхідні біти впливають на складність рівнянь, а вихідні біти впливають на кількість необхідних булевих рівнянь. Якщо комбінаторний розв'язок не враховує внутрішню структуру S-box, зайнята площа буде швидко зростати зі збільшенням кількості вхідних та вихідних бітів. Висока нелінійність вимагає більшої площі, але також пропонує більш високу стійкість до диференціального та лінійного аналізу.

Розмір S-box є ще одним важливим параметром[45]. Великі S-box можуть досягти високого рівня безпеки, але вимагають багато ресурсів як для апаратної, так і для програмної реалізації. На апаратному рівні кількість GE зростає експоненціально зі збільшенням розміру. Однак, дуже маленькі S-box не можуть забезпечити прийнятний рівень безпеки. Гарний вибір, що забезпечує баланс між ефективністю і безпекою, і прийнятий для більшості малоресурсний шифрів, є S-

box 4 × 4 (як альтернатива типовим S-box 8 × 8). Якщо потрібен більш високий рівень безпеки, бажано збільшити кількість раундів.

2.6 Вимоги до малоресурсної криптографії

Виходячи із завдань проектування, полегшені алгоритми використовують менші розміри блоків (32, 48 або 64 біт), ніж звичайний шифр, який має більший розмір блоків (64 або 128 біт). Малоресурсна криптографія також використовує менші розміри ключів (менше 96 біт). Найменший розмір ключа, за даними NIST, становить 112 біт[6]. У стандарті ISO/IEC 29192[42] детально описані властивості малоресурсності, що встановлюються на цільових платформах. По-перше, малоресурсність апаратних засобів оцінюється за такими основними показниками, як розмір мікросхеми та енергоспоживання. По-друге, розмір оперативної пам'яті з меншим кодом є переважним для малоресурсних додатків у програмних реалізаціях.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Вибір алгоритму

Хоча блокові шифри мають низьку затримку, вони є найбільш досліджені та модифіковані рішення для безпеки IoT [42,64]. Як зазначено в першому розділі, популяція пристроїв IoT дуже неоднорідна і демонструє надзвичайні відмінності з точки зору обчислювальних можливостей і ресурсів. Деякі пристрої настільки обмежені, що криптографічні операції можуть бути реалізовані тільки в апаратному забезпеченні (наприклад, RFID-мітки), в той час як інші пристрої досить потужні, щоб запускати криптографічне програмне забезпечення з прийнятною швидкістю. Оскільки всі ці пристрої повинні мати можливість безпечно взаємодіяти і спілкуватися один з одним, вони повинні використовувати один і той же шифр. Для того, щоб бути придатним для IoT, малоресурсних блоковий шифр повинен бути ефективним як в апаратному, так і в програмному забезпеченні[49]. Таким чином, має сенс оцінювати програмну ефективність апаратно-орієнтованих шифрів і навпаки.

Вимоги до ресурсів виражаються по-різному в апаратних і програмних реалізаціях. В апаратному випадку вони описуються як площа вентилів, виражена логічними блоками для FPGA або еквівалентами вентилів (GE)[9] для реалізацій ASIC. Однак ці показники сильно залежать від конкретної технології, тому неможливо зробити справедливе і релевантне порівняння реалізацій полегшених алгоритмів точно між різними технологіями. У випадку програмного забезпечення вимоги до ресурсів описуються як кількість регістрів, споживання ОЗП та ПЗП в байтах. Обсяг ПЗП фактично відповідає розміру коду. Апаратні реалізації підходять для пристроїв з високими обмеженнями. Наприклад, в найпростішому випадку, недорогі пасивні RFID-мітки можуть мати в цілому 1000-10 000 воріт[8], з яких тільки 200-2000 виділено для цілей безпеки. Програмні реалізації підходять для менш обмежених пристроїв, і вони оптимізовані для пропускної здатності та енергоспоживання.

Існують різні види блокових шифрів[42,61], такі як Advanced Encryption Standard (AES)[44,62], Data Encryption Standard (DES), Blowfish, Twofish 3DES. Для створення цих блокових шифрів дослідники використовували різні підходи, щоб зробити ці блокові шифри малоресурсними і придатними для IoT.

SPN обробляють відкритий текст за допомогою серії послідовних підстановок і перестановок, які перетворюють дані і готують їх до наступного раунду. До шифрів SPN належать AES [44,62], ICEBERG [65], PRESENT [45], LED [66], PRINCE [48], RECTANGLE [67], PRIDE [68] та інші.

Оскільки метою є сприяння кращому розумінню зв'язку між базовими методологіями проектування малоресурсними шифрів та продуктивністю програмного забезпечення на пристроях IoT з обмеженими ресурсами, були обрані 3 шифрів, які представляють широкий спектр підходів до проектування, заснованих на мережах заміни-перестановки (SPN) та мережах Фейстеля (FN). Класичним прикладом SPN є AES [44,62], але можливі й інші конструкції для S-box та лінійного шару, як це продемонстровано в роботах щодо шифру PRESENT [45]. Загальна структура шифру на основі SPN також може змінюватися, зберігаючи при цьому кругову функцію, що складається з шару S-box і лінійного шару. LED [66] додає ключовий матеріал лише кожні чотири раунди, в той час як PRINCE [48] реалізує властивість, яка називається α-відображення, що мінімізує накладні витрати на дешифрування на додаток до шифрування.

З іншого боку, FN може бути побудована з використанням невеликої SPN як функції Фейстеля, як в LBlock [49] та Piccolo [47], або з використанням простих арифметичних та логічних операцій, як в Simon [61] та ARX проектах, таких як HIGHT [61], RC5 [61] та SPECK [61]. Ці операції можуть бути залежними від даних, як у RC5. Різновидом FN є узагальнений FN, який використовує більше двох гілок.

Спосіб змішування гілок в кінці кожного раунду може складатися з простого обертання (HIGHT) або спеціальної перестановки, що оптимізує дифузію (TWINE [61], Piccolo). Велика кількість гілок дозволяє використовувати дуже прості функції Фейстеля, як у TWINE та HIGHT.

Різні зусилля по криптографічній стандартизації розглядають як програмний, так і апаратний аспект безпеки. Порівняння апаратних та програмних реалізацій декількох малоресурсними блокових шифрів було зроблено в [61]. Програмні та апаратні рішення безпеки мають різні метрики. Програмні метрики включають цикли, пам'ять, цикл на байт, тоді як апаратні метрики включають пропускну здатність, площу, співвідношення по всій площі. Важко отримати пряме порівняння між цими двома показниками. В роботі [61] авторам вдалося провести порівняння за допомогою спеціального малоресурсного реконфігурованого процесора. Вони порівнювали площу, пропускну спроможність та співвідношення площа/пропускна спроможність серед AES, PRESENT, LED, TWINE, як апаратних, так і програмних реалізацій. SPECK, TWINE і PRESENT показали хороші результати як в апаратній, так і в програмній реалізації.

3.1.1 AES

AES[44,62] стандартизований NIST[42] і на сьогоднішній день є найбільш поширеним блоковим шифром. Він має SPN-структуру з внутрішнім станом 128 біт, представленим у вигляді матриці (4 × 4) байт. Над станом шифру працюють функції SubBytes, ShiftRows, MixColumns та AddRoundKey [44]. На сьогоднішній день найкращим одноключовим криптоаналізом AES-128 є атака «зустріч посередині» на семи раундах з десяти [69]. Оптимізовані за розміром реалізації AES поміщають S-box і константи раундів в таблиці пошуку, оскільки вони займають трохи більше 256 байт. Вихідний код нашої оптимізованої за розміром реалізації в основному повторює псевдокод шифру на всіх трьох розглянутих архітектурах. Оскільки T-таблиці є дуже великими (4 кБ як для шифрування, так і для розшифрування), не були включені до реалізації. Останні досягнення в апаратних реалізаціях AES на основі серійного AES S-box [62] використовують 2400GE та 226 циклів на блок. S-box базується на дослідженнях [70], який ретельно дослідив вимоги до апаратного забезпечення для S-box AES. В статті пропонується дуже компактний S-box, який складається з менших полів.

Скануючі перемикачі в основному використані при побудові масиву станів та ключів. Також зменшуються вимоги до площі логіки керування за рахунок

заміни FSM на LFSR. Обробка по рядках є більш апаратно-дружньою, ніж по стовпчиках. Шифр залишається стійким, при цьому бікроклітинний криптоаналіз дає дещо кращі результати, ніж перебір [62].

3.1.2 PRESENT

PRESENT [45] відповідає вимогам малоресурсного шифрування. Це один з перших шифрів, реалізованих на ультра-обмежених пристроях з майже 1000GE (тільки шифрування) [45]. PRESENT є віхою в еволюції малоресурсних блокових шифрів і використовується разом з AES як еталон для нових пропозицій. Як і CLEFIA, PRESENT також стандартизований в ISO/IEC 29192[42]. Він використовує 80- та 128-розрядні ключі з 64-розрядними блоками через 31 раунд. Він має структуру SPN і потребує 1030GE для 80-бітових ключів обох шифрів. Головною особливістю цього алгоритму є заміна звичайних восьми S-box на ретельно підібраний один S-box[45]. PRESENT є першим шифром, в якому застосована серіалізована архітектура, і його конструкція є надзвичайно ефективною з точки зору апаратного забезпечення, оскільки він використовує повністю провідний дифузійний шар без жодного алгебраїчного вузла.

PRESENT має SPN-структуру і поставляється з біт-орієнтованим шаром перестановок. Нелінійний шар базується на одному 4-бітному S-box, який був розроблений для ефективності в апаратному забезпеченні [45]. Усічена диференціальна атака проти 26 з 31 раундів PRESENT описана в [45]. Оскільки S-box досить малий, у всіх наших реалізаціях використовується таблиця пошуку. Однак її поєднання з перестановкою бітів над 64-бітним словом важко оптимізувати без введення надзвичайно великих таблиць пошуку (до 1 МБ для розшифрування). Оптимізована за розміром реалізація нагадує псевдокод шифру і була взята з [71]. Загалом, біт-орієнтований дизайн PRESENT робить реалізацію на мові програмування C дуже повільною, якщо тільки не можна дозволити собі величезні таблиці пошуку. Реалізації на Асемблері використовують переваги інструкцій маніпулювання бітами, які підтримуються цільовими платформами. На AVR реалізація на Асемблері приблизно в 12 разів швидша, ніж її аналог на C, а версія на MSP Асемблері навіть в 19 разів швидша, ніж код на C.

Вивчалися також сучасні програмні реалізації PRESENT на різних носіях [72]. Найновіша програмна пропозиція [73] зменшує розмір коду, зберігає пропускну здатність та виконує як шифрування, так і дешифрування. Розмір коду становить 1000 байт, а алгоритм потребує 11 342/13 599 тактів для шифрування/розшифрування на 8-розрядному мікроконтролері. Реалізація зберігає круглий ключ та стан у регістрах. Дві 256-байтових таблиці використовуються для S-box шифрування та дешифрування, щоб дозволити паралельний пошук, в той час як код для перестановки використовується в обох операціях.

З іншого боку, під час 17 раунду PRESENT було зафіксовано атаки з використанням побічних каналів [45] та атаки з використанням пов'язаного ключа [45]. Вдосконалений диференційний аналіз збоїв був попередньо надісланий в [73], який відновлював ключ шляхом індукування двох або трьох випадкових 2-байтових збоїв. Біклічний криптоаналіз[45] на повному раунді шифру дещо кращий, ніж вичерпний перебір.

3.1.3 CLEFIA

CLEFIA [46] - малоресурсний блоковий шифр, відомий своїми високоефективними апаратними та програмними реалізаціями. Він був розроблений компанією SONY і стандартизований в ISO/IEC 29192[42]. CLEFIA використовує 128-розрядні блоки з 128-, 192- та 256-розрядними ключами через 18, 22 та 26 раундів відповідно. Найбільш компактна реалізація займає 2488GE (тільки для шифрування) для 128-бітного ключа. Вона має серіалізовану архітектуру і не потребує додаткових регістрів. Розшифрування може бути реалізовано з накладними витратами в 116GE. Найбільш малоресурсною версією шифрування/розшифрування є 2604GE, яка на 23% менша за відповідну версію AES-128. Для зменшення кількості мультиплексорів та збільшення кількості циклів автори застосовують методи тактової синхронізації. Крім того, вони взяли на озброєння деякі старі ідеї, що використовуються в компактних реалізаціях AES. Для компактного матричного помножувача операції обчислюються стовпчик за стовпчиком. Неймовірні диференціальні атаки на зменшених раундових перевірках

досягають найкращих результатів криптоаналізу і є дещо кращими, ніж вичерпний перебір [74].

В табл. 2 наведено стислі відомості щодо результатів порівняння деяких параметрів AES, PRESENT, CLEFIA[76].

Таблиця 3.1 –Порівняння характеристик обраних алгоритмів

Малоресурсний алгоритм	Алгоритм проектування шаблону	Розмір вхідного блоку	Розмір ключа	Раунди	Характеристика
AES	SPN	128	128 192 256	10 12 14	Відмінна безпека, гнучкість
PRESENT	SPN	64	80 128	31	Менша кількість воріт, менше пам'яті, використовується для шифрування невеликих обсягів даних
CLEFIA	GFN	128	128 192 256	18 22 26	Демонструє високу продуктивність та стійкість до різних атак

Для програмної реалізації пропонується уніфікована система справедливої оцінки малоресурсних криптографічних систем (FELICS), яка дозволяє оцінити продуктивність малоресурсних блокових або потокових шифрів за розміром реалізації, споживанням оперативної пам'яті та часом, що витрачається на виконання заданої операції [75].

3.2 Система справедливої оцінки малоресурсних критографічних систем FELICS

Для об'єктивної та послідовної оцінки та порівняння програмних реалізацій малоресурсних блокових та потокових шифрів можна використовувати

безкоштовну та відкриту бенчмаркінгову платформу FELICS (Fair Evaluation of Lightweight Cryptographic Systems) [75,77]. На даний час оцінка може бути виконана на трьох широко розповсюджених мікроконтролерах: 8-розрядний AVR[76], 16-розрядний MSP та 32-розрядний ARM, а виділеними метриками є час виконання, споживання оперативної пам'яті та розмір двійкового коду, з яких обчислюється єдине значення "Figure Of Merit" (FOM)[75,77].

FELICS – система бенчмаркінгу з відкритим вихідним кодом, призначену для об'єктивного та послідовного оцінювання програмних реалізацій малоресурсних криптографічних примітивів для вбудованих пристроїв [75,77]. Фреймворк є довільно гнучким завдяки своїй модульній структурі, що дозволяє легко інтегрувати нові метрики, цільові пристрої та сценарії оцінювання.

FELICS має відносно простий користувальницький інтерфейс і призначений для використання розробниками шифрів для порівняння нових примітивів із існуючими. При цьому, отримані метрики є досить детальними та допомагають розробникам у виборі найкращого рішення, такого, що відповідає вимогам конкретного застосування.

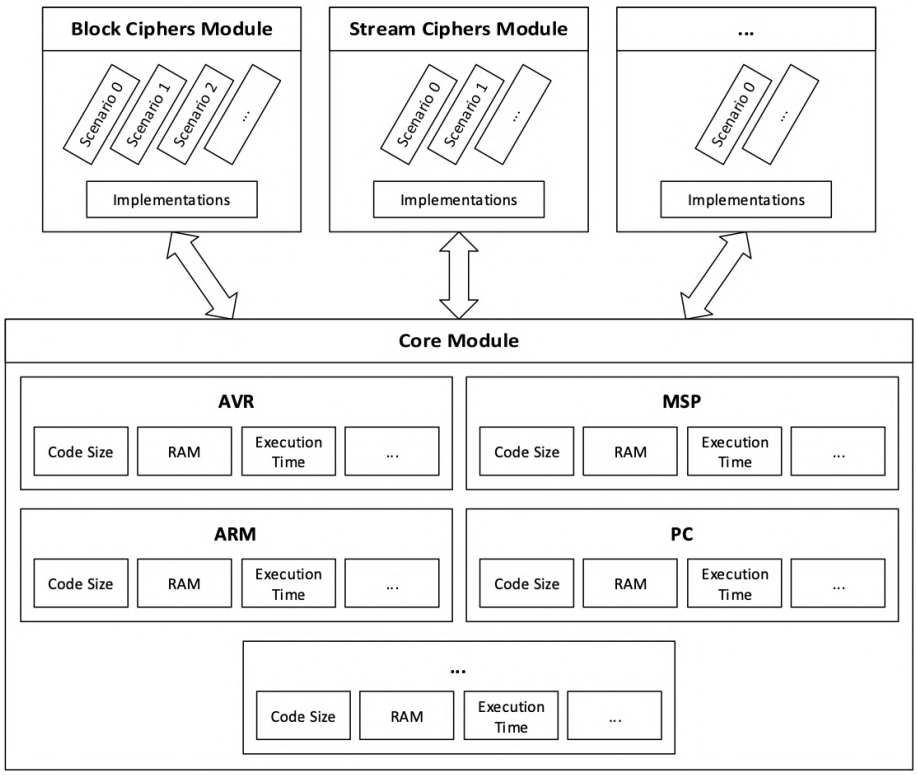


Рисунок 3.1 - Модульна структура FELICS[75]

Слід відмітити, що FELICS має реалізацію PRESENT, однак, вона не є вдалою [45, 78]. Тому, в межах проведеного аналізу, було обрано 32-бітну реалізацію [78]. Ця реалізація була згодом оптимізована за допомогою 3-ох різних методів. В першу чергу, 4-розрядні S-box були замінені на 8-розрядні S-box для покращення продуктивності програмного забезпечення. Потім були розгорнуті всі перимутації через їх надмірну вартість для програмних реалізацій. Крім того, цикли були також повністю розгорнуті, щоб усунути всі залежності і підштовхнути шифр-код до найшвидшого рівня продуктивності. В кінці, звернення до пам'яті було зведено до мінімуму за рахунок утримання стану шифру в регістрах процесора протягом більшої частини часу виконання шифру.

Еталонна реалізація була отримана з сайту CLEFIA та адаптована до фреймворку FELICS[46,74]. Еталонний алгоритм обчислює константи, які використовуються в планувальнику ключів, що призводить до невиправдано високого часу виконання. Тому була розроблена альтернативна версія 32-бітної реалізації, яка має попередньо обчислені значення всіх констант, що зберігаються в таблиці. Решта сім реалізацій, що були розроблені, експлуатують використання T-box. У той час як в цих реалізаціях застосовуються стандартні T-box для 8-бітового орієнтованого еталонного алгоритму та його оптимізованої 32-бітової орієнтованої версії відповідно, всі інші реалізації використовують скорочені T-box. Також, варто відмітити, що потокові шифри широко досліджуються в криптографічному середовищі через більш швидке виконання, але вони є вразливими до атак у порівнянні з блоковими шифрами.

3.2.1 Сценарії використання

Крім оцінки чотирьох основних операцій блокового шифру (тобто шифрування, дешифрування, розклад ключів шифрування та розклад ключів дешифрування), система бенчмаркінгу також підтримує більш просунуті форми оцінки, засновані на сценаріях використання[46,74]. Сценарій використання повинен реалізовувати деяку загальну послугу безпеки, що має практичне значення для IoT, і використовувати основні операції шифрування. Таким чином, можна отримати реалістичні результати бенчмаркінгу, які мають значення в реальному

світі. Результати, представлені в Таблиці 3.1 базуються на сценарії використання, які будуть описані нижче. Інші сценарії використання можуть бути легко додані завдяки модульному дизайну системи бенчмаркінгу.

3.2.1.1 Сценарій 1

Протокол зв'язку. Цей сценарій охоплює необхідність безпечного зв'язку між двома пристроями IoT, такими як два сенсорних вузла в мережі WSN[3]. Передбачається, що конфіденційні дані шифруються та розшифровуються за допомогою малоресурсного блокового шифру в режимі роботи CBC. Оскільки стандартні протоколи зв'язку для IoT, такі як IEEE 802.15.4 та ZigBee [79], характеризуються низькими швидкостями передачі та малими розмірами пакетів, припускається, що в цьому сценарії відкритий текст має довжину 128 байт (тобто 1024 біт). Немає необхідності в схемі заповнення, оскільки довжина відкритого тексту кратна як 64, так і 128 бітам, тобто двом розмірам блоків, які розглядається в цій роботі. Крім того, припускаємо, що головний ключ знаходиться в оперативній пам'яті і що круглі ключі (отримані за допомогою операції розкладу ключів) також зберігаються в оперативній пам'яті для подальшого використання в операціях шифрування або дешифрування. Відкритий текст і вектор ініціалізації для режиму CBC також повинні знаходитися в оперативній пам'яті пристрою на початку виконання операції. З метою зменшення обсягу оперативної пам'яті шифрування виконується на місці, тобто відкритий текст перезаписується шифрованим (і навпаки для розшифрування). При цьому розклад ключів не модифікує головний ключ[74].

3.2.2 Метрики оцінки

Система бенчмаркінгу здатна виділити три основні метрики, а саме: час виконання, споживання пам'яті під час виконання (тобто оперативної пам'яті) та розмір коду[74]. Ці показники є "первинними", оскільки 1) вони значною мірою визначають, наскільки добре блоковий шифр відповідає обмеженням і вимогам IoT, і 2) вони не можуть бути отримані з інших показників. Оскільки досягнення високої продуктивності є основною метою розробки практично будь-якого програмно-орієнтованого шифру, зрозуміло, що система бенчмаркінгу повинна бути здатна

всебічно і точно оцінювати час виконання. Крім того, інші показники, такі як енергоспоживання шифру, мають сильну кореляцію з часом виконання. Хоча більшість існуючих робіт, які представляють результати реалізації малоресурсних блокових шифрів, повідомляють лише про час виконання, вважається важливим також враховувати розмір оперативної пам'яті та розмір коду. Багато пристроїв Інтернету речей настільки обмежені, що мають лише кілька сотень байт оперативної пам'яті та кілька кілобайт флеш-пам'яті, а це означає, що обсяг оперативної пам'яті та розмір коду є вирішальними критеріями для інженерів-криптографів при виборі малоресурсного шифру. Крім того, оскільки неможливо оптимізувати всі три метрики одночасно, доводиться шукати компроміс [46]. Система бенчмаркінгу дозволяє криптографічним інженерам аналізувати такі компроміси і, таким чином, визначати найкращу стратегію оптимізації для вимог цільової програми та обмежень цільового пристрою. Час виконання, обсяг fвизначивши коефіцієнт корисної дії (КДК), який дає змогу ранжувати різні реалізації.

3.2.2.1 Час виконання

Час виконання визначається кількістю тактів, необхідних для виконання кожної з чотирьох основних операцій блокового шифру, а саме: шифрування, дешифрування, розклад ключів шифрування та розклад ключів дешифрування (див. Додаток Б). Як згадувалося раніше, наш фреймворк підтримує симулятори наборів інструкцій, а також отримання реальних вимірювань з плат для розробки. Зокрема, для оцінки часу виконання для 8-розрядного AVR[75] та 16-розрядної платформи MSP430 використовуються симулятори з точністю до циклу Avrora [55,54] та MSPDebug [80], відповідно. З іншого боку, кількість тактів на ARM Cortex-M3 визначається за допомогою системного таймера шляхом зчитування регістру поточного значення SysTick (SYST_CVR)[74]. Цей регістр декрементується з кожним тактовим циклом процесора і дозволяє дуже точно вимірювати кількість пройдених циклів. Фреймворк автоматично вставляє код на мові C для зчитування системного таймера безпосередньо перед операцією, що вимірюється, та одразу після неї, і обчислює час виконання як різницю двох значень таймера[75]. Однак,

отриманий час виконання може відрізнятися на декілька тактів в залежності від того, як компілятор транслює код на мові С для читання таймера в інструкції асемблера в різних контекстах та як вирівняні типи даних в пам'яті. Тому можна отримати дещо різний час виконання однієї і тієї ж операції в різних сценаріях використання. Час виконання для ARM, були зібрані з використанням плати Arduino Due [81] з процесором Cortex-M3 [82].

3.2.2.2 Займання оперативної пам'яті.

Обсяг оперативної пам'яті визначається розміром секції даних (яка містить всі статичні змінні, що ініціалізуються ненульовим значенням) та максимальним споживанням стеку[83]. Обсяг оперативної пам'яті, який займає секція даних, визначається за допомогою інструменту size з колекції GNU Binutils[84]. З іншого боку, максимальне споживання стеку операцією або сценарієм використання оцінюється стандартним способом за допомогою так званої стекової канарки. На початку операції/сценарію (тобто безпосередньо після виклику функції для цієї операції/сценарію) вільний простір стеку заповнюється за певним шаблоном. Потім, в кінці виконання функції, значення в області стеку порівнюються з шаблоном і кількість перезаписаних байт дає витрату стеку[83]. Параметри, що передаються в оцінювану функцію, не потребують підрахунку, оскільки аргументи на всіх трьох платформах розміщуються в регістрах, а не виштовхуються в стек. Однак, до споживання оперативної пам'яті додається розмір специфічних для операції або сценарію змінних (наприклад, байт-масивів для відкритого тексту, ключа, круглого ключа, вектора ініціалізації тощо), які оголошуються та визначаються в головній функції.

3.2.2.3 Розмір коду

Розмір коду вимірюється в байтах і кількісно визначає обсяг пам'яті, який займає операція або сценарій використання в енергонезалежній пам'яті (наприклад, флеш-пам'яті) цільового пристрою. Він оцінюється шляхом застосування інструменту розміру до відповідних об'єктних файлів, згенерованих компілятором. Цей інструмент, який є частиною GNU Binutils[84], перераховує розмір різних розділів об'єктного файлу або виконуваного файлу. Для того, щоб отримати

загальний розмір коду, фреймворк просто додає розмір текстової секції та секції даних відповідних об'єктних файлів. Текстова секція містить виконувані машинні інструкції, які компілятор згенерував з вихідного коду. З іншого боку, секція даних включає в себе всі статичні змінні, які ініціалізуються ненульовим значенням. Вміст цієї секції завантажується з флеш-пам'яті в оперативну пам'ять на початку виконання програми. Як вже зазначалося[46], розділ bss є порожнім, оскільки жодна з тестованих операцій та сценаріїв не використовує неініціалізованих статичних змінних. Слід також зазначити, що загальні фрагменти коду (наприклад, допоміжні функції, що використовуються як при шифруванні, так і при розшифруванні) враховуються лише один раз при підрахунку загального обсягу коду. Таким чином, реалізаторам є сенс виявляти спільну функціональність і виносити її в одну функцію або процедуру на мові C[76]. При цьому не враховуємо розмір головної функції (звідки викликаються функції для основних операцій - шифрування або дешифрування), оскільки він однаковий для всіх шифрів і не є релевантним в контексті досліджуваних сценаріїв.

3.2.3 Аналіз результатів

В таблиці 3 наведено деякі дані про програмну продуктивність деяких малоресурсних блокових шифрів з поточними найкращими результатами FELICS для шифрування 128 байт даних в режимі CBC (сценарій 1 в [76]), відсортовані за показником FoM, де найменший результат є найкращим(Рисунок 3.2).

Табл 3.2 - Поточні найкращі результати FELICS для сценарію 1: Шифрування 128 байт даних в режимі CBC[76].

Шифр	ARM			MSP			AVR			FoM
	Код (B)	RAM (B)	Час (цикли)	Код (B)	RAM (B)	Час (цикли)	Код (B)	RAM (B)	Час (цикли)	
AES	3040	451	73,86	2663	495	86,507	3012	384	58,244	15,8
PRESENT	2116	470	274,463	1818	448	202,05	2160	448	245,232	38,8
CLEFIA	1700	448	233,941	2028	236	386,781	2412	367	288,119	39,9

Рисунок 3.2 – графік порівняння результатів тесту для сценарію 1

Важливо відзначити, що серед усіх блокових шифрувальних алгоритмів, AES (Advanced Encryption Standard)[44,62,69,85] є найбільш добре вивченим алгоритмом. Численні дослідження вже проведені і продовжуються над AES, щоб зробити його малорусурсним та зручним для IT.

Як і очікувалось, стандартизовані реалізації показують доволі повільні результати, за винятком PRESENT [45,71,78], де найповільніший час виконання має реалізація з невеликим обсягом коду. Це свидчить про те, що більшість оптимізацій дозволяє покращити час виконання навіть тоді, коли основна увага приділяється зменшенню розміру коду. PRESENT є «недружнім» малоресурсним шифром при націленості на програмні реалізації і тому, навіть з урахуванням декількох удосконалень та доопрацювань, все ще залишається дуже «важким» для програмного забезпечення (особливо для умов мобільних платформ). Ще одним резуль- татом є те, що для збалансованих шифрів час виконання близький до 1000 тактів майже для кожного шифру, і лише PRESENT дає далекі від цього значення. Він має біт-орієнтовані перестановки, які важко обчислювати в програмному

забезпеченні [45]. В той же час, як AES та CLEFIA підтримують блоки в 128 біт [44,46].

CLEFIA - алгоритм шифрування, котрий має серед досліджуваних алгоритмів найбільшу довжину блоку з довжиною блоку 128 біт [46,74], в той час як в інших алгоритмах перевага віддається довжині блоку 64 біт. Це важливо для пристроїв, що обмінюються даними в мережі Інтернет з об'єктами малої ємності. Ефективніше шифрувати блоки невеликого роз- міру, а також ті, що застосовують архітектуру Фейстеля. З іншого боку, збільшення розміру ключа знижує енергоефективність.

Звичайно, чим більший розмір ключа, тим краще забезпечується безпека. Однак, в умовах роботи IoT, більш вдалими слід вважати ключі від 80 біт до 128 біт (принаймні поки). Вибір структур простим способом, який не потребує занадто багато енергії, підвищує ефективність. Енергоємні структури, такі як процеси редукції та змішані удари, що використовуються в алгоритмах CLEFIA підтверджує цю ситуацію.

PRESENT має певну перевагу над CLEFIA: - нелінійний S-box використовує 4-бітову структуру, що призводить до меншого GE і меншого енергоспоживання. Додаткові властивості S-box допомагають PRESENT досягти бажаного лавинного ефекту, а результати роботи [45] свідчать про те, що PRESENT має компактний S-box. Також в PRESENT є 16 S-box, які розділені на чотири групи. Деякі важливі відмінності цих S-box наведені нижче [45]:

1) Вхідний біт до S-box надходить з 4 чітко визначених S-box тієї ж групи.
2) Вхідні біти до групи з чотирьох S-box надходять з 16 різних S-box.
3) Чотири вихідні біти з певного S-box надходять у чотири чітко визначені S-box, кожен з яких належить до окремої групи S-box у наступному раунді.
4) Вихідні біти S-box у різних групах подаються до різних S-box.

Апаратна реалізація AES для IoT, з точки зору ІБ, може залучати деякі апаратні атаки. Тому важливо спостерігати за цими спробами і своєчасно знаходити потрібні рішення. В ці- лому, AES та CLEFIA є двома найбільш вдалими

прикладами шифрів, які витрачають багато ресурсів (на розмір коду) для досягнення їх більш швидкої роботи.

Блокові шифри мають фіксовану довжину бітів та різні етапи перетворення, які визначаються симетричним ключем. Блокові шифри є дуже універсальними, що дуже корисно з точки зору IoT. Ще однією перевагою є те, що цей процес має майже ідентичні методи шифрування та дешифрування. Отже, його можна реалізувати з меншими ресурсами [1].

3.3 Програмна реалізація PRESENT

Як правило, існує декілька варіантів реалізації алгоритмів, які дозволяють збалансувати продуктивність та площу. В умовах обмеженого простору, оскільки площа, як правило, дуже обмежена, перевага надається компактним рішенням. "Round-based" - це добре відома реалізація, в якій на кожному такті обчислюється цілий раунд шифру. Реалізується тільки один раунд і повторно використовується для інших раундів (у циклічній послідовності): AES (режим ECB), PRESENT і GIFT були реалізовані таким чином, тільки для шифрування[45].

Високорівневий опис алгоритму блокового шифру розміром блоку у 64 біта, та довжиною ключа 80 або 128 біт і кількістю 32 раундів [45] наведено нижче та зображено на рисунку 3.3.

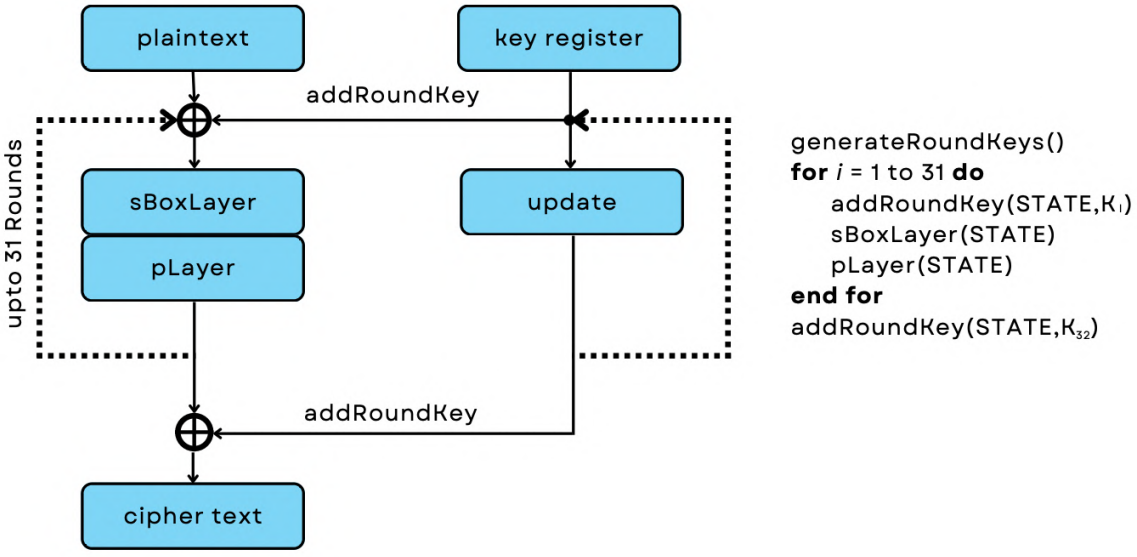


Рисунок 3.3 – Алгоритмічний опис PRESENT[45]

Він займає 1570 вентильних еквівалентів (GE), що підходить для пристроїв з обмеженим ресурсом[45]. Кількість раундів для шифрування або дешифрування даних становить 32, що передбачає підстановку та перестановку відкритого тексту. Мережа перестановок алгоритму показана на рисунку 3.4.

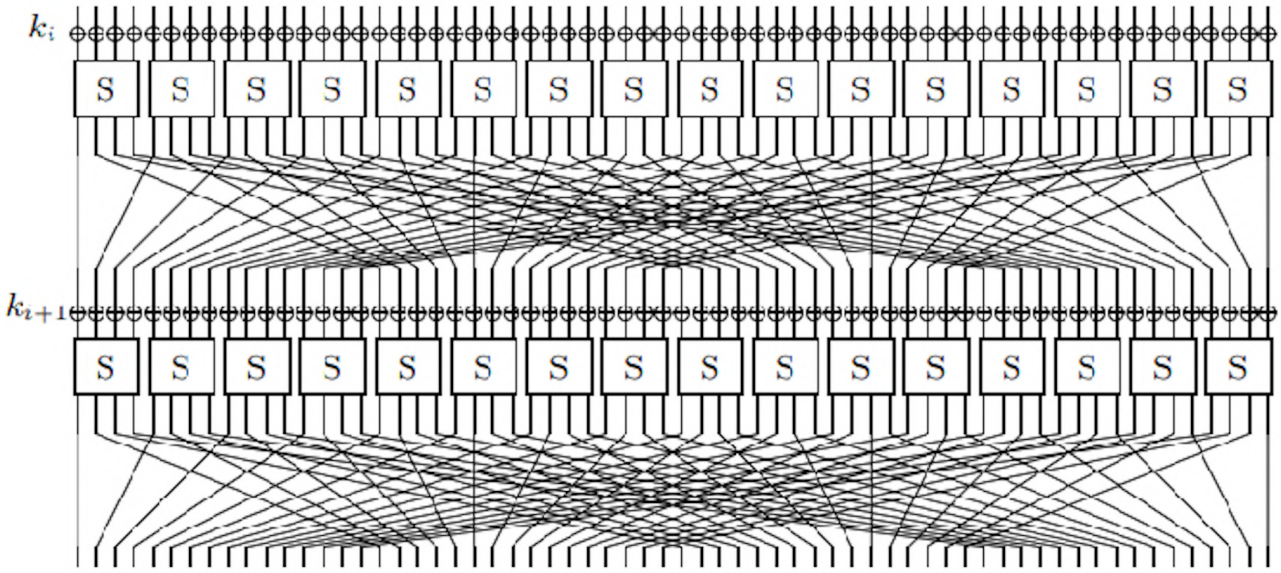


Рисунок 3.4 – S/P Мережа PRESENT[45]

У всіх блокових шифрах генерується комбінація ByteSub, ShiftRow, MixColum та AddRoundKey або їх еквівалентів [45,71,78].

Комбінація цих операцій називається раунд і повинна мати операції, засновані на рівномірних і обернених перетвореннях, які називаються шарами, що були розроблені для того, щоб протистояти лінійному і диференціальному криптоаналізу, якими вони є:

- Нелінійний шар: Полягає в застосуванні паралельно S-образних блоків з оптимальними властивостями нелінійності.
- Лінійний змішувальний шар: Гарантує високий рівень дифузії протягом декількох раундів.
- Ключовий шар додавання: Це унікальна операція АБО між проміжним етапом і унікальним ключем кожного раунду.

64-розрядний відкритий текст спочатку піддається операції АБО з круглим ключем. Потім результат обробляється в шарі підстановки. S-box змінює 4-

бітові дані в інші. Зміст представлених S-box можна знайти в [45]. Нарешті, рівень перестановки переставляє біти з бітової позиції x вхідного відкритого тексту в бітову позицію у вихідного. Всі ці кроки виконуються для кожного з 31 раундів.

Під час шифрування 64 крайні ліві біти ключового регістра використовуються як ключ раунду. Потім, програма Key Schedule оновлює 128-бітний ключовий регістр на кожному раунді. Для PRESENT-80 відрізняється тільки робота розкладу ключів. Нинішній етап складається з 31 регулярного раунду та фінального раунду, який складається лише з етапу змішування ключів. Один звичайний раунд складається з етапу змішування ключів, шару підстановки та шару перестановки[45,86].

3.3.1 Структура алгоритму PRESENT

Реалізований алгоритм на мові програмування. Java версії JDK 17.0.4. Інструментом для реалізації – середовище розробки. Intellij IDEA. Вихідний програмний код представлений в Додатку А. Час виконання алгоритму PRESENT та розробленого алгоритму вимірювався для певного відкритого тексту.

3.3.1.1 Байтовий шар підстановки SboxLayer.

Складається з нелінійного блоку підстановки, що застосовується до кожного відліку матриці етапу незалежно, він називається шаром S-Box[87], як показано в рисунку 3.5 та реалізовано на рисунку 3.6.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Рисунок 3.5 - S-box з алгоритму PRESENT[45]

```
4 usages
public static final int[] SBOX = {
        0x0C, 0x05, 0x06, 0x08, 0x09, 0x00, 0x0A, 0x0D, 0x03, 0x0E, 0x0F, 0x08, 0x04, 0x07, 0x01, 0x02
};
```

Рисунок 3.6 – реалізація S-box з алгоритму PRESENT на мові Java[45]

3.3.1.2 Перестановка бітів pLayer.

Це шар, який змішує шляхом побітової заміни 64-бітовий інформаційний блок, де біт і раунду переміщується в позицію P(i) (формула 3.1) . Порядок такої заміни наведено на рисунку 3.7.

$$P(i) = f(x) = \begin{cases} 16 \times \text{mod}63, & \text{for } 0 \le i \le 62 \\ 63, & \text{for } i = 63 \end{cases} \quad (3.1)$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Рисунок 3.7 - S-box з алгоритму PRESENT[45]

3.3.1.3 Функція розширення ключа addRoundKey.

Операція addRoundKey виконується так само, як і в процедурі шифрування. При заданому круглому ключі $K_i = k^i_{63} \dots k^i_0$ для $1 \le i \le 32i$ поточному STATE $b_i \dots b_0$ addRoundKey складається з операції для $0 \le j \le 63$ (формула 3.2 та реалізація рисунок 3.8),

$$b_j \to b_j \oplus k^i_j \quad (3.2)$$

```
1 usage
public byte[] encrypt(byte[] block) {
    /*Зашифрування 1 блок (8 байт)

    Вхідні дані: блок відкритого тексту у вигляді байтового массиву рядка
    Вихід: блок зашифрованого тексту у вигляді байтового массиву рядка*/
    int state = byte2number(block);

    for (int i = 0; i<NUM_ROUNDS-1; i++) {
        state = addRoundKey(state, roundkeys.get(i));
        state = sBoxLayer(state);
        state = pLayer(state);
    }

    int cipher = addRoundKey(state, roundkeys.get(roundkeys.size()-1));
    return number2string_N(cipher, N: 8);
}
```

Рисунок 3.8 - програмна реалізація зашифрування PRESENT[45]

3.3.1.4 Генерація раундових ключів

Це один з найважливіших блоків, так як відбувається генерація нових ключів для кожного раунду, для даної конкретної реалізації була реалізація тільки для 80-бітових ключів, з цієї причини був згенерований вектор K з 80 позицій, як це видно нижче $K_{79}K_{78} \dots K_0$.[45] У кожному раунді будуть перемішуватися тільки

старші 64 біти нового ключа K_i наступного раунду, спосіб, у який повинна відбуватися ротація, показано нижче в формулі 3.3:

$$K_i = K_{63}K_{62} \dots K_0 = K_{79}K_{78} \dots K_0 \quad (3.3)$$

Потім необхідно виконати наступні операції, в порядку, в якому вони представлені в якому вони представлені:

Обертання бітів ключів вводу(формула 3.4):

$$[K_{79}K_{78} \dots K_0] = [K_{18}K_{17} \dots K_{20}K_{18}] \quad (3.4)$$

Заміна за допомогою S-Box на німб (4 біти) K_{78} на K_{76} ключа(формула 3.5):

$$[K_{79}K_{78}K_{77}K_{76}] = S[K_{79}K_{78}K_{77}K_{76}] \quad (3.5)$$

Додавання або змішування K_{19} до K_{15} ключа з круглим лічильником, через операцію АБО(формула 3.6):

$$[K_{19}K_{18}K_{17}K_{16}K_{15}] = [K_{19}K_{18}K_{17}K_{16}K_{15}] \oplus \text{RoundCounter} \quad (3.6)$$

Таким чином, ключовий регістр повертається на 61 бітову позицію вліво, крайні ліві чотири біти пропускаються через S-box PRESENT, а значення лічильника і виключається з бітами $K_{19}K_{18}K_{17}K_{16}K_{15}$ з K, молодший біт лічильника розташовується праворуч. На рисунку 3.9 графічно зображено розклад клавіш для PRESENT-80 та реалізовано на рисунку 3.10.

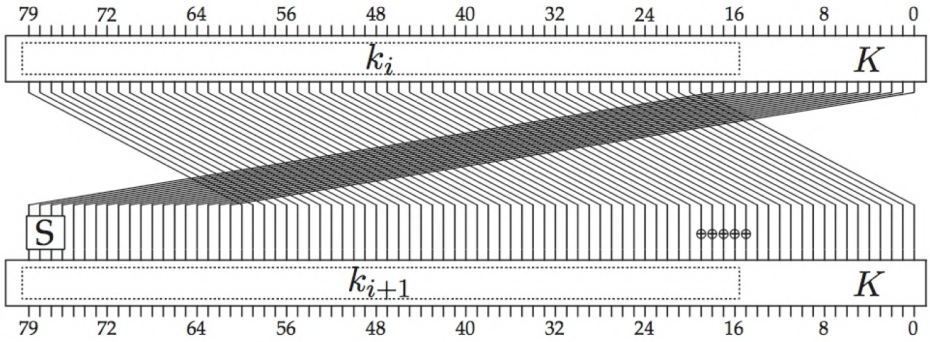
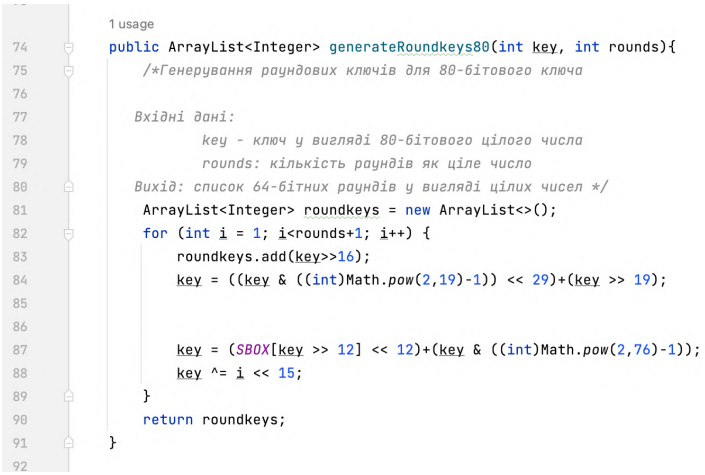


Рисунок 3.9 - Генерації раундових ключів PRESENT-80[45].



```
1 usage
public ArrayList<Integer> generateRoundkeys80(int key, int rounds){
    /*Генерування раундових ключів для 80-бітового ключа

   Вхідні дані:
           key - ключ у вигляді 80-бітового цілого числа
           rounds: кількість раундів як ціле число
   Вихід: список 64-бітних раундів у вигляді цілих чисел */
    ArrayList<Integer> roundkeys = new ArrayList<>();
    for (int i = 1; i<rounds+1; i++) {
        roundkeys.add(key>>16);
        key = ((key & ((int)Math.pow(2,19)-1)) << 29)+(key >> 19);


        key = (SBOX[key >> 12] << 12)+(key & ((int)Math.pow(2,76)-1));
        key ^= i << 15;
    }
    return roundkeys;
}
```

Рисунок 3.10 – Реалізація генерації раундових ключів PRESENT-80[45] на мові програмування Java.

Операція addRoundKey повинна бути виконана в кожному з необхідних раундів для шифрування інформації, також важливо мати на увазі, що в процесі розшифрування беруться блоки зі списку ключів K_i кінцевих значень до тих пір, поки не дійдемо до початкових, тобто останній ключ, використаний для шифрування, буде першим ключем, використаним для розшифрування, тому перед початком процесу розшифрування необхідно виконати всі раунди addRoundKey до тих пір, поки не дійдемо до останнього ключа K_i, зробивши зворотний процес до початкового ключа.

3.4 Сценарій застосувань реалізації PRESENT

PRESENT є прикладом SP-мережі [45] і складається з 31 раунду. Довжина блоку - 64 біти, підтримуються дві довжини ключів - 80 та 128 біт. Враховуючи застосування, рекомендується версія з 80-бітними ключами. Це більш ніж достатній рівень безпеки для додатків з низьким рівнем безпеки, які зазвичай вимагаються при розгортанні на основі тегів, але не менш важливо, що це відповідає цілям проектування апаратно-орієнтованих потокових шифрів в проекті eSTREAM і дозволяє нам зробити більш справедливе порівняння[88].

Апаратна архітектура шифрування PRESENT представлена на рисунку 3.11 Вона використовує 64 біт на вході, 64 біт на виході та 80 біт на ключі, тобто розмір ключа 80 біт. Один 64-розрядний регістр зберігає стан разом з ключем 80-розрядного регістра, який обслуговує паралельний ввід та зсув на декілька біт[89].

Для того, щоб завантажити дані для обробки даних 64 бітового текстового блоку, потрібно 16 тактів. Ця процедура дає вихідні дані(рисунок 3.11).

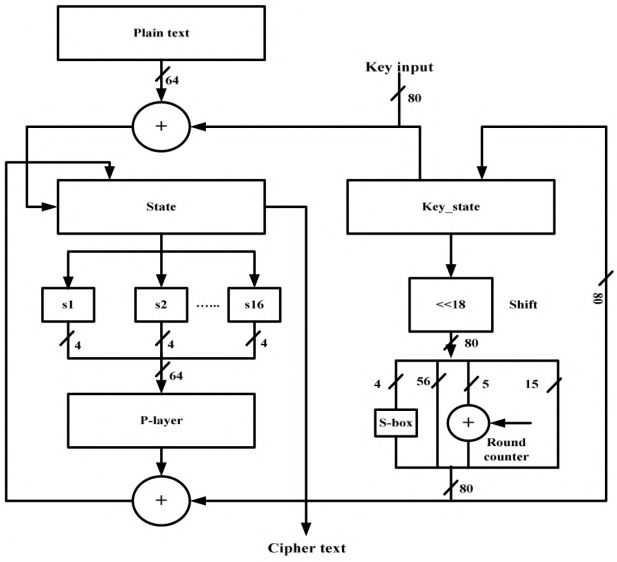


Рисунок 3.11 - Апаратна архітектура шифру Present Encryption 80 bit Key Block[45]

Для різних сценаріїв застосування існують також різні вимоги до реалізації та оптимізації. Реалізація для недорогих пасивних інтелектуальних пристроїв, таких наприклад як RFID-мітки або безконтактні смарт-карти, вимагає невеликої площі і енергоспоживання, в той час як пропускна здатність має другорядне значення. Також варто взяти до уваги, що пристрій для зчитування RFID[88], який зчитує багато пристроїв одночасно, вимагає більш високої пропускної здатності, але площа і енергоспоживання є менш важливими. Активні смарт-пристрої, такі як контактні смарт-карти, не стикаються з жорсткими обмеженнями по потужності, але мають обмеження по часу, а іноді і по енергії. Основними ключовими показниками сучасного блокового шифру є площа, пропускна здатність та енергоспоживання. Пропонується три реалізації сучасного шифру, з яких можна вибрати архітектуру, що найбільше задовольняє заданим вимогам. Перша архітектура - кругова, як описано в [85]. Вона оптимізована за площею, швидкістю та енергоспоживанням. Друга архітектура використовує конвеєрну технологію і забезпечує високу пропускну здатність. Третя архітектура є серіалізованою та мінімізованою з точки зору площі та енергоспоживання. Для того, щоб ще більше

зменшити вимоги до площі, всі архітектури можуть виконувати тільки шифрування. Цього достатньо для шифрування та дешифрування даних при роботі блокового шифру, наприклад, в режимі лічильника. Крім того, це дозволяє більш справедливе порівняння з іншими полегшеними реалізаціями. Наприклад, знакова реалізація Фельдхофера та ін. [89]. Використовучи кругову архітектуру PRESENT-128, можлива реалізація криптографічного співпроцесора з можливостями шифрування та дешифрування [89]. Вибір відповідного інтерфейсу вводу/виводу є дуже специфічним для конкретного застосування, і в той же час може мати значний вплив на показники площі, потужності та часу виконання. Тому для більш чіткої оцінки ефективності криптографічного ядра необхідно реалізовувати спеціальні інтерфейси вводу-виводу, а обрали природну ширину 64-бітового вводу, 64-бітового виводу та 80- або 128-бітового вводу ключа відповідно.

Малоймовірно, що програмі потрібно буде шифрувати великі обсяги даних. Таким чином, реалізація може бути оптимізована для продуктивності або простору без особливого практичного впливу.

У деяких додаткових частинах мається можливість, що ключ буде фіксованим під час виготовлення пристрою. У таких випадках немає необхідності повторно встановлювати ключ на пристрій (що, до речі, виключає цілий ряд атак з маніпулюванням ключами).

Фізичний простір, необхідний для реалізації, буде основним фактором, який слід враховувати після безпеки. За ним слідують пікове і середнє енергоспоживання, а третім важливим показником є часові вимоги.

Блокові шифри, як правило, реалізуються тільки як шифротексти в додатках, що вимагають найбільш ефективного використання простору. Як такий, він може бути використаний в протоколах автентифікації "виклик-відповідь", і при деякому ретельному управлінні станом шаблон лічильника може бути використаний для шифрування і дешифрування повідомлень на пристрій[36].

3.5 Реалізація на IoT платформі

Для можливої реалізації на платформі IoT була підібрана безкоштовна платформа OpenRemote[90].

OpenRemote - це професійний IoT-проект з відкритим вихідним кодом, метою якого є подолання викликів, пов'язаних з інтеграцією багатьох різних протоколів зв'язку та джерел даних в єдине, просте у використанні рішення для управління даними[90].

Її перевагами є те, що вона має протоколи на основі IoT, такі як HTTP, TCP, UDP, Websocket або MQTT, для підключення ваших IoT-пристроїв, шлюзів або служб передачі даних або створення відсутнього API конкретного постачальника. Інші протоколи, такі як KNX або Modbus. Механізм правил з редактором Flow, WHEN-THEN і Groovy UI. Інформаційна панель для забезпечення, автоматизації, контролю та моніторингу вашого додатку, а також компоненти веб-інтерфейсу для створення специфічних для проекту додатків. Мобільний додаток для Android та iOS, включаючи можливість використання геозони та push-повідомлень [90-91]. Рішення Edge Gateway для з'єднання декількох екземплярів з центральним керуючим екземпляром. Можливість створення мультиреалій в поєднанні з управлінням обліковими записами та сервісом ідентифікації(рисунок 3.12).

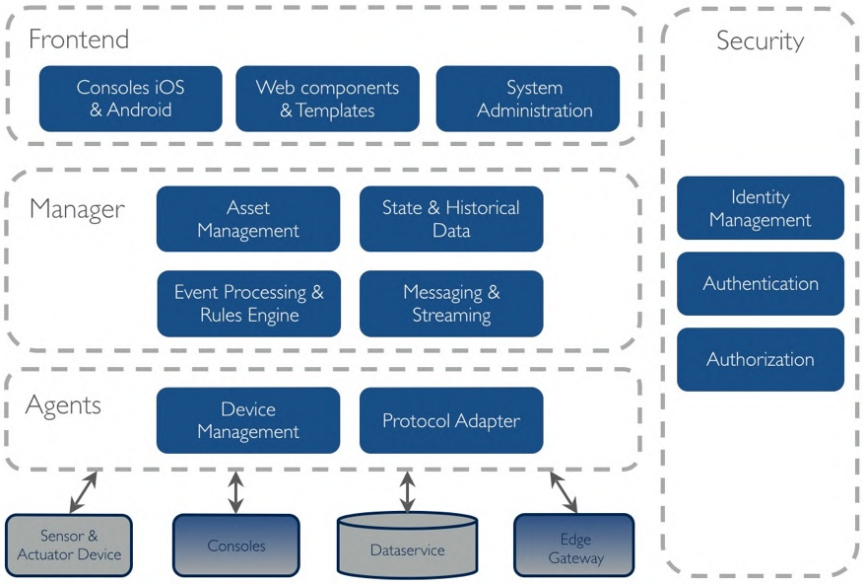


Рис 3.12– загальна структура додатка[90,91]

Для кожної області, створеної в Диспетчері (за допомогою інтерфейсу користувача, коду забезпечення або REST API), автоматично створюється клієнт з назвою openremote і всі ролі, визначені в ClientRole, автоматично додаються до цього клієнта[91].

ВИСНОВКИ

В даній дипломній роботі розглянуто та досліджено реалізацію малоресурсного алгоритму для реалізації в Інтернеті речей для вирішення викликів безпеки для пристроїв із обмеженими ресурсами в децентралізованих системах.

Зростання масштабів використання IoT, обумовлює потребу в більш широкому запровадженні механізмів (алгоритмів) малоресурсного шифрування[1]. Численні обмежені в енергоспоживанні машини і датчики повинні постійно спілкуватися один з одним, надійність яких не повинна порушуватися.

Фокус дослідження був зроблений на область малоресурсної криптографії з симетричним ключем, яка може широко застосовується на пристроях, що мають жорсткі ресурсні обмеженням, для того отримати уявлення про деякі з її проблем.

Для пристроїв із певними ресурсними обмеженнями, наявні стандарти криптографічних алгоритмів можуть бути занадто складними та/або занадто енерговитратними. Крім того, кіберзлочинці можуть скористатися недоліками паролів, які відносно легко підбираються, якщо немає жорстко декларованих вимог до паролів, які створюються користувачами[1].

Для пристроїв з обмеженими ресурсами, зокрема пристроїв IoT, малоресурсна криптографія є ефективним напрямом забезпечення безпеки їх мережевої взаємодії[1]. Вони оцінюються на основі площі мікросхеми, зайнятої в апаратному забезпеченні, або вимог до пам'яті для їх програмної реалізації. Централізовані моделі IoT обмежують корисність сенсорних мереж і створюють значні проблеми з конфіденційністю і безпекою.

Загалом, бажаний малоресурсний алгоритм повинен забезпечувати баланс між вартістю, продуктивністю та безпекою. Зауважимо, що дуже важко оптимізувати всі три цілі одночасно.

Симетричний алгоритм в основному використовується для забезпечення конфіденційності та приватності, а один ключ використовується одночасно для

шифрування дешифрування. Кожен раунд має певні математичні функції для заплутування та розсіювання.

Криптографію з секретним ключем можна класифікувати як блоковими так і потоковими шифри. Потокові шифри шифрують/дешифрують по одному біту за раз з постійно змінюваним ключем і підходять для додатків в реальному часі, таких як аудіо та відео. Блокові шифри шифрують/розшифровують по одному блоку даних за раз, використовуючи один і той же ключ для кожного блоку.

Спираючись на результаті проведеного аналізу, можна констатувати, що AES, PRESENT та CLEFIA, є найбільш експериментально дослідженими та широко адаптованими блоковими шифрами [1]. Однак, з появою нових алгоритмів малоресурсної криптографії, що є об'єктивним процесом, з'являються і нові методики та різновиди атак.

Безумовно, безпечна мережева взаємодія має велике значення в сфері IoT, однак, крім питані ІБ, ефективне застосування IoT, є не менш важливим аспектом. Тому при розробці малоресурсних криптографічних алгоритмів, параметри їх енергоспоживання будуть вкрай актуальні.

Процес обчислень алгоритму AES займає більше кроків і складніший. Малоресурсні блокові шифри мають меншу кількість раундів обчислень, тому їх використовують для зменшення складності. Безпека цих пристроїв аналізується за допомогою атаки силового аналізу.

Визначено, що проектні рішення в малоресурсній криптографії повинні визначатися застосуванням. Ключ до малоресурсної криптографії полягає не тільки в застосуванні нових академічних знань до криптографічних конструкцій, але і в підборі протоколів з криптографічними примітивами, які найкраще підходять для них.

Рекомендації, які вирішують в тому полягають, що сучасні програми, особливо програми реального часу, прагнуть високошвидкісної наскрізної передачі, яка зазвичай суперечить необхідним вимогам конфіденційної та безпечної передачі. Для пристроїв із обмеженнями стандартні криптографічні алгоритми можуть бути занадто великими, занадто повільними або занадто

енерговитратними. Ризик витоку даних залишається значним для пристроїв IoT із жорстко закодованими паролями. Крім того, кіберзлочинці можуть скористатися перевагами паролів, які вгадуються, якщо немає жорстких вимог до паролів, створених користувачами.

В результаті ці методи можна використовувати як в централізованих, так і децентралізованих архітектурах у різноманітних додатках.

Це дослідження також може містити рекомендації та огляд наявних загроз щодо безпеки для будь-якого методу передачі даних. Ці данні можна використовувати для створення надійної, високоточної системи передачі даних, яка відповідатиме потребам у безпеці підприємств та організацій.

Запропонована структура повинна вирішити проблему втрати шаблону даних, а захищаючи дані про зареєстрований пристрій і базу даних вузлів користувача, а також дані каналу зв'язку, коли фрейм підключений до Інтернету, дозволяє користувачам виконувати перевірку ідентичності, покращуючи безпеки, стандартів.

IoT, що розвивається, вимагає малоресурсних конструкцій шифрів, що мають багаті стандарти шифрування, надійну архітектуру, меншу складність, менший час виконання, менше енергоспоживання, низьке використання ресурсів і хорошу стійкість.

Порівнянювались продуктивність існуючих малоресурсних алгоритмів та деякі дослідницькі проблеми в середовищі IoT, на яких будуть зосереджені майбутні дослідження.

Вимоги до дизайну і різні елементи, проблеми і тенденції в області безпеки IoT дають цінну міцну ідею при розробці малоресурсного блокового шифру. Таким чином, для використання IoT з усіма його технологічними перевагами необхідна прийнятна система безпеки, що робить дослідження інноваційними.

З точки зору вимог до пам'яті, алгоритм PRESENT найкраще підходить для малоресурсного криптографічного дизайну. Також обрана IoT-платформа для реалізації та застосування даної криптографії, яка надасть можливість реалізувати та імплементувати обраний алгоритм.

PRESENT був розроблений таким чином, щоб бути особливо придатним для недорогих пристроїв. Точніше, він був розроблений таким чином, щоб мати мінімальну кількість еквівалентів воріт при апаратній реалізації. PRESENT може бути реалізований з приблизною кількістю вентильних еквівалентів. Він також може бути ефективно реалізований на FPGA (field-programmable gate array).

Слід зазначити, що S-box PRESENT реалізує дуже компактну реалізацію, що споживає лише 21 GE для одного 4-бітового S-box. Крім того, з точки зору вимог до обсягів використовуваної пам'яті, цей алгоритм (в порівнянні з іншими) найкращім чином підходить для вирішення питань забезпечення малоресурсного криптографічного захисту даних при здійсненні мережевої взаємодії IoT (принаймні у найближчій перспективі).

Реалізований алгоритм блокового шифру PRESENT на мові програмування Java із генеруванням 80 та 128 бітних ключів.

Проведене дослідження та реалізація підтвердили актуальність алгоритму. Перспективи подальших досліджень можуть полягати у впровадженні на апаратному рівні, що дозволяє виконати більш об’єктивні дослідження.

ПЕРЕЛІК ПОСИЛАНЬ

1. Деменко Є., Нарєжній О. Дослідження застосування алгоритмів малоресурсної криптографії у децентралізованих середовищах. Комп'ютерні науки та кібербезпека. 2022. Т. (1). С. 20–29. URL: https://doi.org/10.26565/2519-2310-2022-1-03 (дата звернення: 20.11.2022).
2. Internet of things / Xia F. та ін. *International journal of communication systems*. 2012. Т. 25, № 9. P. 1101–1102. URL: https://doi.org/10.1002/dac.2417 (дата звернення: 10.11.2022).
3. Andrade R. O., Tello-Oquendo L., Ortiz I. Cybersecurity risks of iot on smart cities. *Cybersecurity risk of iot on smart cities*. Cham, 2021. P. 1–22. URL: https://doi.org/10.1007/978-3-030-88524-3_1 (дата звернення: 10.11.2022).
4. Gartner says a thirty-fold increase in internet-connected physical devices by 2020 will significantly alter how the supply chain operates. *Gartner*. URL: https://www.gartner.com/en/newsroom/press-releases/2014-03-24-gartner-says-a-thirty-fold-increase-in-internet-connected-physical-devices-by-2020-will-significantly-alter-how-the-supply-chain-operates (дата звернення: 10.11.2022).
5. Internet of Things (IoT): a vision, architectural elements, and future directions / Gubbi J. та ін. *Future generation computer systems*. 2013. Т. 29, № 7. P. 1645–1660. URL: https://doi.org/10.1016/j.future.2013.01.010 (дата звернення: 10.11.2022).
6. Report on lightweight cryptography / McKay K. A. та ін. Gaithersburg, MD : National Institute of Standards and Technology, 2017. URL: https://doi.org/10.6028/nist.ir.8114 (дата звернення: 20.11.2022).
7. Correcting the iot history | chetan sharma. *Chetan Sharma*. URL: http://www.chetansharma.com/correcting-the-iot-history/ (дата звернення: 10.11.2022).

8. RFID technology and its applications in Internet of Things (IoT) / Jia X. та ін. *2012 2nd international conference on consumer electronics, communications and networks (cecnet)*, м. Yichang, China, 21–23 квіт. 2012 р. 2012. URL: https://doi.org/10.1109/cecnet.2012.6201508(дата звернення: 12.11.2022).
9. A look into the future: technology being developed by these engineers aims to make the world more sustainable | GE news. *GE | Building a world that works | General Electric*. URL: https://www.ge.com/news/reports/a-look-into-the-future-technology-being-developed-by-these-engineers-aims-to-make-the-world (дата звернення: 12.11.2022).
10. American Airlines takes flight with analytics transformation. *CIO*. URL: https://www.cio.com/article/406357/american-airlines-takes-flight-with-analytics-transformation.html (дата звернення: 17.11.2022).
11. Anonymous authentication for privacy-preserving IoT target-driven applications / Alcaide A. та ін. *Computers & security*. 2013. Т. 37. P. 111–123. URL: https://doi.org/10.1016/j.cose.2013.05.007 (дата звернення: 12.11.2022).
12. Security in the internet of things: a review / Suo H. та ін. *2012 international conference on computer science and electronics engineering (ICCSEE)*, м. Hangzhou, Zhejiang, China, 23–25 берез. 2012 р. 2012. URL: https://doi.org/10.1109/iccsee.2012.373 (дата звернення: 15.11.2022).
13. OWASP internet of things | OWASP foundation. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. URL: https://owasp.org/www-project-internet-of-things/ (дата звернення: 10.11.2022).
14. Cybersecurity: These are the Internet of Things devices that are most targeted by hackers. *ZDNET*. URL: https://www.zdnet.com/article/cybersecurity-these-are-the-internet-of-things-devices-that-are-most-targeted-by-hackers/ (дата звернення: 08.11.2022).
15. Blockchain for the internet of vehicles: a decentralized iot solution for vehicles communication using ethereum / Jabbar R. та ін. Sensors. 2020. Т. 20, № 14. P. 3928. URL: https://doi.org/10.3390/s20143928 (дата звернення: 21.11.2022).

16. IoT and Big Data: Understanding the relationship between these two technologies - Ryax Technologies. Ryax Technologies. URL: https://ryax.tech/iot-and-big-data-understanding-the-relationship-between-these-two-technologies/ (дата звернення: 10.11.2022).
17. Liu D., Ning P., Li R. Establishing pairwise keys in distributed sensor networks. *ACM transactions on information and system security*. 2005. Т. 8, № 1. P. 41–77. URL: https://doi.org/10.1145/1053283.1053287 (дата звернення: 11.11.2022).
18. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions / Singh S. та ін. *Journal of ambient intelligence and humanized computing*. 2017. URL: https://doi.org/10.1007/s12652-017-0494-4 (дата звернення: 12.11.2022).
19. Rishabh, Sharma T. P. Lightweight encryption algorithms, technologies, and architectures in internet of things: a survey. *Innovations in computer science and engineering*. Singapore, 2020. P. 341–351. URL: https://doi.org/10.1007/978-981-15-2043-3_39 (дата звернення: 12.11.2022).
20. Kalman G. 10 common web security vulnerabilities. Toptal Engineering Blog. URL: https://www.toptal.com/security/10-most-common-web-security-vulnerabilities (дата звернення: 12.11.2022).
21. Shea S., Wigmore I. What is IoT Security? - Definition from TechTarget.com. *IoT Agenda*. URL: https://www.techtarget.com/iotagenda/definition/IoT-security-Internet-of-Things-security (дата звернення: 11.11.2022).
22. Millions of iot devices affected by ripple20 vulnerabilities - informazioni sulla sicurezza. *#1 bei Cloud-Sicherheit & Endpunkt-Cybersicherheit | Trend Micro (DE)*. URL: https://www.trendmicro.com/vinfo/it/security/news/internet-of-things/millions-of-iot-devices-affected-by-ripple20-vulnerabilities (дата звернення: 13.11.2022).
23. Kovacs E. Vast majority of OT devices affected by urgent/11 vulnerabilities still unpatched | securityweek.com. *Cybersecurity News, Insights and Analysis |*

SecurityWeek. URL: https://www.securityweek.com/vast-majority-ot-devices-affected-urgent11-vulnerabilities-still-unpatched (дата звернення: 12.11.2022).

24. Urgent/11. *Armis*. URL: https://www.armis.com/research/urgent11/ (дата звернення: 13.11.2022).
25. Woolf N. DDoS attack that disrupted internet was largest of its kind in history, experts say. the Guardian. URL: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet (дата звернення: 14.11.2022).
26. Average number of connected devices in U.S. 2020 | statista. Statista. URL: https://www.statista.com/statistics/1107206/average-number-of-connected-devices-us-house/ (дата звернення: 15.11.2022).
27. Teleworking in the COVID-19 pandemic: trends and prospects. *OECD*. URL: https://www.oecd.org/coronavirus/policy-responses/teleworking-in-the-covid-19-pandemic-trends-and-prospects-72a416b6/ (дата звернення: 16.11.2022).
28. Artificial intelligence and machine learning in 5G and beyond: a survey and perspectives / A. Haidine та ін. *Moving broadband mobile communications forward - intelligent technologies for 5G and beyond*. 2021. URL: https://doi.org/10.5772/intechopen.98517 (дата звернення: 16.11.2022).
29. An introduction to the cyber threat environment - Canadian Centre for Cyber Security. *Canadian Centre for Cyber Security*. URL: https://cyber.gc.ca/en/guidance/introduction-cyber-threat-environment (дата звернення: 16.11.2022).
30. Køien G. M. Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. *River Publishers: Professional Books | Journal | Research Monographs| Handbooks | Edited Volumes*. URL: https://www.riverpublishers.com/journal_read_html_article.php?j=JCSM/4/1/4 (дата звернення: 16.11.2022).
31. Threats from the digital world lead to greater cyber security concerns - Information Security Forum. *Information Security Forum*.

URL: https://www.securityforum.org/in-the-news/threats-from-the-digital-world-lead-to-greater-cyber-security-concerns/ (дата звернення: 09.11.2022).

32. Light weight authentication scheme for smart home iot devices / Kumar V. та ін. *Cryptography*. 2022. Т. 6, № 3. P. 37. URL: https://doi.org/10.3390/cryptography6030037(дата звернення: 09.11.2022).
33. Information security: goals, types and applications - exabeam. *Exabeam*. URL: https://www.exabeam.com/information-security/information-security/ (дата звернення: 12.11.2022).
34. Introduction to microelectronics. Top-Down digital VLSI design. 2015. P. 1–40. URL: https://doi.org/10.1016/b978-0-12-800730-3.00001-0 (дата звернення: 12.11.2022).
35. A comparative survey of symmetric and asymmetric key cryptography / Chandra S. та ін. *2014 international conference on electronics,communication and computational engineering (ICECCE)*, м. Hosur, Tamilnadu, India, 17–18 листоп. 2014 р. 2014. URL: https://doi.org/10.1109/icecce.2014.7086640 (дата звернення: 11.11.2022).
36. An intrusion detection system for connected vehicles in smart cities / Aloqaily M. та ін. *Ad hoc networks*. 2019. Т. 90. P. 101842. URL: https://doi.org/10.1016/j.adhoc.2019.02.001 (дата звернення: 10.11.2022).
37. Application Domain-Based Overview of IoT Network Traffic Characteristics / Pekar A. та ін. *ACM Computing Surveys*. 2020. Т. 53, № 4. P. 1–33. URL: https://doi.org/10.1145/3399669 (дата звернення: 10.11.2022).
38. A systematic survey of Internet of Things frameworks for smart city applications / Abadía P. та ін. *Sustainable Cities and Society*. 2022. P. 103949. URL: https://doi.org/10.1016/j.scs.2022.103949 (дата звернення: 11.11.2022).
39. Decentralised iot architecture for efficient resources utilisation / Mocnej J. та ін. *IFAC-PapersOnLine*. 2018. Т. 51, № 6. P. 168–173. URL: https://doi.org/10.1016/j.ifacol.2018.07.148 (дата звернення: 12.11.2022).
40. Blockchains | web3 identity. Blockchains. URL: https://www.blockchains.com (дата звернення: 12.11.2022).

41. A blockchain based distributed vehicular network architecture for smart cities / Rehman M. та ін. Advances in intelligent systems and computing. Cham, 2020. P. 320–331. URL: https://doi.org/10.1007/978-3-030-44038-1_29 (дата звернення: 12.11.2022).
42. ISO/IEC 29192-2:2012. Information technology – security techniques – lightweight cryptography – part 2: block ciphers. Чинний від 2012-11-01. Вид. офіц. URL: https://www.iso.org/obp/ui#iso:std:iso-iec:29192:-2:ed-2:v1:en (дата звернення: 19.11.2022).
43. ISO/IEC JTC 1/SC 27. Information security, cybersecurity and privacy protection. Вид. офіц. URL: https://www.din.de/en/meta/jtc1sc27?level=tpl-home&contextid=jtc1sc27&languageid=en (дата звернення: 20.11.2022).
44. FIPS 197. Advanced encryption standard (AES). Чинний від 2001-11-26. Вид. офіц. URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (дата звернення: 18.11.2022).
45. PRESENT: an ultra-lightweight block cipher / Bogdanov A. та ін. *Cryptographic hardware and embedded systems - CHES 2007*. Berlin, Heidelberg. P. 450–466. URL: https://doi.org/10.1007/978-3-540-74735-2_31 (дата звернення: 18.11.2022).
46. The 128-bit blockcipher CLEFIA (extended abstract) / Shirai T. та ін. *Fast software encryption*. Berlin, Heidelberg, 2007. P. 181–195. URL: https://doi.org/10.1007/978-3-540-74619-5_12 (дата звернення: 18.11.2022).
47. Piccolo: An Ultra-Lightweight Blockcipher / Shibutani K. та ін. *Cryptographic hardware and embedded systems – CHES 2011*. Berlin, Heidelberg, 2011. P. 342–357. URL: https://doi.org/10.1007/978-3-642-23951-9_23 (дата звернення: 12.11.2022).
48. PRINCE – A low-latency block cipher for pervasive computing applications / Borghoff J. та ін. *Advances in cryptology – ASIACRYPT 2012*. Berlin, Heidelberg,

2012. P. 208–225. URL: https://doi.org/10.1007/978-3-642-34961-4_14 (дата звернення: 10.11.2022).

49. Wu W., Zhang L. LBlock: a lightweight block cipher. *Applied cryptography and network security*. Berlin, Heidelberg, 2011. P. 327–344. URL: https://doi.org/10.1007/978-3-642-21554-4_19 (дата звернення: 10.11.2022).
50. Lightweight cryptography algorithms for internet of things enabled networks: an overview / Shamala L. M. та ін. *Journal of physics: conference series*. 2021. Т. 1717. P. 012072. URL: https://doi.org/10.1088/1742-6596/1717/1/012072 (дата звернення: 11.11.2022).
51. Das R. (writer/revisions editor). A review of asymmetric cryptography | infosec resources. *Infosec Resources*. URL: https://resources.infosecinstitute.com/topic/review-asymmetric-cryptography/ (дата звернення: 12.11.2022).
52. ISO/IEC 29192-4:2013. Information technology – Security techniques – Lightweight cryptography – Part 4: mechanisms using asymmetric technique. Чинний від 2013-06-01. Вид. офіц. URL: https://www.iso.org/standard/56427.html. (дата звернення: 12.11.2022).
53. An overview of cryptography. GaryKessler.net Home Page. URL: https://www.garykessler.net/library/crypto.html (дата звернення: 12.11.2022).
54. Symmetric and asymmetric key cryptography: a detailed guide in 2022. Jigsaw Academy. URL: https://www.jigsawacademy.com/blogs/cyber-security/symmetric-and-asymmetric-key-cryptography (дата звернення: 10.11.2022).
55. Tiwari H., Asawa K. A secure and efficient cryptographic hash function based on NewFORK-256. Egyptian informatics journal. 2012. Т. 13, № 3. P. 199–208. URL: https://doi.org/10.1016/j.eij.2012.08.003 (дата звернення: 21.11.2022).
56. Partala J. Indifferentiable hash functions in the standard model. IET information security. 2021. Т. 15, № 4. P. 309–316. URL: https://doi.org/10.1049/ise2.12025 (дата звернення: 10.11.2022).

57. Nurdiyanto H., Rahim R., Wulan N. Symmetric stream cipher using triple transposition key method and base64 algorithm for security improvement. Journal of physics: conference series. 2017. Т. 930. P. 012005. URL: https://doi.org/10.1088/1742-6596/930/1/012005 (дата звернення: 09.11.2022).
58. Babbage S., Dodd M. The stream cipher MICKEY 2.0. *ECRYPT*. URL: https://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf (дата звернення: 10.11.2022).
59. De Cannière C., Preneel B. Trivium. *Lecture notes in computer science*. Berlin, Heidelberg, 2008. P. 244–266. URL: https://doi.org/10.1007/978-3-540-68351-3_18 (дата звернення: 14.11.2022).
60. Hell M. Grain-128AEAD - A lightweight AEAD stream cipher. *NIST Computer Security Resource Center | CSRC*. URL: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/grain-128aead-spec-round2.pdf (дата звернення: 14.11.2022).
61. A review of lightweight block ciphers / Hatzivasilis G. та ін. *Journal of cryptographic engineering*. 2017. Т. 8, № 2. P. 141–184. URL: https://doi.org/10.1007/s13389-017-0160-y (дата звернення: 14.11.2022).
62. Pushing the limits: A very compact and a threshold implementation of AES / Moradi A. та ін. *Advances in cryptology – EUROCRYPT 2011*. Berlin, Heidelberg, 2011. P. 69–88. URL: https://doi.org/10.1007/978-3-642-20465-4_6 (дата звернення: 18.11.2022).
63. Compact implementation and performance evaluation of block ciphers in attiny devices / Eisenbarth T. та ін. *Progress in cryptology - AFRICACRYPT 2012*. Berlin, Heidelberg, 2012. P. 172–187. URL: https://doi.org/10.1007/978-3-642-31410-0_11 (дата звернення: 14.11.2022).
64. Comparison of hardware and software implementations of selected lightweight block ciphers / Diehl W. та ін. *2017 27th international conference on field programmable logic and applications (FPL)*, м. Ghent, Belgium, 4–8 верес. 2017 р.

2017. URL: https://doi.org/10.23919/fpl.2017.8056808 (дата звернення: 14.11.2022).

65. ICEBERG : an involutional cipher efficient for block encryption in reconfigurable hardware / Standaert F.-X. та ін. *Fast software encryption*. Berlin, Heidelberg, 2004. P. 279–298. URL: https://doi.org/10.1007/978-3-540-25937-4_18 (дата звернення: 14.11.2022).
66. The LED block cipher / Guo J. та ін. Cryptographic hardware and embedded systems – CHES 2011. Berlin, Heidelberg, 2011. P. 326–341. URL: https://doi.org/10.1007/978-3-642-23951-9_22 (дата звернення: 14.11.2022).
67. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms / Zhang W. та ін. Science china information sciences. 2015. Т. 58, № 12. P. 1–15. URL: https://doi.org/10.1007/s11432-015-5459-7 (дата звернення: 14.11.2022).
68. Block ciphers – focus on the linear layer (feat. PRIDE) / Albrecht M. R. та ін. Advances in cryptology – CRYPTO 2014. Berlin, Heidelberg, 2014. P. 57–76. URL: https://doi.org/10.1007/978-3-662-44371-2_4 (дата звернення: 20.11.2022).
69. Dunkelman O., Keller N., Shamir A. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. Journal of cryptology. 2013. Т. 28, № 3. P. 397–422. URL: https://doi.org/10.1007/s00145-013-9159-4 (дата звернення: 20.11.2022).
70. Construction of s-box based on chaotic map and algebraic structures / Hussain I. та ін. Symmetry. 2019. Т. 11, № 3. P. 351. URL: https://doi.org/10.3390/sym11030351 (дата звернення: 20.11.2022).
71. GIFT: a small present / Banik S. та ін. Lecture notes in computer science. Cham, 2017. P. 321–345. URL: https://doi.org/10.1007/978-3-319-66787-4_16 (дата звернення: 20.11.2022).
72. Imdad M., Ramli S. N., Mahdin H. An enhanced key schedule algorithm of PRESENT-128 block cipher for random and non-random secret keys. Symmetry. 2022. Т. 14, № 3. P. 604. URL: https://doi.org/10.3390/sym14030604 (дата звернення: 20.11.2022).

73. Rashidi B. Flexible structures of lightweight block ciphers PRESENT, SIMON and LED. IET circuits, devices & systems. 2020. Т. 14, № 3. P. 369–380. URL: https://doi.org/10.1049/iet-cds.2019.0363 (дата звернення: 20.11.2022).
74. Akishita T., Hiwatari H. Very compact hardware implementations of the blockcipher CLEFIA. *Selected areas in cryptography*. Berlin, Heidelberg, 2012. P. 278–292. URL: https://doi.org/10.1007/978-3-642-28496-0_17 (дата звернення: 18.11.2022).
75. Efficient implementation of eSTREAM ciphers on 8-bit AVR microcontrollers / Meiser G. та ін. *2008 international symposium on industrial embedded systems (SIES)*, м. Le Grande Motte, France, 11–13 черв. 2008 р. 2008. URL: https://doi.org/10.1109/sies.2008.4577681 (дата звернення: 18.11.2022).
76. FELICS -fair evaluation of lightweight cryptographic systems / Dinu D. та ін. *NIST Computer Security Resource Center | CSRC*. URL: https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session7-dinu-paper.pdf (дата звернення: 18.11.2022).
77. CryptoLUX > FELICS. *CryptoLUX*. URL: https://www.cryptolux.org/index.php/FELICS(дата звернення: 20.11.2022).
78. Jawad Kubba Z. M., Hoomod H. K. Modified PRESENT Encryption algorithm based on new 5D Chaotic system. IOP conference series: materials science and engineering. 2020. Т. 928. P. 032023. URL: https://doi.org/10.1088/1757-899x/928/3/032023 (дата звернення: 20.11.2022).
79. Yen L.-H., Tsai W.-T. The room shortage problem of tree-based ZigBee/IEEE 802.15.4 wireless networks. *Computer communications*. 2010. Т. 33, № 4. P. 454–462. URL: https://doi.org/10.1016/j.comcom.2009.10.013 (дата звернення: 20.11.2022).
80. MSPDebug. Daniel Beer. URL: https://dlbeer.co.nz/mspdebug/ (дата звернення: 21.11.2022).
81. Arduino due: development - technet articles - united states (english) - technet wiki. Become a member to get the most out of TechNet.

URL: https://social.technet.microsoft.com/wiki/contents/articles/31354.arduino-due-development.aspx (дата звернення: 21.11.2022).

82. Cortex-M3. Arm Developer. URL: https://developer.arm.com/Processors/Cortex-M3 (дата звернення: 21.11.2022).
83. Energy evaluation of software implementations of block ciphers under memory constraints / J. Grossschadl та ін. *Design, automation & test in europe conference*, м. Nice, France, 16–20 квіт. 2007 р. 2007. URL: https://doi.org/10.1109/date.2007.364443 (дата звернення: 20.11.2022).
84. Binutils- GNU project - free software foundation. The GNU Operating System and the Free Software Movement. URL: https://www.gnu.org/software/binutils/ (дата звернення: 10.11.2022).
85. Pushing the limits: A very compact and a threshold implementation of AES / Moradi A. та ін. *Advances in cryptology – EUROCRYPT 2011*. Berlin, Heidelberg, 2011. P. 69–88. URL: https://doi.org/10.1007/978-3-642-20465-4_6 (дата звернення: 11.11.2022).
86. Comparison of hardware and software implementations of selected lightweight block ciphers / Diehl W. та ін. *2017 27th international conference on field programmable logic and applications (FPL)*, м. Ghent, Belgium, 4–8 верес. 2017 р. 2017. URL: https://doi.org/10.23919/fpl.2017.8056808 (дата звернення: 12.11.2022).
87. Compact FPGA implementation of PRESENT with Boolean S-Box / Tay J. J. et al. 2015 6th asia symposium on quality electronic design (ASQED), Kula Lumpur, Malaysia, 4–5 August 2015. 2015. URL: https://doi.org/10.1109/acqed.2015.7274024 (дата звернення: 12.11.2022).
88. Midori: a block cipher for low energy / Banik S. та ін. *Advances in cryptology – ASIACRYPT 2015*. Berlin, Heidelberg, 2015. P. 411–436. URL: https://doi.org/10.1007/978-3-662-48800-3_17 (дата звернення: 18.11.2022).
89. Midori: a block cipher for low energy / Banik S. та ін. *Advances in cryptology – ASIACRYPT* 2015. Berlin, Heidelberg, 2015. P. 411–436.

URL: https://doi.org/10.1007/978-3-662-48800-3_17 (дата звернення: 12.11.2022).

90. OpenRemote | the 100% open source iot platform. OpenRemote. URL: https://www.openremote.io (дата звернення: 12.11.2022).
91. GitHub - openremote/openremote: 100% open-source IoT Platform - Integrate your devices, create rules, and analyse and visualise your data. GitHub. URL: https://github.com/openremote/openremote (дата звернення: 20.11.2022).

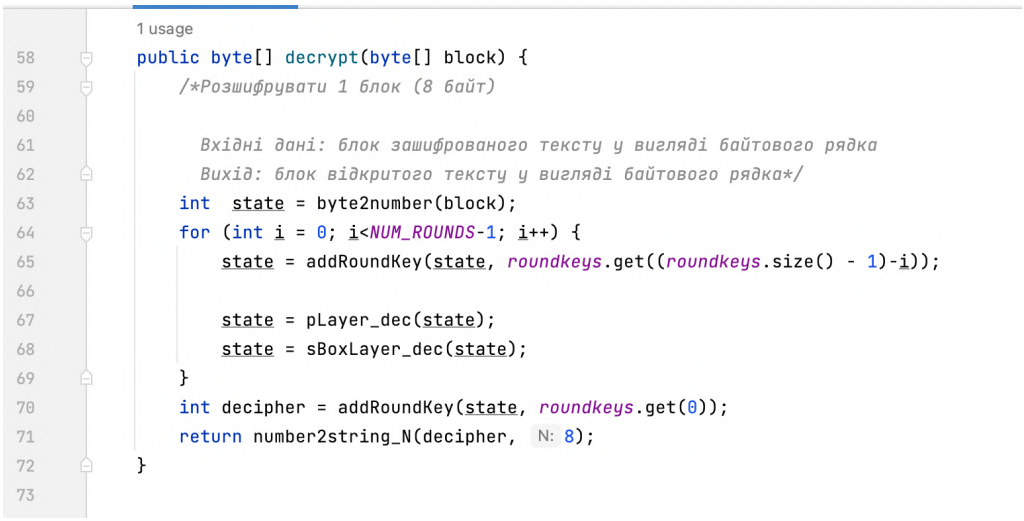
ДОДАТОК А

```java
import java.nio.ByteBuffer;
import java.util.ArrayList;

2 usages
public class PRESENT {

    4 usages
    public static final int NUM_ROUNDS = 32;
    1 usage
    public static final int[] INVERSE_PERMUTATION = {
            0, 16, 32, 48,  1, 17, 33, 49,  2, 18, 34, 50,  3, 19, 35, 51,
            4, 20, 36, 52,  5, 21, 37, 53,  6, 22, 38, 54,  7, 23, 39, 55,
            8, 24, 40, 56,  9, 25, 41, 57, 10, 26, 42, 58, 11, 27, 43, 59,
            12, 28, 44, 60, 13, 29, 45, 61, 14, 30, 46, 62, 15, 31, 47, 63
    };
    1 usage
    public static final int[] INVERSE_SBOX = {
            0x05, 0x0E, 0x0F, 0x08, 0x0C, 0x01, 0x02, 0x0D, 0x0B, 0x04, 0x06, 0x03, 0x00, 0x07, 0x09, 0x0A
    };
    1 usage
    public static final int[] PERMUTATION = {
            0x00,0x04,0x08,0x0c,0x10,0x14,0x18,0x1c,0x20,0x24,0x28,0x2c,0x30,0x34,0x38,0x3c,
            0x01,0x05,0x09,0x0d,0x11,0x15,0x19,0x1d,0x21,0x25,0x29,0x2d,0x31,0x35,0x39,0x3d,
            0x02,0x06,0x0a,0x0e,0x12,0x16,0x1a,0x1e,0x22,0x26,0x2a,0x2e,0x32,0x36,0x3a,0x3e,
            0x03,0x07,0x0b,0x0f,0x13,0x17,0x1b,0x1f,0x23,0x27,0x2b,0x2f,0x33,0x37,0x3b,0x3
    };
    4 usages
    public static final int[] SBOX = {
            0x0C, 0x05, 0x06, 0x0B, 0x09, 0x00, 0x0A, 0x0D, 0x03, 0x0E, 0x0F, 0x08, 0x04, 0x07, 0x01, 0x02
    };
    8 usages
    public static ArrayList<Integer> roundkeys;

```

Рисунок А.1 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.

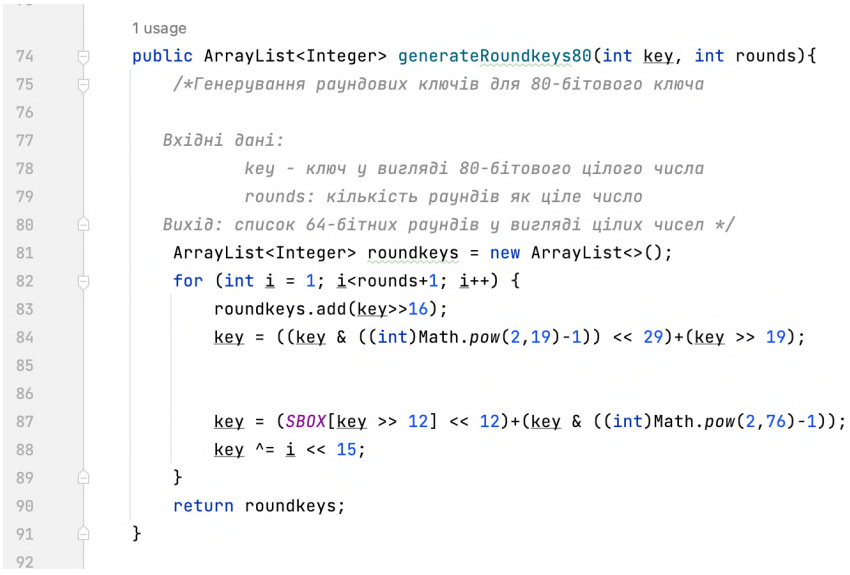
```java
    1 usage
    public PRESENT(byte[] key) {
        /*Створили конструктор для створення об'єкт шифру PRESENT
                ключ: ключ у вигляді 128-бітного або 80-бітного сирого рядка
                rounds: кількість раундів у вигляді цілого числа, за замовчуванням 32
        */
        if (key.length*8 == 80) {
            roundkeys = generateRoundkeys80(byte2number(key), NUM_ROUNDS);
        } else if (key.length * 8 == 128) {
            roundkeys = generateRoundkeys128(byte2number(key), NUM_ROUNDS);
        } else {
            throw new IllegalArgumentException("Key must be a 128-bit or 80-bit rawstring");
        }
    }

    1 usage
    public byte[] encrypt(byte[] block) {
        /*Зашифрування 1 блок (8 байт)

        Вхідні дані: блок відкритого тексту у вигляді байтового массиву рядка
        Вихід: блок зашифрованого тексту у вигляді байтового массиву рядка*/
        int state = byte2number(block);

        for (int i = 0; i<NUM_ROUNDS-1; i++) {
            state = addRoundKey(state, roundkeys.get(i));
            state = sBoxLayer(state);
            state = pLayer(state);
        }

        int cipher = addRoundKey(state, roundkeys.get(roundkeys.size()-1));
        return number2string_N(cipher,  N: 8);
    }
```

Рисунок А.2 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.



```
1 usage
public byte[] decrypt(byte[] block) {
    /*Розшифрувати 1 блок (8 байт)

      Вхідні дані: блок зашифрованого тексту у вигляді байтового рядка
      Вихід: блок відкритого тексту у вигляді байтового рядка*/
    int  state = byte2number(block);
    for (int i = 0; i<NUM_ROUNDS-1; i++) {
        state = addRoundKey(state, roundkeys.get((roundkeys.size() - 1)-i));

        state = pLayer_dec(state);
        state = sBoxLayer_dec(state);
    }
    int decipher = addRoundKey(state, roundkeys.get(0));
    return number2string_N(decipher,  N: 8);
}
```

Рисунок А.3 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.

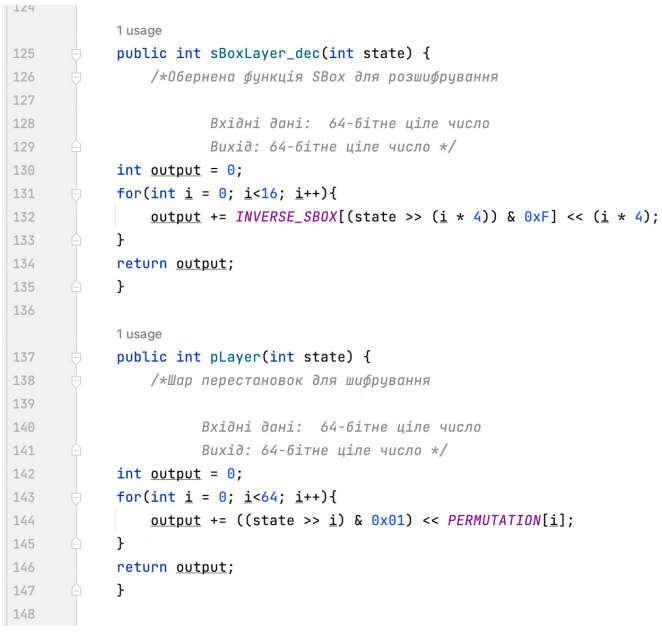


```
1 usage
public ArrayList<Integer> generateRoundkeys80(int key, int rounds){
    /*Генерування раундових ключів для 80-бітового ключа

    Вхідні дані:
            key - ключ у вигляді 80-бітового цілого числа
            rounds: кількість раундів як ціле число
    Вихід: список 64-бітних раундів у вигляді цілих чисел */
     ArrayList<Integer> roundkeys = new ArrayList<>();
     for (int i = 1; i<rounds+1; i++) {
         roundkeys.add(key>>16);
         key = ((key & ((int)Math.pow(2,19)-1)) << 29)+(key >> 19);


         key = (SBOX[key >> 12] << 12)+(key & ((int)Math.pow(2,76)-1));
         key ^= i << 15;
     }
     return roundkeys;
}
```

Рисунок А.4 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.

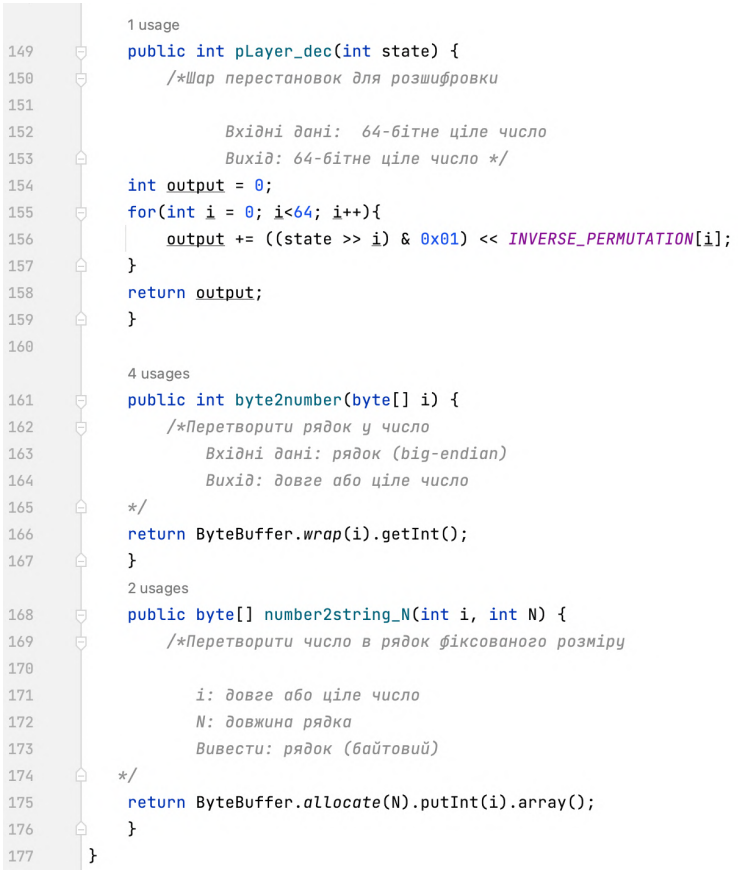
```
1 usage
public ArrayList<Integer> generateRoundkeys128(int key, int rounds){
    /*Згенерувати круглі ключі для 128-бітового ключа

    Вхідні дані:
            key - ключ у вигляді 128-бітового цілого числа
            rounds - кількість раундів цілим числом
    Вихід: список 64-бітних раундів у вигляді цілих чисел */
ArrayList<Integer> roundkeys = new ArrayList<>();
for (int i = 1; i<rounds+1; i++){
     roundkeys.add(key >> 64);
     key = ((key & ((int)Math.pow(2,67)-1)) << 61)+(key>>67);
     key = (SBOX[key >> 124] << 124) + (SBOX[(key >> 120) & 0xF] << 120) + (key & ((int)Math.pow(2,120)-1));
     key ^= i << 62;
}
return roundkeys;
}
4 usages
public int addRoundKey(int state,  Integer roundkey) {
return  state^roundkey;
}

1 usage
public int sBoxLayer(int state) {
    /*Функція SBox для шифрування

   Вхідні дані:  64-бітне ціле число
   Вихід: 64-бітне ціле число */
int output = 0;
for(int i = 0; i<16; i++){
    output += SBOX[(state >> (i * 4)) & 0xF] << (i * 4);
}
return output;
}
```

Рисунок А.5 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.



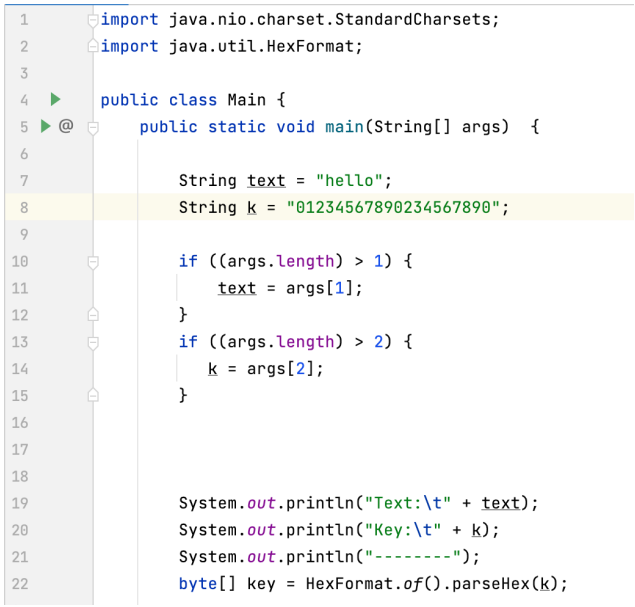
```
1 usage
public int sBoxLayer_dec(int state) {
    /*Обернена функція SBox для розшифрування

            Вхідні дані:  64-бітне ціле число
            Вихід: 64-бітне ціле число */
int output = 0;
for(int i = 0; i<16; i++){
    output += INVERSE_SBOX[(state >> (i * 4)) & 0xF] << (i * 4);
}
return output;
}

1 usage
public int pLayer(int state) {
    /*Шар перестановок для шифрування

          Вхідні дані:  64-бітне ціле число
          Вихід: 64-бітне ціле число */
int output = 0;
for(int i = 0; i<64; i++){
    output += ((state >> i) & 0x01) << PERMUTATION[i];
}
return output;
}
```

Рисунок А.6 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.



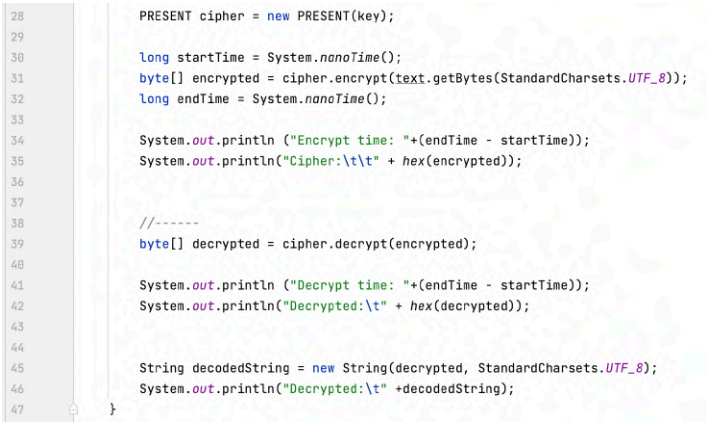
```
    1 usage
    public int pLayer_dec(int state) {
        /*Шар перестановок для розшифровки

                Вхідні дані:  64-бітне ціле число
                Вихід: 64-бітне ціле число */
    int output = 0;
    for(int i = 0; i<64; i++){
        output += ((state >> i) & 0x01) << INVERSE_PERMUTATION[i];
    }
    return output;
    }

    4 usages
    public int byte2number(byte[] i) {
        /*Перетворити рядок у число
            Вхідні дані: рядок (big-endian)
            Вихід: довге або ціле число
    */
    return ByteBuffer.wrap(i).getInt();
    }
    2 usages
    public byte[] number2string_N(int i, int N) {
        /*Перетворити число в рядок фіксованого розміру

            i: довге або ціле число
            N: довжина рядка
            Вивести: рядок (байтовий)
    */
    return ByteBuffer.allocate(N).putInt(i).array();
    }
}
```

Рисунок А.7 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.



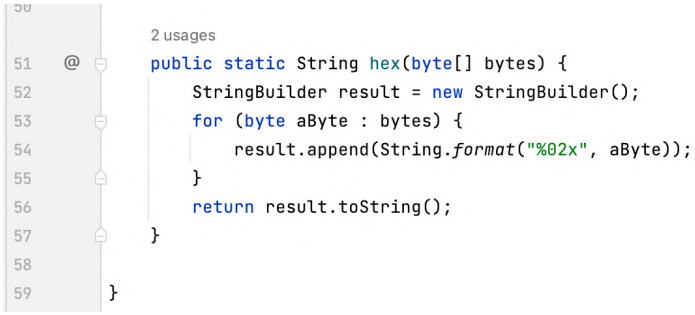
```
import java.nio.charset.StandardCharsets;
import java.util.HexFormat;

public class Main {
    public static void main(String[] args)  {

        String text = "hello";
        String k = "01234567890234567890";

        if ((args.length) > 1) {
            text = args[1];
        }
        if ((args.length) > 2) {
           k = args[2];
        }



        System.out.println("Text:\t" + text);
        System.out.println("Key:\t" + k);
        System.out.println("--------");
        byte[] key = HexFormat.of().parseHex(k);
```

Рисунок А.8 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.



```
28          PRESENT cipher = new PRESENT(key);
29
30          long startTime = System.nanoTime();
31          byte[] encrypted = cipher.encrypt(text.getBytes(StandardCharsets.UTF_8));
32          long endTime = System.nanoTime();
33
34          System.out.println ("Encrypt time: "+(endTime - startTime));
35          System.out.println("Cipher:\t\t" + hex(encrypted));
36
37
38          //------
39          byte[] decrypted = cipher.decrypt(encrypted);
40
41          System.out.println ("Decrypt time: "+(endTime - startTime));
42          System.out.println("Decrypted:\t" + hex(decrypted));
43
44
45          String decodedString = new String(decrypted, StandardCharsets.UTF_8);
46          System.out.println("Decrypted:\t" +decodedString);
47      }
```

Рисунок А.9 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.



```
                2 usages
51      @       public static String hex(byte[] bytes) {
52                  StringBuilder result = new StringBuilder();
53                  for (byte aByte : bytes) {
54                      result.append(String.format("%02x", aByte));
55                  }
56                  return result.toString();
57              }
58
59      }
```

Рисунок А.10 – Програмна реалізація блокового алгоритму PRESENT[45] на мові програмування Java.

```
/Library/Java/JavaVirtualMachines/jdk-17.0.4.jdk
Text:   hello world
Key:    09876543210987654321
--------
Encrypt time: 0.0011491909999676864
Cipher:     e1f4aa6b33766e64
Decrypt time: 0.0011066029996982252
Decrypted:  726c640505050505
Decrypted text: hello world

Process finished with exit code 0
```

Рисунок А.11 – Результат програмної реалізації блокового алгоритму PRESENT[45] на мові програмування Java.

```
/Library/Java/JavaVirtualMachines/jdk
Text:   demenko
Key:    09876543210987654321
--------
Encrypt time: 0.0015359120002358395
Cipher:     644a273ac173236a
Decrypt time: 0.0016914369998630718
Decrypted:  64656d656e6b6f01
Decrypted text: demenko

Process finished with exit code 0
```

Рисунок А.12 – Результат програмної реалізації блокового алгоритму PRESENT[45] на мові програмування Java.

```
Text:   demenko
Key:    10203010404040501320
--------
Encrypt time: 0.031156597000062902
Cipher:     6b7b374eed4fd83e
Decrypt time: 0.0014917019998392789
Decrypted:  64656d656e6b6f01
Decrypted text: demenko

Process finished with exit code 0
```

Рисунок А.13 – Результат програмної реалізації блокового алгоритму PRESENT[45] на мові програмування Java.

ДОДАТОК Б

КОМП’ЮТЕРНІ НАУКИ ТА КІБЕРБЕЗПЕКА

COMPUTER SCIENCE AND CYBERSECURITY (CS&CS)

Рисунок Б.1 - Обкладинка журналу [1]

ISSN 2519-2310 CS&CS, Issue 1(21) 2022

УДК 004.056.55

ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ АЛГОРИТМІВ МАЛОРЕСУРСНОЇ КРИПТОГРАФІЇ У ДЕЦЕНТРАЛІЗОВАНИХ СЕРЕДОВИЩАХ

Євгеній Деменко

Харківський національний університет імені В.Н. Каразіна, Харків, 61022, Україна
xa11868404@student.karazin.ua

Рецензент: Сергій Толюпа, д.т.н., проф., Київський національний університет імені Т. Шевченка, м. Київ, Україна.
tolupa@i.ua

Надійшла: Липень 2022.

***Анотація**: Метою даного матеріалу є ознайомлення з областю досліджень застосування малоресурсних алгоритмів криптографії для систем Інтернету речей (IoT) та можливості прямого впровадження в децентралізованих системах. За останні кілька років Інтернет речей став однією з найважливіших технологій століття. Зараз людство досягло високого рівня розвитку технологій, що дозволяє налаштовувати взаємодію між пристроями та створювати безперебійний зв'язок між людьми, процесами та речами. З появою 5G технології, IoT стали центром розвитку майже для всіх сучасних галузей. Пристрої в цій архітектурі значно менші та мають низьке енергоспоживання. Звичайні алгоритми шифрування, як правило, дорогі в обчислювальному плані через їхню складність і вимагають багато раундів, однак це може поставити під загрозу бажану цілісність. Криптографія з низьким ресурсом — це компроміс між вартістю впровадження, швидкістю, безпекою, продуктивністю та енергоспоживанням на пристроях IoT. Мотивація полегшеної криптографії полягає в тому, щоб використовувати менше пам'яті, менше обчислювальних ресурсів і менше енергоспоживання, щоб забезпечити рішення безпеки, яке може працювати на пристроях з обмеженими ресурсами. Блокові шифри мають фіксовану довжину бітів і різні кроки перетворення, які визначаються симетричним ключем. Блокові шифри дуже універсальні, що дуже корисно з точки зору IoT. Ще одна перевага полягає в тому, що цей процес має майже ідентичні методи шифрування та дешифрування. Тому його можна реалізувати з меншими ресурсами.*

***Ключові слова:** малоресурсна криптографія; легка криптографія; Інтернет речей; блокові шифри; IoT.*

1. Вступ

Сьогодні широкосмуговий Інтернет є загальнодоступним, а вартість підключення постійно знижується. Як наслідок, все більше гаджетів та різних датчиків підключаються до всесвітньої мережі Інтернет [1-2]. Всі ці тенденції створюють сприятливе підґрунтя для розвитку Інтернету речей (IoT). Однак, навколо Інтернету речей існує багато складнощів [3]. Перш за все це зв'язано зі складністю, як елементної бази, так і спеціальних алгоритмів обробки даних, що реалізуються в IoT пристроях. Процес обміну такою великою кількістю даних починається з самих пристроїв, які повинні безпечно взаємодіяти з платформою [3].

Пристрої, з яких складається система *Інтернет Речей* - це будь-який фізичний об'єкт, який можна унікально ідентифікувати (за допомогою URI або унікального ідентифікатора ресурсу) і який може надсилати/отримувати дані шляхом підключення до мережі [1]. Прикладами є транспортні засоби, промислові контролери, *RFID*-мітки, сенсорні вузли, смарт-карти, побутова техніка, тощо [2]. Вони можуть бути з'єднані між собою, з центральним сервером/мережею серверів або через хмарні сервіси.

Сутність Інтернету речей - зв'язок та обмін інформацією [1]. Однак, всі ці дані не генеруються лише для того, щоб їх десь зберігати і «забути про їх існування». Основна ціль їх використання, це автоматизація. IoT практично «стирає» розрив між цифровим та фізичним світом, однак має і зворотний бік процесу - компрометація IoT пристроїв, може мати небезпечні наслідки в «реальному» світі.

Як правило пристрій Інтернету речей, містять один або декілька датчиків, які використовуються для збору даних та підтримки мережевих інтерфейсів. Тип і номенклатура даних,

Рисунок Б.2 — Перша сторінка статті [1], с. 20