КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)
кібербезпеки та комп'ютерної інженерії
(назва випускової кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ бакалавр

(бакалавр, магістр)

на тему:

Захищена система IoT на міні-комп'ютері Raspberry Pi. Серверна частина

Задорожний Іван Михайлович
(прізвище, ім'я та по батькові здобувача повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет**)**

кібербезпеки та комп'ютерної інженерії

(назва кафедри**)**

ЗАТВЕРДЖУЮ

Завідувач кафедри

Делембовський Максим Михайлович

„___" _____2026 року

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ бакалавр

(бакалавр, магістр)

Захищена система IoT на міні-комп'ютері Raspberry Pi. Серверна частина

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач

Задорожний Іван Михайлович

(прізвище, ім'я та по батькові повністю)

123-комп'ютерна Інженерія

(спеціальність)

комп'ютерні системи та мережі

(освітня програма)

Група КСМ-22

Керівник к.т.н., доцент, Вишняков В. М.

(прізвище та ініціали)

Рецензент

(вчене звання, науковий ступінь)

Ідентичність підтверджую

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Випускова кафедра: кібербезпеки та комп'ютерної інженерії
Ступінь вищої освіти: бакалавр
Спеціальність: 123-комп'ютерна Інженерія
Освітня програма: комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри
Делембовський Максим
Михайлович
„___"_____2026 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ бакалавр

(бакалавр, магістр)

Задорожний Іван Михайлович
(прізвище, ім'я та по батькові здобувача)
Тема роботи Захищена система IoT на міні-комп'ютері Raspberry Pi. Серверна частина

1. затверджена наказом ректора КНУБА № 577/52-25/13.2/26 від «14» травня 2026 року
2. Керівник роботи Вишняков Володимир Михайлович к.т.н., доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)
3. Термін подання здобувачем роботи до захисту_____
4. Зміст пояснювальної записки за розділами:

Р.1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

Р. 2. ОБҐРУНТУВАННЯ ТА РОЗРОБЛЕННЯ РІШЕННЯ

Р. 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4. 5. Графічний матеріал за розділами
   Р. 1. 8 Таблиць, 10 Рисунків
   Р. 2. 7 Таблиць, 10 Рисунків
   Р. 3. 13 Таблиць, 4 Рисунки

5. Консультанти розділів кваліфікаційної випускної роботи:

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		Дата	Підпис
Розділ 1.	Вишняков В. М. к.т.н., доцент		
Розділ 2.	Вишняков В. М. к.т.н., доцент		
Розділ 3.	Вишняков В. М. к.т.н., доцент		
Розділ 4.			
Розділ 5.			

6. Календарний план виконання роботи:

Види робот та їх вміст	Дата виконання
Розділ 1.	25.04.2026
Розділ 2.	20.05.2026
Розділ 3.	02.06.2026
Розділ 4.	
Розділ 5.	
Остаточне оформлення роботи	10.06.2026
Направлення роботи для перевірки на плагіат	
Попередній захист роботи на випусковій кафедрі	
Направлення роботи на рецензування	

7. Дата видачі завдання _____

Керівник _____ Вишняков В. М.

(Підпис) (прізвище та ініціали)

Здобувач _____ Задорожний І. М.

(Підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ *здобувача українською та англійською мовами* *Задорожний Іван Михайлович* *Zadorozhnyi Ivan Mykhailovych*		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	Захищена система IoT на міні-комп'ютері Raspberry Pi. Серверна частина Protected IoT system on mini-computer Raspberry Pi. Server side		
Освітній ступінь	бакалавр		
Факультет	автоматизації та інформаційних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерії		
Спеціальність	123-комп'ютерна інженерія		
Освітня програма	комп'ютерні системи та мережі		
Керівник	Вишняков В. М.		
Обсяг роботи:	*Пояснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	96	3	16
Розділ 1	АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ 28 стор.		
Розділ 2	ОБҐРУНТУВАННЯ ТА РОЗРОБЛЕННЯ РІШЕННЯ 28 стор.		
Розділ 3	РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ 19 стор.		
Розділ 4	-		
Розділ 5	-		
Висновки по роботі	2 стор.		
Ключові слова: Keywords:	Інтернет речей, Raspberry Pi, сервер-посередник, інформаційна безпека, шифр Вернама, протокол Діффі-Геллмана, TCP, HTTP, дистанційне керування. Internet of Things, Raspberry Pi, proxy server, information security, Vernam cipher, Diffie-Hellman protocol, TCP, HTTP, remote control.		

Здобувач Задорожний І. М.

Керівник Вишняков В. М.

АНОТАЦІЯ

Кваліфікаційна випускна робота присвячена розробленню серверної частини захищеної системи Інтернету речей для дистанційного керування об'єктами на базі мікрокомп'ютера Raspberry Pi. Актуальність теми зумовлена стрімким поширенням систем дистанційного керування на базі мікрокомп'ютерів і потребою в простих та надійних засобах їх захисту, що не вимагають ані виділеної IP-адреси, ані дорогого обладнання. У роботі обґрунтовано застосування схеми із сервером-посередником, розроблено серверне програмне забезпечення для обміну даними з мікрокомп'ютером за протоколом TCP та взаємодії з користувачем за протоколом HTTP, реалізовано засоби захищеного обміну даними на основі шифру Вернама з узгодженням ключів за алгоритмом Діффі-Геллмана, а також розгорнуто, налаштовано й протестовано серверну частину. Практичне значення роботи полягає в можливості застосовувати розроблену серверну частину для побудови захищених багатокористувацьких систем дистанційного керування за моделлю SaaS.

Ключові слова: Інтернет речей, Raspberry Pi, сервер-посередник, інформаційна безпека, шифр Вернама, протокол Діффі-Геллмана, TCP, HTTP, дистанційне керування.

ABSTRACT

The qualification graduation work is devoted to the development of the server side of a secure Internet of Things system for remote control of objects based on the Raspberry Pi single-board computer. The relevance of the topic stems from the rapid spread of remote-control systems built on single-board computers and the need for simple and reliable means of protecting them that require neither a dedicated IP address nor expensive equipment. The work substantiates the use of a proxy-server scheme, develops server software for exchanging data with the microcomputer over the TCP protocol and interacting with the user over the HTTP protocol, implements secure data

exchange based on the Vernam cipher with key agreement by the Diffie-Hellman algorithm, and deploys, configures and tests the server side. The practical value of the work lies in the ability to use the developed server side for building secure multi-user remote-control systems under the SaaS model.

Keywords: Internet of Things, Raspberry Pi, proxy server, information security, Vernam cipher, Diffie-Hellman protocol, TCP, HTTP, remote control.

ЗМІСТ

ВСТУП

Інтернет речей перетворився з технологічної концепції на повсякденну інфраструктуру, що охоплює промисловість, транспорт, енергетику, медицину та житлове господарство. За даними аналітичної компанії IoT Analytics, наприкінці 2024 року у світі налічувалося близько вісімнадцяти мільярдів підключених пристроїв, і це зростання не сповільнюється [1, 29]. Дедалі більше фізичних об'єктів стають керованими дистанційно, через загальнодоступні канали мережі Інтернет.

Саме відкритість цих каналів і породжує головну проблему галузі. Пристрої Інтернету речей часто проєктують з міркувань низької вартості та простоти, через що захисту приділяють недостатньо уваги: вони мають слабкі паролі за замовчуванням, відкриті порти й позбавлені механізмів оновлення. Скомпрометований пристрій загрожує не лише власній системі, адже його ресурси можуть бути використані для розподілених атак типу «відмова в обслуговуванні» на інші вузли мережі. За опитуванням компанії Gemalto, близько 90 % користувачів не впевнені в захищеності своїх рішень [2].

Дієвий спосіб подолання цієї проблеми полягає у відмові від прямого підключення керованого вузла до Інтернету на користь схеми із сервером-посередником. За такої схеми мікрокомп'ютер, що керує об'єктами, під'єднується до мережі через маршрутизатор і не отримує реальної IP-адреси, тому керувати ним ззовні можна лише через захищений канал. Сервер-посередник фільтрує потоки даних між користувачем та об'єктами, а в разі його компрометації відомості про керовані об'єкти зберігають цілісність.

Керування об'єктами через мережу не потребує великих обсягів даних, тому дає змогу застосовувати найдосконаліші методи криптографічного захисту. Абсолютну, математично доведену стійкість забезпечує шифр Вернама [3, 4], відомий як одноразовий блокнот, а узгодження випадкових послідовностей для нього виконується за алгоритмом Діффі-Геллмана [5] над полем Галуа, параметри якого унеможливлюють розкриття даних.

Розглянута система є практичною реалізацією цього підходу. Вона призначена для дистанційного керування вісьмома двійковими об'єктами (реле), під'єднаними до мікрокомп'ютера Raspberry Pi через інтерфейс GPIO, і складається з клієнтської частини, що працює на самій Raspberry Pi, та серверної частини на віддаленому сервері-посереднику. Серверна частина приймає від мікрокомп'ютера стан об'єктів, передає йому команди керування й надає користувачеві вебінтерфейс для спостереження та управління. Клієнтська частина є предметом окремої кваліфікаційної роботи, тоді як ця робота присвячена серверній частині.

Актуальність теми зумовлена стрімким поширенням систем дистанційного керування на базі мікрокомп'ютерів і потребою в простих та надійних засобах їх захисту, що не вимагають ані виділеної IP-адреси, ані дорогого обладнання.

Метою роботи є підвищення інформаційної безпеки систем дистанційного керування об'єктами на базі Raspberry Pi через розроблення серверної частини захищеної системи Інтернету речей, побудованої за схемою сервера-посередника.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати сучасний стан розвитку Інтернету речей, типові архітектури таких систем і притаманні їм загрози безпеки.

2. Обґрунтувати застосування схеми із сервером-посередником для захисту систем Інтернету речей від несанкціонованого доступу та атак на відмову в обслуговуванні.

3. Обґрунтувати вибір апаратної платформи, операційної системи та програмних засобів серверної частини.

4. Розробити серверне програмне забезпечення для обміну даними з мікрокомп'ютером за протоколом TCP та взаємодії з користувачем за протоколом HTTP.

5. Реалізувати засоби захищеного обміну даними між складовими системи.

6. Розгорнути, налаштувати й протестувати серверну частину та оцінити надійність її роботи.

Об'єктом дослідження є процеси віддаленого керування об'єктами в системах Інтернету речей та забезпечення їх інформаційної безпеки.

Предметом дослідження є серверна частина захищеної системи Інтернету речей, а саме програмні засоби сервера-посередника, що забезпечують безпечний обмін даними між вебінтерфейсом користувача та віддаленим мікрокомп'ютером Raspberry Pi.

Практичне значення роботи полягає в тому, що розроблену серверну частину можна використовувати для побудови захищених систем дистанційного керування, а здатність одного сервера обслуговувати багатьох користувачів робить її придатною для надання послуги за моделлю SaaS.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Інтернет речей як предметна галузь

Інтернет речей (Internet of Things, IoT) є концепцією, за якою фізичні пристрої під'єднуються до глобальної мережі та обмінюються даними без постійної участі людини. Діапазон таких пристроїв надзвичайно широкий: від простих датчиків температури й вологості до складних промислових установок, транспортних засобів і побутової техніки. Спільною їх рисою є наявність обчислювального вузла, засобів зв'язку та можливості взаємодіяти з фізичним середовищем, збираючи дані або виконуючи певні дії. Завдяки цьому фізичні об'єкти стають частиною інформаційного простору, а керування ними та спостереження за ними можна здійснювати дистанційно.

Як окремий напрям Інтернет речей сформувався порівняно нещодавно. Сам термін запропонував у 1999 році Кевін Ештон [6], співзасновник дослідницького центру Auto-ID Center при Массачусетському технологічному інституті. Проте передумови концепції з'явилися значно раніше, ще у 1970-1980-х роках, коли почали створювати перші пристрої, здатні обмінюватися даними мережею. За кілька десятиліть технологія пройшла шлях від поодиноких експериментальних рішень до масового явища: пристрої стали компактнішими та потужнішими, а канали зв'язку швидшими й доступнішими. Сьогодні до мережі під'єднують як прості побутові гаджети, так і цілі автоматизовані виробництва.

Стрімке поширення Інтернету речей зумовлене кількома чинниками, що збіглися в часі: здешевленням мініатюрних датчиків і мікроконтролерів, поширенням бездротового зв'язку та появою доступних хмарних сервісів для зберігання й оброблення даних. Унаслідок цього під'єднання пристрою до мережі перестало бути технічно складним і дорогим, що відкрило шлях до масового впровадження таких рішень у промисловості, побуті та містах.

Практична цінність Інтернету речей полягає насамперед в автоматизації процесів. Підключені пристрої дозволяють у реальному часі збирати дані про стан

обладнання, приміщень чи технологічних процесів і на цій основі ухвалювати рішення або виконувати дії без втручання оператора. У промисловості це дає змогу завчасно виявляти несправності та запобігати простоям, у житловому секторі автоматично регулювати освітлення, опалення й енергоспоживання, у медицині дистанційно стежити за станом пацієнтів. Поєднання великих обсягів зібраних даних з аналітичними засобами відкриває можливості, недоступні традиційним системам керування, і саме тому Інтернет речей вважають одним з визначальних технологічних напрямів сучасності.

З погляду користувача головна перевага Інтернету речей полягає в переході від ручного до автоматизованого та дистанційного керування. Якщо раніше для зміни стану обладнання потрібна була фізична присутність оператора, то тепер ці дії виконуються віддалено, з будь-якої точки, де є доступ до мережі. Це не лише економить час, а й уможливлює керування об'єктами, доступ до яких ускладнений або небезпечний. Водночас перенесення керування в мережу робить надійність і захищеність каналу зв'язку критично важливими, що й визначає напрям цієї роботи.

Масштаби поширення технології наочно ілюструє статистика. За даними аналітичної компанії IoT Analytics [29], наприкінці 2024 року у світі налічувалося близько вісімнадцяти з половиною мільярдів підключених пристроїв, у 2025 році їх кількість зросла приблизно до двадцяти одного мільярда, а за прогнозами до 2030 року вона досягне близько тридцяти дев'яти мільярдів [1, 29]. Динаміку кількості пристроїв та прогноз наведено в таблиці 1.1.

Таблиця 1.1 - Кількість підключених пристроїв Інтернету речей у світі: фактичні дані та прогноз, млрд

Рік	**Кількість пристроїв, млрд**
2023	16,6
2024	18,5
2025	21,1
2030 (прогноз)	39
2035 (прогноз)	понад 50

Наведені дані свідчать, що Інтернет речей охоплює практично всі галузі економіки, від житлового господарства й охоронних систем до промисловості та державного сектору, причому кількість пристроїв у кожній зі сфер щороку зростає. Водночас стрімке поширення технології загострює низку проблем, головними з яких є інформаційна безпека, приватність даних та масштабованість інфраструктури. Саме питання безпеки, як буде показано далі, є визначальним для систем дистанційного керування об'єктами, яким присвячено цю роботу.

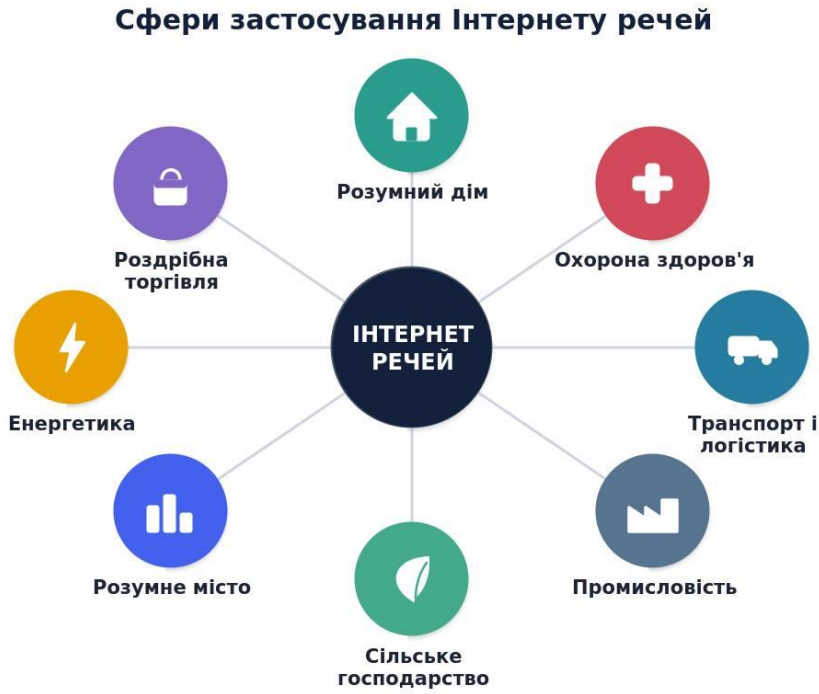

Рисунок 1.1 - Сфери застосування Інтернету речей

1.2 Архітектура та складові систем Інтернету речей

Незважаючи на різноманіття конкретних реалізацій, більшість систем Інтернету речей побудовано за спільною багаторівневою схемою [7]. Зазвичай у ній виділяють чотири рівні: рівень пристроїв, мережевий рівень, рівень обробки даних та рівень застосунків. Такий поділ дозволяє відокремити фізичну взаємодію зі середовищем від передавання, обробки й подання інформації, а отже, проєктувати та вдосконалювати кожен рівень незалежно. Узагальнену архітектуру показано на рисунку 1.2.

Рисунок 1.2 - Багаторівнева архітектура систем Інтернету речей

Рівень пристроїв утворюють фізичні елементи системи, які безпосередньо взаємодіють зі середовищем: датчики, що збирають дані, та виконавчі механізми, що виконують дії. Мережевий рівень забезпечує зв'язок між пристроями та серверами і може використовувати найрізноманітніші технології, від дротового Ethernet до бездротових Wi-Fi, Bluetooth чи мобільних мереж. Рівень обробки даних відповідає за приймання, зберігання та аналіз зібраної інформації; він може бути реалізований як на хмарних серверах, так і безпосередньо на периферійних вузлах, поблизу джерела даних. Нарешті, рівень застосунків є тим, з чим взаємодіє кінцевий користувач, найчастіше через веб- або мобільний застосунок, що відображає дані та дозволяє керувати системою.

Взаємодія між рівнями відбувається через мережу і утворює характерний потік даних. Датчики та виконавчі механізми збирають інформацію про стан об'єктів і передають її через мережевий інтерфейс до сервера, де вона обробляється і за потреби зберігається. У зворотному напрямку, від рівня застосунків через сервер до пристроїв, надходять команди керування. Саме така двоспрямована схема обміну, у якій сервер виступає сполучною ланкою між користувачем та об'єктами, лежить в основі системи, розглянутої в цій роботі.

Рівень обробки даних може бути організований двома способами. За централізованої, хмарної обробки всі дані надсилаються на віддалені сервери, що зручно для масштабних систем, але створює навантаження на канали зв'язку та збільшує затримки. За периферійної обробки частину обчислень виконують ближче до джерела даних, на проміжних вузлах або шлюзах, що знижує обсяг переданих даних і пришвидшує реакцію системи. У системі, що розглядається, сервер-посередник фактично відіграє роль такого проміжного вузла між користувачем та об'єктами керування.

Мережевий рівень відповідає за передавання даних між пристроями та сервером і визначає, якими каналами та протоколами відбувається обмін. Рівень обробки даних перетворює, зберігає та аналізує отримані відомості, а рівень застосунків надає їх користувачеві у зручному вигляді та приймає його команди. Чітке розмежування цих рівнів дозволяє змінювати реалізацію одного з них, не зачіпаючи інших, що спрощує розвиток системи.

У межах цієї роботи увагу зосереджено на ланці, що поєднує рівень пристроїв із рівнем застосунків, на сервері-посереднику та каналах обміну з ним. Саме ця ланка визначає, наскільки безпечно команди користувача досягають керованих об'єктів, тому її захист є ключовим для всієї системи.

За призначенням складові рівня пристроїв поділяють на три групи: датчики, виконавчі механізми та мережеві інтерфейси. Датчики перетворюють фізичні величини, такі як температура, вологість, освітленість, тиск чи рух, на цифрові дані і фактично надають системі здатність сприймати навколишнє середовище. Вибір датчика визначається конкретним завданням: одні системи потребують контролю клімату, інші - виявлення руху чи вимірювання рівня освітлення. Мережеві інтерфейси, дротові або бездротові, забезпечують під'єднання пристрою до мережі та обмін даними; їх вибір залежить від вимог до швидкості, дальності, енергоспоживання та захищеності.

Особливу роль у системах автоматизації відіграють виконавчі механізми, або актуатори, що виконують зворотну до датчиків функцію: вони перетворюють електричний сигнал на фізичну дію. Існують електромеханічні, гідравлічні,

пневматичні та теплові актуатори, проте найпоширенішим типом у системах дистанційного керування є електромеханічні реле. За керувальним сигналом реле замикає або розмикає зовнішнє електричне коло і в такий спосіб вмикає чи вимикає під'єднаний пристрій. Саме вісьмома релейними виходами керує система, розглянута в цій роботі, тому надалі під об'єктами керування розуміємо двійкові виконавчі елементи, кожен з яких може перебувати в одному з двох станів: увімкненому або вимкненому.

У термінах цієї архітектури розглянута система має таку відповідність складових. Роль обчислювального вузла рівня пристроїв виконує одноплатний комп'ютер Raspberry Pi, виконавчими механізмами є вісім електромеханічних реле, а мережевий інтерфейс забезпечує під'єднання до сервера-посередника. Датчики в системі не використовуються, оскільки її призначення полягає у керуванні станом об'єктів, а не у зборі вимірювань; натомість зворотний зв'язок реалізовано програмно, шляхом передавання поточного стану реле назад на сервер.

1.3 Платформи Інтернету речей

Окремою складовою екосистеми Інтернету речей є програмні платформи, що пов'язують апаратну частину з прикладним програмним забезпеченням, яке нею керує. Платформа Інтернету речей є сукупністю служб для під'єднання пристроїв, керування ними, збирання та обробки даних. Вона утворює сполучну ланку між сенсорами, актуаторами й пристроями, з одного боку, та користувацькими застосунками, з іншого. Попри велику кількість таких платформ, більшість із них надає схожий набір базових можливостей.

До ключових можливостей платформ Інтернету речей належать: під'єднання пристроїв та керування їхніми з'єднаннями; збирання, зберігання й обробка даних із застосуванням аналітичних засобів; віддалене керування пристроями, зокрема оновлення їх програмного забезпечення та відстеження стану; забезпечення безпеки, що охоплює шифрування обміну й автентифікацію пристроїв; а також інтеграція з іншими інформаційними системами. Поєднання цих можливостей дозволяє будувати завершені рішення без розроблення інфраструктури з нуля.

Готові платформи спрощують розгортання типових рішень, проте їх використання не завжди доцільне. За підвищених вимог до захищеності та незалежності від сторонніх постачальників розробники часто реалізують власні легкі серверні рішення, що виконують лише необхідні функції. Саме такий підхід застосовано в цій роботі: замість громіздкої універсальної платформи серверну частину побудовано з компактних спеціалізованих сервісів, кожен з яких виконує чітко визначену задачу обміну або керування. Це зменшує поверхню можливих атак та полегшує супровід системи.

Поширені комерційні платформи здебільшого є хмарними, тобто дані користувача проходять через інфраструктуру стороннього постачальника. Це спрощує впровадження, але створює залежність від постачальника, потенційні ризики для приватності та додаткові витрати на обслуговування. Для систем, де ключовою вимогою є захищеність і контроль над даними, самостійно розгорнутий сервер-посередник є виправданою альтернативою, оскільки залишає всі дані під контролем власника системи.

1.4 Протоколи та технології передавання даних

Передавання даних є одним з ключових аспектів роботи систем Інтернету речей. Для організації обміну застосовують різні протоколи прикладного рівня та технології фізичного підключення, вибір яких залежить від вимог конкретної системи до швидкості, надійності, енергоспоживання та захищеності. Оскільки керування віддаленими об'єктами зазвичай не потребує передавання великих обсягів інформації, на перший план виходять не швидкість, а надійність доставляння та сумісність із наявною мережевою інфраструктурою.

Серед протоколів прикладного рівня найпоширенішими в Інтернеті речей є MQTT, CoAP та HTTP. Протокол MQTT побудовано на моделі «публікація - підписка» і розраховано на мережі з обмеженою пропускною спроможністю та низьким енергоспоживанням, тому його часто застосовують для автономних датчиків. Протокол CoAP орієнтовано на малопотужні пристрої, що працюють в обмежених умовах. Протокол HTTP, попри більшу надлишковість, лишається

зручним для систем, де пристрої взаємодіють із вебзастосунками: він підтримується всіма браузерами й не потребує встановлення додаткового програмного забезпечення на боці користувача. Основні характеристики цих протоколів узагальнено в таблиці 1.2.

Таблиця 1.2 - Порівняння поширених протоколів прикладного рівня систем Інтернету речей

Протокол	**Модель взаємодії**	**Транспорт**	**Типове застосування**
MQTT	Публікація - підписка	TCP	Автономні датчики, мережі з обмеженою смугою
CoAP	Запит - відповідь	UDP	Малопотужні пристрої в обмежених умовах
HTTP	Запит - відповідь	TCP	Взаємодія з вебзастосунками та браузерами
AMQP	Черги повідомлень	TCP	Надійний обмін у корпоративних системах

Технології фізичного підключення також різноманітні. Дротовий Ethernet забезпечує стабільне й швидке з'єднання, але обмежений розташуванням кабелю. Бездротовий Wi-Fi зручний для під'єднання пристроїв до локальної мережі та Інтернету, тоді як Bluetooth і його енергоощадний варіант BLE застосовують на невеликих відстанях. Для віддалених об'єктів використовують мобільні мережі або спеціалізовані технології далекого радіозв'язку з низьким енергоспоживанням, такі як LoRaWAN та NB-IoT. Незалежно від обраної технології, захищеність каналу забезпечують криптографічними засобами, зокрема протоколами TLS та DTLS.

Для систем дистанційного керування надійність доставляння команд важливіша за швидкість. Втрата команди або зміна порядку команд може призвести до того, що об'єкт перейде у стан, відмінний від бажаного, тому канал керування має гарантувати доставляння кожного повідомлення в незмінному вигляді. Цю вимогу задовольняє протокол TCP [10], який встановлює з'єднання, підтверджує отримання даних і відновлює втрачені сегменти, на відміну від протоколу UDP, що працює без гарантій доставляння.

На прикладному рівні в системах Інтернету речей поширені спеціалізовані протоколи. Протокол MQTT побудовано за моделлю публікації та підписки і розраховано на обмін короткими повідомленнями за участю проміжного брокера [8], а протокол CoAP орієнтовано на пристрої з обмеженими ресурсами [9]. У багатьох рішеннях, зокрема в розглянутому, для взаємодії з користувачем застосовують звичний протокол HTTP, що не потребує додаткового програмного забезпечення на боці користувача, окрім браузера.

У системі, що розглядається, обмін організовано на двох рівнях. Між мікрокомп'ютером і сервером-посередником використовується надійне з'єднання за протоколом TCP, що гарантує доставляння даних у незмінному порядку та без втрат. Взаємодія користувача із сервером відбувається за протоколом HTTP, що дозволяє керувати об'єктами зі звичайного браузера без встановлення спеціального програмного забезпечення. Такий вибір зумовлено невеликим обсягом даних керування та вимогою максимальної доступності інтерфейсу для користувача.

1.5 Апаратні платформи: мікроконтролери та мікрокомп'ютери

Апаратною основою пристроїв Інтернету речей слугують два класи обчислювальних засобів: мікроконтролери та мікрокомп'ютери, до яких належать одноплатні комп'ютери. Вони суттєво різняться складністю, обчислювальними можливостями, вартістю та енергоспоживанням, тому вибір між ними визначається вимогами конкретного проєкту. Основні відмінності цих двох класів пристроїв узагальнено в таблиці 1.3.

Таблиця 1.3 - Порівняння мікропроцесора та мікроконтролера

Критерій порівняння	**Мікропроцесор**	**Мікроконтролер**
Будова	Окрема мікросхема з арифметико-логічним пристроєм, пристроєм керування та регістрами	Інтегрує процесор, пам'ять, порти вводу-виводу та контролер переривань на одному кристалі
Самодостатність	Потребує зовнішніх компонентів для роботи	Є автономним функціональним блоком
Порти вводу-виводу	Власні порти відсутні	Має вбудовані порти вводу-виводу

Критерій порівняння	Мікропроцесор	Мікроконтролер
Призначення	Обчислення загального призначення	Виконання конкретних прикладних задач
Енергоспоживання	Вищі витрати, менше засобів енергозбереження	Нижчі витрати, розвинені засоби енергозбереження

Мікроконтролери є простими обчислювальними системами, що виконують одну задачу або вузький набір задач і об'єднують процесор, пам'ять та порти вводу-виводу на одному кристалі. Вони дешеві, надійні та енергоефективні, проте мають обмежені можливості обчислення й зв'язку. Мікрокомп'ютери, навпаки, є значно потужнішими та гнучкішими: вони містять багатоядерні процесори, помітно більший обсяг оперативної пам'яті й здатні працювати під керуванням повноцінної операційної системи. Це дозволяє виконувати на них складніші обчислення, обробляти великі обсяги даних та підтримувати розширені засоби зв'язку, що робить їх придатними для побудови серверних і шлюзових вузлів.

Найпоширенішим представником класу мікрокомп'ютерів є Raspberry Pi, який завдяки невисокій вартості, гнучкості та великій спільноті розробників став стандартним вибором для проєктів Інтернету речей. Платформу побудовано на архітектурі ARM, що широко застосовується у вбудованих системах. Модель Raspberry Pi 3 Model B+, зокрема, містить чотириядерний процесор Cortex-A53, один гігабайт оперативної пам'яті, інтерфейси Ethernet, Wi-Fi та Bluetooth, а також сорокаконтактний роз'єм GPIO для під'єднання зовнішніх компонентів. Зовнішній вигляд плати та її інтерфейс показано на рисунку 1.3.

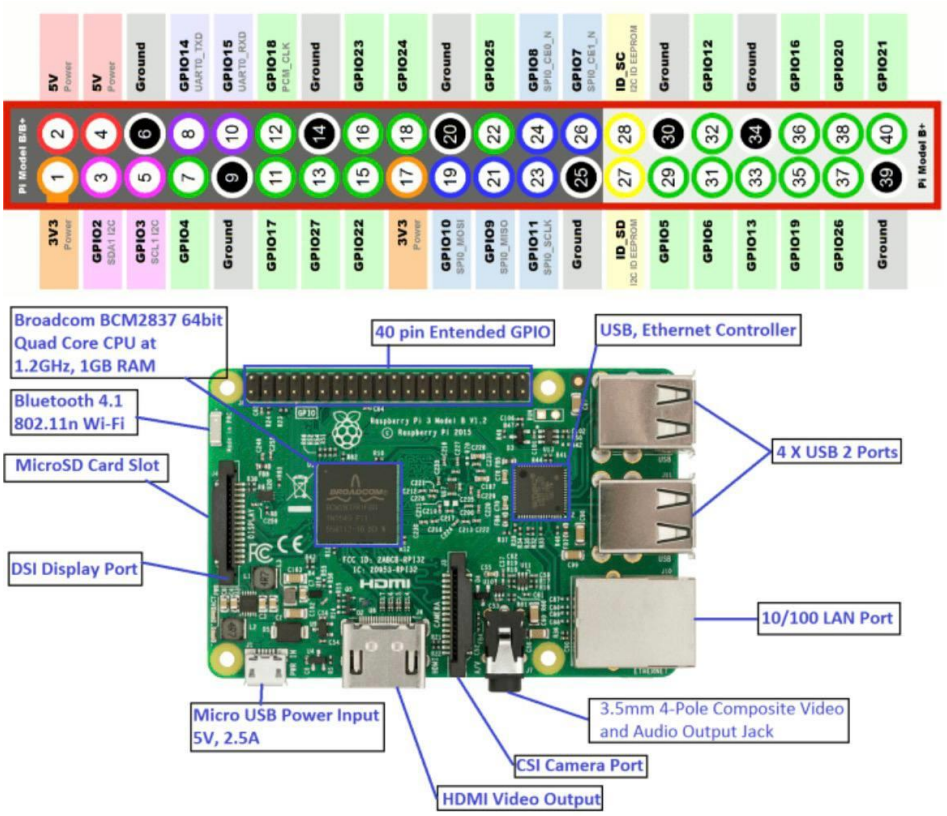

Рисунок 1.3 - Одноплатний комп'ютер Raspberry Pi та його інтерфейс GPIO

Лінійка Raspberry Pi охоплює моделі різної потужності, що дозволяє добирати плату під конкретну задачу. Стисле порівняння кількох поширених моделей наведено в таблиці 1.4. Для розглянутої системи дистанційного керування достатньо можливостей моделі Raspberry Pi 3, оскільки вона забезпечує потрібну кількість контактів GPIO, мережеві інтерфейси та здатна працювати під керуванням повноцінної операційної системи.

Таблиця 1.4 - Порівняння поширених моделей Raspberry Pi

Модель	**Тип**	**Процесор / пам'ять**	**Орієнтовне застосування**
Raspberry Pi 4 Model B	Мікрокомп'ютер	4 ядра, до 8 ГБ ОЗП	Вимогливі задачі, обробка відео та великих даних
Raspberry Pi 3 Model B+	Мікрокомп'ютер	4 ядра Cortex-A53, 1 ГБ ОЗП	Універсальні проєкти Інтернету речей
Raspberry Pi Zero W	Мікрокомп'ютер	1 ядро, 512 МБ ОЗП	Компактні рішення з обмеженим бюджетом

Модель	Тип	Процесор / пам'ять	Орієнтовне застосування
Raspberry Pi Pico	Мікроконтролер	RP2040, без ОС	Прості задачі керування, низьке енергоспоживання

Інтерфейс GPIO (General Purpose Input/Output) робить Raspberry Pi придатним для задач керування. Він є набором контактів загального призначення, кожен з яких можна програмно перевести у стан логічної одиниці або нуля чи зчитати з нього сигнал. До цих контактів під'єднують зовнішні пристрої, зокрема релейні модулі, що дозволяє керувати реальними електричними колами. У розглянутій системі для керування вісьмома реле використовують контакти з номерами 4, 17, 18, 22, 23, 24, 25 і 27, що відповідають вісьмом окремим об'єктам.

Безпосереднє під'єднання навантаження до контактів GPIO неможливе, оскільки вони розраховані на малі струми та низьку напругу. Тому між мікрокомп'ютером і керованими колами встановлюють релейний модуль. Сигнал з контакту GPIO керує реле, яке своєю чергою комутує зовнішнє, потужніше коло, причому керувальна та силова частини зазвичай гальванічно розділені оптронами для захисту мікрокомп'ютера. Багато релейних модулів керуються інверсним сигналом, за якого логічний нуль вмикає реле, а логічна одиниця вимикає його, що враховують під час розроблення керувальної програми.

За своїми характеристиками одноплатний комп'ютер Raspberry Pi добре підходить на роль керувального вузла [11]. Він має достатні для таких задач обчислювальні ресурси, споживає мало енергії, має невеликі розміри й підтримує повноцінну операційну систему, що дозволяє виконувати на ньому програми загального призначення. Поєднання цих властивостей з наявністю інтерфейсу GPIO робить його зручною основою для побудови пристроїв дистанційного керування.

Поряд із самостійно розгорнутими рішеннями поширені готові хмарні платформи Інтернету речей, які надають постачальники хмарних послуг. Вони беруть на себе зберігання даних, масштабування та частину питань безпеки, проте

передбачають передавання даних користувача на сторонні сервери та залежність від конкретного постачальника. Для систем, де важливими є контроль над даними та незалежність від зовнішніх сервісів, самостійно розгорнута серверна частина, як у цій роботі, лишається доречнішим вибором.

Важливою перевагою Raspberry Pi є здатність працювати під керуванням різних операційних систем. Найчастіше застосовують Raspberry Pi OS (раніше Raspbian) на основі Debian, проте плата підтримує також Ubuntu, Fedora та інші системи на базі Linux. Програмне забезпечення для таких систем розробляють різними мовами: Python зручний для швидкого створення прототипів, мови C та C++ забезпечують максимальну продуктивність і близький до апаратного контроль, а платформа Node.js, що використовує асинхронну модель вводу-виводу, добре підходить для мережевих застосунків. Вибір операційної системи та середовища виконання безпосередньо впливає на доступні бібліотеки й способи керування контактами GPIO, тому його обґрунтуванню приділено окрему увагу в наступному розділі.

1.6 Проблеми інформаційної безпеки систем Інтернету речей

Зі зростанням кількості підключених пристроїв пропорційно загострюється проблема їх захищеності. Підключення до загальнодоступної мережі робить пристрої Інтернету речей потенційними цілями для зловмисників, а компрометація таких пристроїв створює загрози, що виходять далеко за межі окремої системи. Виклики безпеки прийнято розділяти на три групи: безпека самих пристроїв, безпека комунікації та безпека даних.

Безпека пристроїв є найслабшою ланкою [25, 26]. Виробники нерідко проєктують пристрої з пріоритетом низької вартості та простоти, через що нехтують захистом: пристрої постачаються зі слабкими паролями за замовчуванням, мають відкриті порти й позбавлені механізмів оновлення програмного забезпечення. Безпека комунікації стосується захисту даних під час їх передавання мережею, адже без шифрування ці дані можуть бути перехоплені або

підмінені. Безпека даних охоплює захист інформації, що зберігається на пристроях або серверах, від викрадення та несанкціонованої зміни.

Стосовно систем дистанційного керування ці загрози набувають конкретних форм. Перехоплення трафіку дозволяє зловмисникові дізнатися про стан і режим роботи об'єктів. Підміна даних дає змогу нав'язати системі хибні команди або приховати реальний стан об'єктів. Несанкціоноване керування, тобто отримання прямого доступу до керованого вузла, є найнебезпечнішим, оскільки надає зловмисникові повний контроль над фізичними об'єктами. Нарешті, атаки на відмову в обслуговуванні можуть унеможливити керування у потрібний момент. Архітектура захищеної системи має враховувати всі ці форми загроз.

Заведено розглядати інформаційну безпеку через три основні властивості: конфіденційність, цілісність та доступність. Конфіденційність означає неможливість прочитання даних сторонньою особою, цілісність - неможливість їх непомітної зміни, а доступність - гарантованість доступу до системи та її функцій для законних користувачів. Системи дистанційного керування потребують одночасного забезпечення всіх трьох властивостей, оскільки порушення будь-якої з них безпосередньо впливає на здатність користувача безпечно керувати фізичними об'єктами.

Окремою, але важливою складовою безпеки є своєчасне оновлення програмного забезпечення. Значна частина успішних атак на пристрої Інтернету речей використовує давно відомі вади, які залишилися невиправленими через відсутність оновлень. Тому здатність системи отримувати оновлення безпеки та підтримуваність використаних програмних засобів є не менш суттєвими, ніж початковий рівень її захищеності.

Масштаб проблеми підтверджують відомі інциденти. Значна частина пристроїв Інтернету речей роками експлуатується зі стандартними паролями та без оновлень, що робить їх легкою здобиччю для зловмисників і дозволяє об'єднувати захоплені пристрої у великі мережі для проведення розподілених атак. Це показує, що безпеку потрібно закладати в систему вже на етапі її проєктування, а не додавати згодом.

Особливістю систем дистанційного керування є те, що наслідки порушення їх безпеки виходять за межі інформаційних. Несанкціоноване перемикання реального обладнання може призвести до матеріальних збитків або загрози безпеці людей, тому до захисту таких систем висувають підвищені вимоги порівняно із системами, що лише збирають дані.

Окрему й особливо серйозну загрозу становить використання скомпрометованих пристроїв для організації розподілених атак на відмову в обслуговуванні. Через недостатню захищеність зловмисники отримують можливість об'єднувати велику кількість пристроїв у ботнети та спрямовувати їх ресурси проти інших вузлів мережі. Кількість і потужність таких атак зростає разом із поширенням Інтернету речей: за наявними оцінками, щорічний приріст підключених пристроїв збільшувався приблизно від трьох мільярдів у 2017 році до п'яти мільярдів у 2020 році, що відповідно розширює потенційну базу для атак. Основні вектори атак на пристрої Інтернету речей наведено на рисунку 1.4.

Рисунок 1.4 - Основні вектори атак на пристрої Інтернету речей

Наочним підтвердженням реальності цих загроз стала поява у 2016 році шкідливої програми Mirai [12], яка масово вражала погано захищені пристрої Інтернету речей, зокрема камери та маршрутизатори, і об'єднувала їх у ботнет. Сформована в такий спосіб мережа здійснила одні з найпотужніших на той час

розподілених атак на відмову в обслуговуванні, тимчасово порушивши роботу великих інтернет-сервісів. Цей випадок засвідчив, що недостатньо захищені побутові пристрої здатні завдати шкоди далеко за межами власної системи.

Складність забезпечення безпеки в Інтернеті речей зумовлена кількома чинниками. По-перше, багато пристроїв мають обмежені обчислювальні ресурси, що ускладнює застосування стійких криптографічних алгоритмів. По-друге, через велике різноманіття сфер застосування й відмінні вимоги до них розроблення єдиних стандартів безпеки є складним завданням, попри очевидну потребу в ньому. По-третє, значна частина пристроїв не оновлюється регулярно, через що з часом накопичуються невиправлені вразливості. За результатами опитування компанії Gemalto, близько 90 % користувачів не впевнені в захищеності своїх рішень, а дослідники відносять мережевий рівень до найуразливішої ланки систем Інтернету речей.

Подолання цих загроз потребує заходів на кількох рівнях. Захист самих пристроїв досягається застосуванням надійних паролів, регулярним оновленням програмного забезпечення та закриттям непотрібних портів. Захист комунікації забезпечується шифруванням каналу обміну даними. Захист даних передбачає шифрування сховища та регулярне резервне копіювання. Для систем дистанційного керування найбільший вплив на загальну захищеність має саме захист каналу обміну між користувачем та об'єктами, тому далі розглянуто криптографічні засоби, що його забезпечують, та архітектурні рішення, які унеможливлюють прямий доступ до керованого вузла.

1.7 Криптографічні засоби захисту даних

Центральне місце серед засобів захисту даних в Інтернеті речей посідає криптографія, яка забезпечує конфіденційність, цілісність та автентичність інформації під час її передавання й зберігання. Розглянемо основні криптографічні механізми, що становлять теоретичне підґрунтя захищеного обміну даними між складовими системи дистанційного керування.

Шифрування є процесом перетворення зрозумілого тексту на незрозумілий за допомогою криптографічного ключа. Розрізняють симетричне та асиметричне шифрування [13, 24]. За симетричного шифрування той самий ключ використовують і для зашифровування, і для розшифровування, що забезпечує високу швидкодію, але потребує захищеного способу передавання спільного ключа. За асиметричного шифрування застосовують пару ключів: відкритий для зашифровування та закритий для розшифровування, що знімає проблему передавання спільного ключа, проте потребує більших обчислювальних витрат. Узагальнену схему процесів зашифровування й розшифровування показано на рисунку 1.5.

Рисунок 1.5 - Узагальнена схема процесів зашифровування та розшифровування даних

На практиці симетричне та асиметричне шифрування часто поєднують: повільніше асиметричне шифрування використовують для безпечного узгодження спільного ключа, а подальший обмін даними виконують швидшим симетричним алгоритмом із цим ключем. Окремою задачею є безпечне узгодження спільного ключа сторонами, що спілкуються відкритим каналом, без прямого передавання ключа. Її розв'язують спеціальні протоколи обміну ключами, конкретне застосування яких у розробленій системі розглянуто в наступному розділі.

Найскладнішою на практиці є саме проблема розподілу ключів. Для симетричного шифрування обидві сторони повинні мати однаковий секретний ключ, але передати його незахищеним каналом неможливо без ризику перехоплення, а організація окремого захищеного каналу лише для ключів

повертає задачу до початкового стану. Історично цю суперечність розв'язали методи відкритого узгодження ключів, які дозволяють сторонам отримати спільний секрет, обмінюючись лише відкритими повідомленнями. Саме такий підхід покладено в основу захищеного обміну в розглянутій системі.

Хешування слугує для забезпечення цілісності даних. Хеш-функція перетворює вхідні дані довільного розміру на бітову послідовність фіксованої довжини, причому цей процес є одностороннім: відновити вихідні дані з хешу неможливо. Будь-яка, навіть найменша, зміна вхідних даних призводить до зміни хешу, що дозволяє виявляти спотворення інформації під час її передавання або зберігання.

Цифрові підписи забезпечують автентичність даних і ґрунтуються на асиметричному шифруванні: закритий ключ використовують для створення підпису, а відкритий для його перевірки. Це гарантує, що дані надійшли саме від визначеного відправника і не були змінені в дорозі. Тісно пов'язана з цим автентифікація є процесом підтвердження ідентичності користувача, пристрою або системи. У системах Інтернету речей вона особливо важлива через велику кількість під'єднаних пристроїв, кожен з яких є потенційним вектором атаки. Для автентифікації застосовують паролі, цифрові сертифікати, одноразові паролі та мультифакторні схеми, що поєднують кілька методів перевірки.

Особливе місце серед методів шифрування посідають системи, що забезпечують так звану досконалу секретність. Доведено, що абсолютну стійкість, яку неможливо подолати навіть за необмежених обчислювальних ресурсів, має шифр, у якому довжина випадкового ключа дорівнює довжині повідомлення, а сам ключ використовується лише один раз. Невеликий обсяг даних керування робить застосування такого підходу в системах дистанційного керування цілком реалістичним, що буде використано під час проєктування захищеного каналу.

Окремою складовою захисту є приватність даних, яка передбачає захист зібраної інформації від несанкціонованого доступу, мінімізацію обсягу збираних даних та надання користувачеві контролю над ними. Описані механізми становлять загальну криптографічну основу захисту. Конкретні рішення, застосовані в

розробленій системі, зокрема абсолютно стійкий шифр Вернама та алгоритм узгодження ключів Діффі-Геллмана над полем Галуа, спираються на ці засади і детально розглянуті в розділі 2.

1.8 Аналіз схем організації безпечного обміну даними

Захищеність системи дистанційного керування значною мірою визначається тим, у який спосіб керований вузол під'єднано до мережі Інтернет. Можливі два принципові підходи, які доцільно порівняти, щоб обґрунтувати вибір архітектури системи.

На практиці для доступу до пристроїв, розташованих за маршрутизатором, застосовують кілька підходів. Найпростіший полягає у відкритті портів маршрутизатора та використанні служб динамічних доменних імен, проте він фактично робить пристрій безпосередньо доступним з Інтернету з усіма супутніми ризиками. Інший підхід передбачає організацію віртуальної приватної мережі, що забезпечує захищений доступ, але потребує складнішого налаштування та кваліфікованого адміністрування. Найбільш збалансованим для масового застосування є підхід із проміжним сервером, до якого пристрій під'єднується самостійно. Саме тому далі докладно порівнюються дві базові архітектури: пряме керування та керування через сервер-посередник.

Вибір між цими підходами не зводиться лише до зручності: він безпосередньо визначає рівень захищеності системи. Підходи, за яких керований вузол стає доступним з мережі Інтернет, переносять на нього всі ризики відкритого вузла, тоді як підхід із проміжним сервером дозволяє зосередити захист в одному місці й приховати керований вузол. Тому подальше порівняння зосереджено саме на наслідках кожної схеми для безпеки.

За схемою прямого керування, показаною на рисунку 1.5, сервер, до якого звертається термінал користувача, безпосередньо під'єднано до мережі Інтернет, і саме він керує об'єктами. Така схема найпростіша і цілком придатна для внутрішніх корпоративних мереж, проте в умовах загальнодоступної мережі має суттєві недоліки. Пряме підключення сервера до Інтернету відкриває шлях для

непередбачуваних зовнішніх загроз, що можуть втрутитися в процеси керування. Підвищується ймовірність встановлення зловмисниками власних шкідливих програм для організації атак на відмову в обслуговуванні. Окрім того, такий сервер потребує виділеної реальної IP-адреси, що пов'язано з додатковими витратами, а також кваліфікованого обслуговування для підтримання належного рівня захищеності.

Рисунок 1.6 - Схема прямого керування об'єктами через загальнодоступну мережу

Перелічених недоліків позбавлена схема з використанням сервера-посередника, показана на рисунку 1.7. У ній потоки даних між терміналом користувача та об'єктами керування проходять через проміжний сервер, який фільтрує їх і захищає від втручання зловмисників. Мікрокомп'ютер, що безпосередньо керує об'єктами, під'єднується до мережі через маршрутизатор і не отримує реальної IP-адреси. Унаслідок цього керувати ним ззовні неможливо інакше, ніж через консоль, яка слугує лише для встановлення програмного забезпечення, або через прикладну програму з вбудованими засобами захисту.

Рисунок 1.7 - Схема керування об'єктами з використанням сервера-посередника

Схема із сервером-посередником має кілька важливих переваг. Один такий сервер здатен одночасно обслуговувати багатьох користувачів, захищаючи їхні потоки даних, тому його можуть розгортати інтернет-провайдери, надаючи доступ до ресурсів за моделлю «програмне забезпечення як послуга». За підвищених вимог до захищеності користувач може встановити власний окремий або корпоративний сервер-посередник. Навіть у разі компрометації самого сервера-посередника відомості про керовані об'єкти зберігають свою цілісність, оскільки безпосереднє керування ними залишається недоступним. Єдиним помітним недоліком схеми є деяке зростання затримки сигналів порівняно з прямим керуванням, проте цей недолік є несуттєвим: швидкодія систем керування через Інтернет у будь-якому разі обмежена непередбачуваним часом доступу до мережі. Порівняння обох схем за основними критеріями наведено в таблиці 1.5.

Таблиця 1.5 - Порівняння схем прямого керування та керування через сервер-посередник

Критерій	**Пряме керування**	**Через сервер-посередник**
Потреба у виділеній IP-адресі для керованого вузла	Так	Ні
Можливість прямого керування вузлом ззовні	Є	Відсутня
Фільтрація трафіку від зловмисників	Відсутня	Виконує сервер-посередник
Ризик використання вузла для DdoS-атак	Високий	Низький

Критерій	Пряме керування	Через сервер-посередник
Обслуговування багатьох користувачів	Обмежене	Підтримується (SaaS)
Затримка передавання сигналів	Менша	Дещо більша
Вимоги до кваліфікованого адміністрування вузла	Високі	Низькі

Аналіз показує, що для систем, які працюють у загальнодоступній мережі, схема із сервером-посередником [14, 27, 28] забезпечує істотно вищий рівень захищеності за прийнятного зниження швидкодії. Саме її покладено в основу системи, що розробляється. Додатковою перевагою такого підходу є те, що невеликий обсяг даних керування дозволяє застосувати найдосконаліші криптографічні методи захисту каналу, аж до абсолютно стійкого шифру Вернама, обґрунтування та реалізацію якого розглянуто в наступному розділі.

1.9 Об'єкт дослідження: огляд наявної системи дистанційного керування

Об'єктом дослідження цієї роботи є система дистанційного керування CONPIN, побудована за схемою сервера-посередника. Вона дозволяє керувати вісьмома двійковими об'єктами, реалізованими у вигляді релейних виходів і під'єднаними до одноплатного комп'ютера Raspberry Pi через інтерфейс GPIO, з будь-якого пристрою, що має браузер і доступ до мережі Інтернет. Узагальнену структуру системи показано на рисунку 1.8.

Рисунок 1.8 - Узагальнена структура системи дистанційного керування CONPIN

Система складається з трьох програмних компонентів, рознесених на дві обчислювальні машини. На самому Raspberry Pi працює клієнтська програма CONPIN.js, яка керує контактами GPIO і періодично, з інтервалом близько десяти секунд, з'єднується із сервером-посередником, передаючи поточний стан об'єктів та отримуючи команди керування. На віддаленому сервері розгорнуто два сервіси: програму SOCKET.js, що приймає з'єднання від мікрокомп'ютера за протоколом TCP на порту 3000, та програму VYBIR.js, що обслуговує користувача за протоколом HTTP на порту 8000 і віддає вебсторінку керування CONPIN.html.

Обмін даними між серверними сервісами організовано через проміжні файли. Поточний стан об'єктів, отриманий від мікрокомп'ютера, зберігається у файлі STAN.txt, а команди керування, сформовані користувачем, записуються у файл TREB.txt. Стан і команди подаються у вигляді рядка з восьми символів, де кожна позиція відповідає одному об'єкту: символи «0» та «1» задають вимкнений та увімкнений стани, а символ «/» означає, що стан відповідного об'єкта залишається без змін. Такий компактний формат спрощує обмін і робить його стійким до помилок.

Вебінтерфейс CONPIN.html, що його віддає сервіс VYBIR.js, відображає вісім об'єктів у вигляді таблиці з індикаторами стану та кнопками керування. Користувач може увімкнути чи вимкнути кожен об'єкт окремо або всі одразу, а інтерфейс кожні десять секунд автоматично оновлює стан, надсилаючи запит до сервера. Завдяки адаптивній побудові він однаково зручний на персональному комп'ютері та на мобільному пристрої. Таким чином, користувач взаємодіє лише із сервером-посередником і ніколи не звертається до мікрокомп'ютера безпосередньо.

Повний цикл роботи системи має такий вигляд. Мікрокомп'ютер з'єднується із сервером-посередником і надсилає рядок поточного стану об'єктів, який сервіс SOCKET.js зберігає у файлі STAN.txt. Користувач через браузер відкриває вебсторінку, яку віддає сервіс VYBIR.js, бачить актуальний стан об'єктів і надсилає команду керування. Сервіс VYBIR.js записує бажаний стан у файл TREB.txt, звідки його зчитує SOCKET.js і передає мікрокомп'ютерові під час наступного з'єднання. Отримавши команду, мікрокомп'ютер змінює стан відповідних реле і повертає

оновлений стан, замикаючи цикл. Така побудова гарантує, що користувач і мікрокомп'ютер ніколи не взаємодіють напряму.

Вісім двійкових об'єктів є демонстраційною, але показовою конфігурацією. Двійковий характер об'єктів, кожен з яких має лише два стани, відповідає найпоширенішому випадку дистанційного керування, коли потрібно ввімкнути або вимкнути певне обладнання. Обрана кількість об'єктів зручна для наочної демонстрації та повного тестування, а закладені в систему принципи дозволяють за потреби збільшити її без зміни самої архітектури.

Описана система ґрунтується на рішенні, запропонованому в попередніх дослідженнях, де клієнтську частину було реалізовано із застосуванням бібліотеки onoff на операційній системі Ubuntu 20.10 та платформі Node.js версії 12. За час, що минув від першої реалізації, ці компоненти значною мірою застаріли: змінилися підходи до керування контактами GPIO в ядрі Linux, оновилися версії середовища виконання, з'явилися додаткові вимоги до захищеності та надійності серверного програмного забезпечення. Тому система потребує модернізації з переходом на сучасні версії програмних засобів.

Потреба в модернізації зумовлена кількома причинами. Бібліотеки керування контактами GPIO, використані в початковій реалізації, припинили підтримуватися й погано сумісні із сучасними ядрами Linux. Версія середовища виконання, на якій будувалася система, вийшла з підтримки, що унеможливлює отримання оновлень безпеки. Окрім того, до серверного програмного забезпечення висуваються вищі вимоги щодо стійкості до збоїв та автоматичного відновлення роботи. Усе це робить доцільним перенесення серверної частини на сучасну та захищену операційну систему із застосуванням актуальних версій програмних засобів, що є одним із завдань цієї роботи.

Систему розроблено зусиллями двох виконавців. Клієнтська частина, що працює безпосередньо на Raspberry Pi й керує апаратними виходами, є предметом окремої кваліфікаційної роботи. Ця робота присвячена серверній частині, до складу якої входять сервіси SOCKET.js і VYBIR.js, вебінтерфейс користувача, організація

обміну даними між компонентами, а також розгортання й супровід серверного програмного забезпечення на віддаленому сервері.

1.10 Постановка задачі

На підставі проведеного аналізу можна сформулювати задачу роботи. Необхідно розробити та модернізувати серверну частину захищеної системи дистанційного керування об'єктами на базі Raspberry Pi, побудованої за схемою сервера-посередника, забезпечивши її надійне й безпечне функціонування на сучасній програмній платформі. Серверна частина має виступати сполучною ланкою між користувачем, що працює через браузер, та віддаленим мікрокомп'ютером, який безпосередньо керує об'єктами.

До серверної частини висуваються такі функціональні вимоги:

1. приймати з'єднання від мікрокомп'ютера за протоколом TCP та отримувати від нього поточний стан об'єктів керування;

2. зберігати отриманий стан і передавати мікрокомп'ютерові команди керування, сформовані користувачем;

3. надавати користувачеві вебінтерфейс для відображення стану об'єктів та керування ними;

4. опрацьовувати команди користувача, що надходять за протоколом HTTP, і коректно узгоджувати їх із поточним станом системи;

5. забезпечувати взаємодію вебінтерфейсу із сервером, зокрема правильно обробляти міждоменні запити (CORS);

6. зберігати стан системи між сеансами та відновлювати його після перезапуску.

Окрім функціональних, до серверної частини висуваються такі нефункціональні вимоги: надійність, тобто здатність програмного забезпечення автоматично відновлювати роботу після збоїв та аварійних завершень; безпека, що передбачає неможливість прямого керування мікрокомп'ютером ззовні та підтримку захищеного обміну даними; сумісність із сучасним програмним середовищем, зокрема з актуальними версіями операційної системи сервера та

середовища виконання; багатокористувацькість, за якої один сервер здатен обслуговувати декілька незалежних клієнтських систем; а також невибагливість до ресурсів і портативність.

Для розв'язання поставленої задачі необхідно обґрунтувати вибір операційної системи сервера та середовища виконання, спроєктувати архітектуру взаємодії сервісів, розробити програмний код сервісів SOCKET.js і VYBIR.js та вебінтерфейсу, реалізувати засоби захищеного обміну даними між складовими системи, а також розгорнути серверну частину на обраній платформі, налаштувати автоматичний перезапуск і провести тестування. Послідовне виконання цих кроків розглянуто в наступних розділах роботи.

1.11 Огляд наявних платформ та узагальнення вимог

Перед остаточним формулюванням вимог доцільно оглянути наявні підходи до побудови систем дистанційного керування та готові платформи Інтернету речей. Це дозволяє врахувати їх сильні та слабкі сторони й обґрунтувати самостійне розроблення серверної частини.

Найпоширенішими є готові хмарні платформи, які надають великі постачальники хмарних послуг. Вони пропонують засоби під'єднання пристроїв, зберігання та оброблення даних, візуалізації й керування доступом, а також автоматичне масштабування під навантаження. Їх перевагою є швидкість розгортання та відсутність потреби самостійно підтримувати серверну інфраструктуру.

Водночас хмарні платформи мають суттєві недоліки для систем з підвищеними вимогами до безпеки. Дані користувача проходять через сервери стороннього постачальника, що створює додаткові ризики конфіденційності, а робота системи стає залежною від доступності та політики цього постачальника. Крім того, виникає прив'язка до конкретної платформи, через яку перенесення системи до іншого середовища стає складним і витратним.

Існують також відкриті платформи, які можна розгорнути на власному обладнанні. Вони усувають частину питань залежності від постачальника, проте

здебільшого є універсальними та надлишковими для невеликої системи керування кількома об'єктами: їх розгортання й супровід потребують значних зусиль, а велика кількість складових збільшує поверхню атак.

Спільною проблемою наявних рішень залишається узгодженість і сумісність. Через велику кількість виробників, протоколів і форматів даних об'єднання пристроїв різних постачальників в одну систему часто потребує додаткових зусиль, а відсутність єдиних підходів до безпеки ускладнює оцінку захищеності готових платформ. Самостійно розроблене рішення з чітко визначеними форматами обміну дозволяє уникнути цієї невизначеності в межах конкретної системи.

Порівняння основних підходів до побудови систем дистанційного керування наведено в таблиці 1.6.

Таблиця 1.6 - Порівняння підходів до побудови систем дистанційного керування

Підхід	**Контроль над даними**	**Залежність від постачальника**	**Захищеність вузла**
Пряме керування	Повний	Відсутня	Низька
Хмарна платформа	Обмежений	Висока	Середня
Відкрита платформа на власному сервері	Повний	Низька	Залежить від налаштування
Власний сервер-посередник	Повний	Відсутня	Висока

Узагальнену структуру типової хмарної платформи Інтернету речей показано на рисунку 1.9. Пристрої під'єднуються до хмарного брокера повідомлень, який передає дані до служб зберігання та оброблення, а доступ користувача забезпечується через вебзастосунок або програмний інтерфейс платформи.

Рисунок 1.9 - Узагальнена структура хмарної платформи Інтернету речей

З огляду на наведене порівняння, для системи, у якій важливими є повний контроль над даними, незалежність від зовнішніх сервісів та високий рівень захищеності керованого вузла за помірних витрат, найдоречнішим є самостійне розроблення серверної частини за схемою сервера-посередника. 40амец ей підхід покладено в основу роботи.

На підставі проведеного аналізу сформульовано вимоги до серверної частини, які узагальнено в таблиці 1.7. Функціональні вимоги визначають, що саме має виконувати система, а нефункціональні - яким якісним характеристикам вона повинна відповідати.

Таблиця 1.7 - Вимоги до серверної частини системи

Група вимог	**Зміст**
Функціональні	Приймання стану об'єктів від мікрокомп'ютера; зберігання поточного стану; передавання команд користувача; надання вебінтерфейсу; опрацювання міждоменних запитів; збереження стану між сеансами
Захищеність	Приховування керованого вузла; шифрування каналу обміну; узгодження ключів без їх передавання; обмеження доступу до сервера
Надійність	Безперервна робота; автоматичне відновлення після збоїв; стійкість до некоректних даних
Сумісність і розвиток	Робота в сучасному середовищі; підтримка кількох клієнтських систем; можливість розширення без зміни архітектури

Сформульовані вимоги є відправною точкою для обґрунтування та розроблення рішення, що розглядається в наступному розділі.

1.12 Висновки до розділу 1

У першому розділі проаналізовано предметну галузь та сформульовано задачу роботи. Розглянуто Інтернет речей як галузь, наведено його визначення, історію становлення та основні сфери застосування, а на основі статистичних даних показано стрімке зростання кількості підключених пристроїв.

Описано типову багаторівневу архітектуру систем Інтернету речей і охарактеризовано їх складові, серед яких особливу увагу приділено виконавчим механізмам, насамперед електромеханічним реле, оскільки саме ними керує розглянута система. Розглянуто призначення платформ Інтернету речей та обґрунтовано доцільність побудови серверної частини з компактних спеціалізованих сервісів. Проаналізовано протоколи й технології передавання даних та обґрунтовано вибір протоколів TCP і HTTP.

Порівняно мікроконтролери та мікрокомп'ютери, розглянуто архітектуру і моделі Raspberry Pi, призначення інтерфейсу GPIO та доступні операційні системи й мови програмування. Досліджено проблему інформаційної безпеки систем Інтернету речей, визначено основні групи загроз і чинники, що ускладнюють захист, та окреслено заходи протидії. Розглянуто криптографічні засоби захисту даних, що становлять підґрунтя для подальших рішень.

Порівняно дві схеми організації обміну даними і обґрунтовано вибір схеми із сервером-посередником як такої, що забезпечує істотно вищий рівень захищеності за прийнятного зниження швидкодії. Описано об'єкт дослідження, тобто наявну систему дистанційного керування CONPIN, її склад та принцип обміну даними, і визначено потребу в модернізації застарілих компонентів. Сформульовано задачу роботи разом із функціональними та нефункціональними вимогами до серверної частини, що становить основу для проєктування та реалізації, розглянутих у наступному розділі.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ТА РОЗРОБЛЕННЯ РІШЕННЯ

2.1 Загальна архітектура серверної частини

Серверну частину системи побудовано за схемою сервера-посередника, обґрунтованою в попередньому розділі. Вона є проміжною ланкою між мікрокомп'ютером Raspberry Pi, що безпосередньо керує об'єктами, та користувачем, який працює через браузер. На сервер покладено два принципово різні види взаємодії: обмін службовими даними з мікрокомп'ютером та обслуговування запитів користувача. Щоб розмежувати ці завдання, серверну частину поділено на два незалежні сервіси, кожен з яких відповідає лише за свій вид взаємодії.

Перший сервіс, SOCKET.js, є сервером протоколу TCP і взаємодіє з мікрокомп'ютером: приймає від нього поточний стан об'єктів та передає йому команди керування. Другий сервіс, VYBIR.js, є сервером протоколу HTTP і взаємодіє з користувачем: віддає вебсторінку керування, повідомляє поточний стан об'єктів та приймає команди. Обидва сервіси не звертаються один до одного напряму, а обмінюються даними через спільні файли. Загальну структуру серверної частини показано на рисунку 2.1.

Рисунок 2.1 - Детальна структура серверної частини системи

Поділ серверної частини на два сервіси має кілька переваг. По-перше, він відповідає принципу розділення відповідальності: кожен сервіс розв'язує одну чітко окреслену задачу, що спрощує його розроблення та супровід. По-друге, сервіси є незалежними, тому збій або перезапуск одного з них не порушує роботи іншого. По-третє, така побудова дозволяє за потреби розгортати сервіси на різних машинах або масштабувати їх окремо. Зв'язування сервісів через файли, а не через прямі мережеві з'єднання, додатково послаблює залежність між ними та забезпечує збереження стану системи навіть після перезапуску.

Розгортання складових системи відображає її архітектуру. Мікрокомп'ютер розташований у локальній мережі за маршрутизатором і самостійно ініціює з'єднання із сервером, тому не потребує власної адреси, доступної з Інтернету. Сервер-посередник є вузлом, відкритим до зовнішніх з'єднань: він приймає з'єднання від мікрокомп'ютера на одному порту та запити користувачів на іншому. Браузер користувача може перебувати в будь-якій точці мережі. Таким чином, єдиною ланкою, відкритою до зовнішніх з'єднань, залишається сервер-посередник, тоді як керований вузол лишається прихованим, що відповідає обраній моделі захисту.

Серверна частина складається з невеликого набору чітко визначених складових. Сервіс SOCKET.js відповідає за обмін з мікрокомп'ютером, сервіс VYBIR.js - за обслуговування користувача, вебсторінка CONPIN.html є інтерфейсом користувача, а файли STAN.txt і TREB.txt зберігають відповідно поточний та бажаний стан об'єктів. Невелика кількість складових, кожна з яких має єдине призначення, спрощує розуміння, розроблення й супровід системи.

Описана побудова відповідає схемі сервера-посередника, відомій з аналізу способів обміну даними в системах Інтернету речей. Сервер-посередник приймає на себе роль єдиної відкритої до мережі ланки, фільтрує потоки даних між користувачем та об'єктами і приховує керований вузол, що узгоджується з висновками першого розділу про переваги цієї схеми.

Така архітектура має й запас для розвитку. Оскільки кожна клієнтська система під'єднується до сервера самостійно й використовує власні файли стану,

один сервер-посередник здатний обслуговувати багато незалежних систем одночасно, а їх кількість обмежена насамперед ресурсами сервера. Це робить рішення придатним не лише для окремого користувача, а й для надання послуги дистанційного керування багатьом користувачам.

Поділ на незалежні сервіси полегшує й діагностику. Оскільки приймання даних від мікрокомп'ютера та взаємодія з користувачем виконуються різними програмами, проблему можна локалізувати в межах однієї з них, не аналізуючи систему загалом. Це скорочує час пошуку та усунення несправностей і знижує ризик внесення нових помилок під час супроводу.

Загальний сценарій використання системи має такий вигляд. Користувач відкриває у браузері сторінку керування й бачить поточний стан восьми об'єктів. Щоб змінити стан певного об'єкта, він натискає відповідну кнопку; інтерфейс надсилає команду серверу, той зберігає її, і під час наступного сеансу зв'язку мікрокомп'ютер отримує команду та перемикає відповідне реле. Оновлений стан повертається на сервер і відображається в інтерфейсі. З погляду користувача керування зводиться до простих дій у звичному вікні браузера, а вся складність обміну та захисту прихована в серверній частині.

2.2 Обґрунтування вибору операційної системи сервера

Оскільки сервер-посередник є відкритим до мережі Інтернет вузлом, через який проходять усі дані керування, до операційної системи сервера висуваються підвищені вимоги щодо захищеності. Серед поширених варіантів розглянуто дистрибутиви Linux загального призначення та операційну систему OpenBSD [15], що належить до родини BSD і відома пріоритетом безпеки.

Дистрибутиви Linux є універсальними та широко підтримуваними, проте їх типове налаштування орієнтоване на функціональність, а не на захищеність, тому належний рівень безпеки потребує додаткового конфігурування. На відміну від них, OpenBSD від початку проєктується з пріоритетом безпеки та коректності коду. Базова система містить лише необхідний мінімум увімкнених служб, що зменшує поверхню можливих атак, має інтегрований брандмауер pf і регулярно проходить

систематичний аудит вихідного коду. Порівняння обох варіантів за основними критеріями наведено в таблиці 2.1.

Таблиця 2.1 - Порівняння операційних систем для сервера-посередника

Критерій	**Типовий дистрибутив Linux**	**OpenBSD**
Пріоритет розробників	Універсальність і функціональність	Безпека та коректність коду
Захищеність за замовчуванням	Потребує додаткового налаштування	Висока, мінімум увімкнених служб
Вбудований брандмауер	nftables або iptables	pf, інтегрований у систему
Аудит базового коду	Залежить від спільноти	Систематичний аудит
Обсяг базової системи	Великий, багато пакетів	Компактний, мінімалістичний

З огляду на призначення сервера-посередника, для якого захищеність є визначальною вимогою, як операційну систему сервера обрано OpenBSD. Її мінімалістичність і невибагливість до ресурсів додатково узгоджуються з вимогою щодо портативності серверної частини, а наявність штатного брандмауера спрощує обмеження доступу до сервісів лише дозволеними напрямами.

Захищеність OpenBSD ґрунтується не на окремих засобах, а на сукупності вбудованих механізмів. Система дотримується принципу безпечних налаштувань за замовчуванням, за якого зайві служби вимкнено, а доступ обмежено від початку. Застосовуються механізми розмежування привілеїв та ізоляції процесів, що зменшують наслідки можливої компрометації окремого сервісу. Програми можуть заздалегідь обмежувати власні повноваження, оголошуючи лише ті системні виклики та файли, які їм справді потрібні, тож навіть у разі вразливості можливості зловмисника залишаються обмеженими.

Штатний брандмауер pf [16] дозволяє чітко обмежити доступ до сервера, відкривши лише потрібні порти: один для з'єднань від мікрокомп'ютера і один для запитів користувачів. Додатковою перевагою OpenBSD є відкритість і безкоштовність, а також невибагливість до апаратних ресурсів, завдяки чому сервер-посередник можна розгорнути навіть на недорогому обладнанні.

Сукупність цих властивостей робить OpenBSD обґрунтованим вибором для вузла, через який проходять усі дані керування.

Операційні системи загального призначення, орієнтовані на настільне використання, для цієї ролі не розглядалися: вони містять велику кількість зайвих для сервера компонентів, що збільшують поверхню атак і споживання ресурсів. Сервер-посередник, навпаки, потребує мінімального, передбачуваного й добре контрольованого оточення, якому найкраще відповідає саме OpenBSD.

Захищеність операційної системи підтримується не лише вихідною конфігурацією, а й моделлю її супроводу. Базова система та її компоненти розвиваються узгоджено й регулярно оновлюються, а виявлені вади усуваються централізовано, без потреби стежити за безліччю незалежних пакетів. Для сервера-посередника, який має працювати тривалий час без постійного нагляду, передбачуваність і узгодженість оновлень є не менш важливими, ніж самі захисні механізми, оскільки саме вони визначають, наскільки система залишатиметься захищеною з плином часу.

Захищеність OpenBSD спирається на низку вбудованих механізмів, які діють незалежно від налаштувань адміністратора. Серед них - заборона одночасного запису та виконання для ділянок пам'яті, випадкове розташування адрес під час запуску програм, обмеження можливостей процесів за принципом найменших привілеїв, а також ретельний аудит вихідного коду базової системи. Сукупність цих засобів суттєво ускладнює використання навіть наявних у програмах вад, що важливо для вузла, постійно доступного з мережі.

Не менш важливою є мінімальність базової системи. На відміну від універсальних дистрибутивів, у яких після встановлення працює багато служб, OpenBSD типово запускає лише необхідний мінімум, тому кількість потенційних точок входу для зловмисника від початку є невеликою. Для сервера-посередника, єдине призначення якого полягає в передаванні даних між користувачем і мікрокомп'ютером, така ощадливість є перевагою, а не обмеженням.

2.3 Обґрунтування вибору середовища виконання та мови програмування

Серверні сервіси є мережевими застосунками, які мають одночасно обслуговувати з'єднання від мікрокомп'ютера та запити користувачів. Для таких задач вирішальне значення має модель вводу-виводу середовища виконання. Розглянуто три варіанти: платформу Node.js [17] з мовою JavaScript, мову Python та мови C і C++.

Платформа Node.js використовує асинхронну, подієву модель вводу-виводу, за якої одна програма здатна обслуговувати багато з'єднань без створення окремого потоку для кожного з них. Це робить її особливо придатною для мережевих сервісів з невеликим обсягом обчислень, до яких належить і розроблювана система. Мова Python зручна для швидкого розроблення, проте її типова модель виконання є переважно синхронною. Мови C та C++ забезпечують найвищу продуктивність, але потребують значно більшого обсягу коду й ретельнішого керування ресурсами. Порівняння варіантів наведено в таблиці 2.2.

Таблиця 2.2 - Порівняння середовищ виконання та мов програмування

Критерій	**Node.js**	**Python**	**C / C++**
Модель вводу-виводу	Асинхронна, подієва	Переважно синхронна	Залежить від реалізації
Зручність для мережевих застосунків	Висока	Середня	Низька
Спільна мова з клієнтом і браузером	Так (JavaScript)	Ні	Ні
Швидкість розроблення	Висока	Висока	Низька
Продуктивність	Висока для мережевих задач	Середня	Найвища

Вирішальною перевагою Node.js для цієї системи є те, що мовою JavaScript написано як серверні сервіси, так і клієнтську програму на мікрокомп'ютері та вебінтерфейс у браузері. Завдяки цьому засоби захищеного обміну даними реалізуються однаковим кодом на всіх боках, що виключає розбіжності між реалізаціями та зменшує ймовірність помилок. З огляду на наведені міркування як середовище виконання серверних сервісів обрано Node.js.

Для реалізації сервісів достатньо стандартних засобів платформи Node.js: вбудований модуль роботи з мережею забезпечує створення сервера протоколу TCP, модуль обробки запитів - сервера протоколу HTTP, а модуль файлової системи - обмін через спільні файли. Відмова від зайвих сторонніх бібліотек є свідомим рішенням: вона зменшує обсяг коду, спрощує супровід і, що особливо важливо для системи з пріоритетом безпеки, скорочує кількість залежностей, кожна з яких могла б стати джерелом вразливостей.

В основі Node.js лежить цикл подій, який обробляє операції вводу-виводу в міру їх готовності, не блокуючи виконання програми очікуванням. Завдяки цьому один процес здатний одночасно підтримувати з'єднання з мікрокомп'ютером та обслуговувати запити кількох користувачів, витрачаючи мінімум ресурсів. Для системи, де обсяг обчислень незначний, а основну частину часу займає очікування мережевих подій, така модель є найдоречнішою.

Важливою є й доступність платформи Node.js та її широка підтримка. Вона є відкритою, безкоштовною і працює на більшості операційних систем, зокрема на обраній OpenBSD, що спрощує розгортання. Наявність єдиного середовища виконання для серверних сервісів і клієнтської програми мікрокомп'ютера означає, що для супроводу всієї системи достатньо володіння однією мовою програмування, а це знижує вимоги до кваліфікації обслуговувального персоналу та зменшує ймовірність помилок під час внесення змін.

Асинхронна модель оброблення подій добре відповідає характеру навантаження сервера-посередника. Основну частину часу сервіси проводять в очікуванні мережевих подій, нового з'єднання від мікрокомп'ютера або запиту від користувача, а не у виконанні обчислень. Завдяки неблокувальному вводу-виводу один процес здатний обслуговувати багато таких подій без створення окремого потоку для кожної, що економно використовує ресурси сервера й спрощує реалізацію.

2.4 Обґрунтування засобу керування процесами

Однією з нефункціональних вимог до серверної частини є надійність, тобто здатність автоматично відновлювати роботу після збоїв. Запуск сервісів безпосередньо з командного рядка цієї вимоги не задовольняє, оскільки в разі аварійного завершення сервіс залишається непрацездатним до ручного втручання. Тому для керування процесами обрано менеджер PM2.

Менеджер процесів PM2 запускає сервіси у фоновому режимі, безперервно стежить за їх станом і автоматично перезапускає в разі аварійного завершення. Він також забезпечує автоматичний запуск сервісів під час старту операційної системи, веде журнали роботи та дозволяє керувати сервісами єдиними командами. Застосування PM2 до обох сервісів, SOCKET.js і VYBIR.js, забезпечує безперервність роботи серверної частини без постійного нагляду адміністратора.

Налаштування PM2 задають у вигляді опису сервісів, який містить імена програм, шляхи до них та політику перезапуску. Менеджер інтегрується з механізмом автозапуску операційної системи, тому після перезавантаження сервера сервіси піднімаються автоматично, без втручання людини. Журнали роботи, які веде PM2, спрощують виявлення та усунення несправностей, а єдині команди керування дозволяють зупиняти, перезапускати й переглядати стан окремих сервісів, не зачіпаючи решти системи.

Менеджер процесів також спрощує спостереження за станом сервісів. Він показує, скільки часу кожен сервіс працює без перезапуску, скільки ресурсів споживає та скільки разів перезапускався, що дозволяє своєчасно помітити нестабільну роботу. Для системи, яка має функціонувати тривалий час без втручання, така прозорість стану є важливою складовою надійності.

Менеджер процесів також спрощує процедуру оновлення системи. Щоб застосувати зміни в коді сервісу, достатньо перезапустити його однією командою, причому решта сервісів продовжує працювати без перерви. Це дозволяє супроводжувати систему поступово, не зупиняючи її роботи повністю, що важливо для засобу, який має бути доступним постійно.

Без засобу керування процесами безперервна робота сервера потребувала б постійного нагляду, що суперечить призначенню системи. PM2 усуває цю потребу,

перетворюючи сервіси на самостійні фонові служби, які відновлюються після збоїв і запускаються разом із операційною системою.

2.5 Проєктування обміну даними між складовими

Обмін даними в системі організовано у двох напрямах. Зовнішній обмін відбувається мережею: між мікрокомп'ютером і сервісом SOCKET.js за протоколом TCP, а між браузером користувача та сервісом VYBIR.js за протоколом HTTP. Внутрішній обмін між двома серверними сервісами відбувається через спільні файли, що зберігаються на сервері.

Стан об'єктів та команди керування подаються у вигляді рядка з восьми символів, де кожна позиція відповідає одному з восьми об'єктів. У рядку стану символ «1» означає увімкнений об'єкт, а символ «0» - вимкнений. У рядку команди додатково використовують символ «/», що означає відсутність змін для відповідного об'єкта, завдяки чому одна команда може змінювати стан лише окремих об'єктів, не зачіпаючи інших. Відповідність об'єктів керування, контактів GPIO мікрокомп'ютера та позицій у рядку наведено в таблиці 2.3.

Таблиця 2.3 - Відповідність об'єктів керування, контактів GPIO та позицій у рядку стану

Об'єкт (реле)	Контакт GPIO (BCM)	Позиція у рядку стану
Реле 1	4	1
Реле 2	17	2
Реле 3	18	3
Реле 4	22	4
Реле 5	23	5
Реле 6	24	6
Реле 7	25	7
Реле 8	27	8

Для внутрішнього обміну використовують два файли. У файлі STAN.txt зберігається поточний стан об'єктів, отриманий від мікрокомп'ютера, а у файлі TREB.txt - бажаний стан, сформований за командами користувача. Сервіс

SOCKET.js записує дані у STAN.txt і зчитує команди з TREB.txt, тоді як сервіс VYBIR.js зчитує STAN.txt і записує команди у TREB.txt. Послідовність обміну даними між усіма складовими системи показано на рисунку 2.2.

Рисунок 2.2 - Послідовність обміну даними між складовими системи

Використання файлів для внутрішнього обміну обрано свідомо. По-перше, воно повністю відокремлює сервіси один від одного: вони не потребують прямого з'єднання і можуть працювати та перезапускатися незалежно. По-друге, файли забезпечують збереження стану системи: навіть після перезапуску сервісів або сервера останній відомий стан об'єктів і остання команда не втрачаються. По-третє, такий підхід є простим і прозорим, що полегшує налагодження та супровід системи.

Взаємодія складових має періодичний характер. Мікрокомп'ютер з'єднується із сервером приблизно кожні десять секунд, передаючи стан і забираючи команду, а вебінтерфейс із таким самим інтервалом запитує стан у сервера. Через це між дією користувача та її виконанням може минути до кількох секунд, що цілком прийнятно для керування об'єктами і узгоджується з висновком про несуттєвість затримок у системах такого класу. Періодична, а не постійна взаємодія також зменшує навантаження на мережу та на сервер.

Оскільки доступ до спільних файлів відбувається в різні моменти часу, узгодженість даних забезпечується простим правилом: дійсним вважається останній записаний стан. Команди записуються та зчитуються цілим рядком, що унеможливлює часткове застосування команди. Дані подаються у текстовому вигляді, тому їх можна переглянути й перевірити звичайними засобами, що полегшує діагностику системи.

Вибір компактного восьмисимвольного формату зумовлений кількома міркуваннями. Він прямо відповідає кількості керованих об'єктів, легко формується та перевіряється, а його фіксована довжина спрощує контроль коректності даних. Водночас формат залишається розширюваним: щоб збільшити кількість об'єктів, достатньо подовжити рядок, не змінюючи самого принципу обміну.

Можливі й інші способи внутрішнього обміну, зокрема через мережеві з'єднання між сервісами або через спільну базу даних. Проте для двох сервісів, що працюють на одному сервері й обмінюються лише коротким рядком стану, такі засоби були б надлишковими: вони ускладнили б систему, додали б залежності та нові точки відмови. Обмін через файли натомість використовує штатні можливості операційної системи, не потребує додаткового програмного забезпечення й добре узгоджується з вимогами простоти та надійності.

Формат повідомлення стану заслуговує на окремий розгляд. Стан усіх восьми об'єктів подається рядком з восьми символів, у якому кожна позиція жорстко відповідає певному об'єкту, а значення символу однозначно задає його стан. Для команд додатково передбачено символ незмінності, що дозволяє змінювати лише окремі об'єкти, не зачіпаючи решти. Така будова робить повідомлення компактним, легким для перевірки й однаково зрозумілим для всіх складових системи, а також спрощує його розширення на більшу кількість об'єктів.

2.6 Проєктування сервісу приймання даних SOCKET.js

Сервіс SOCKET.js реалізує сервер протоколу TCP, що очікує з'єднань на порту 3000. Саме до нього періодично під'єднується мікрокомп'ютер, щоб передати

поточний стан об'єктів та отримати команди керування. Сервіс має забезпечувати коректне приймання даних, їх перевірку та своєчасну передачу команд у відповідь.

Алгоритм роботи сервісу полягає в такому. Після встановлення з'єднання сервіс отримує від мікрокомп'ютера рядок стану та перевіряє його коректність: рядок має складатися рівно з восьми символів, кожен з яких є «0» або «1». Якщо перевірку пройдено, отриманий стан записується у файл STAN.txt, що оновлює відомості про об'єкти. Після цього сервіс зчитує файл TREB.txt і надсилає мікрокомп'ютерові поточну команду керування, а потім закриває з'єднання до наступного сеансу. Узагальнений алгоритм роботи сервісу показано на рисунку 2.3.

Рисунок 2.3 - Алгоритм роботи сервісу SOCKET.js

Перевірка формату отриманих даних є важливою складовою сервісу. Вона захищає систему від хибних або зловмисно сформованих повідомлень, які могли б порушити її роботу або призвести до запису некоректного стану. Дані, що не відповідають очікуваному формату, відхиляються й не впливають на збережений

стан об'єктів. Таким чином, сервіс SOCKET.js не лише забезпечує обмін з мікрокомп'ютером, а й виконує первинний контроль цілісності даних.

Послідовність дій сервісу під час одного сеансу зв'язку можна подати у вигляді таких кроків:

1. очікування та приймання з'єднання від мікрокомп'ютера на порту 3000;

2. отримання рядка стану та перевірка його довжини й допустимості символів;

3. у разі успішної перевірки запис отриманого стану у файл STAN.txt;

4. зчитування поточної команди керування з файлу TREB.txt;

5. надсилання команди мікрокомп'ютерові у відповідь;

6. закриття з'єднання та очікування наступного сеансу.

Окремо передбачено опрацювання нештатних ситуацій. Якщо з'єднання обривається або дані надходять у неповному чи спотвореному вигляді, сервіс не змінює збереженого стану й коректно завершує сеанс, готуючись прийняти наступне з'єднання. Завдяки цьому збій окремого сеансу не впливає на загальну працездатність системи.

Вибір протоколу TCP для цього обміну зумовлений потребою в гарантованому та впорядкованому доставлянні даних. Фіксована довжина повідомлення у вісім символів спрощує визначення меж даних: сервіс очікує рівно стільки символів і відкидає все, що не відповідає цьому формату. Такий простий спосіб розмежування повідомлень є достатнім для невеликих обсягів даних керування й не потребує складніших механізмів кадрування.

Послідовність роботи сервісу обміну є циклічною. Отримавши від мікрокомп'ютера повідомлення з поточним станом об'єктів, сервіс перевіряє його, зберігає та надсилає у відповідь чинну команду керування. Якщо нової команди не надходило, повертається ознака незмінності, і мікрокомп'ютер зберігає поточний стан. Такий обмін повторюється з визначеною періодичністю, що забезпечує постійну узгодженість стану між сервером і керованим вузлом.

Модель з'єднання обрано так, щоб ініціатором завжди був мікрокомп'ютер. Він періодично встановлює короткочасне з'єднання, виконує обмін і розриває його,

а не утримує постійний канал. Це узгоджується зі схемою сервера-посередника, за якої керований вузол не має зовнішньої адреси й не може приймати вхідні з'єднання, а також зменшує споживання ресурсів сервера, оскільки відкриті з'єднання існують лише на час обміну.

2.7 Проєктування сервісу взаємодії з користувачем VYBIR.js

Сервіс VYBIR.js реалізує сервер протоколу HTTP [19], що очікує запитів на порту 8000 і обслуговує користувача. Він виконує три основні функції: віддає вебсторінку керування, повідомляє поточний стан об'єктів та приймає команди користувача. Кожній функції відповідає окремий маршрут запиту, перелік яких наведено в таблиці 2.4.

Таблиця 2.4 - Маршрути запитів сервісу VYBIR.js

Маршрут	Метод	Призначення
/	GET	Віддає головну сторінку вебінтерфейсу керування
/STATE	GET	Повертає поточний стан об'єктів із файлу STAN.txt
/command	POST	Записує бажаний стан об'єктів у файл TREB.txt
/QS1 ... /QS8	GET	Застарілі команди перемикання окремих об'єктів

Опрацювання запитів відбувається так. У відповідь на запит головної сторінки сервіс віддає файл вебінтерфейсу. На запит стану сервіс зчитує файл STAN.txt і повертає його вміст, що дозволяє вебінтерфейсу відображати актуальний стан об'єктів. Отримавши команду користувача, сервіс зчитує її тіло, формує відповідний рядок бажаного стану та записує його у файл TREB.txt, звідки команду згодом забере сервіс SOCKET.js. Застарілі маршрути перемикання окремих об'єктів збережено для сумісності з попередніми версіями системи. Узагальнений алгоритм опрацювання запитів показано на рисунку 2.4.

Рисунок 2.4 - Алгоритм опрацювання запитів сервісом VYBIR.js

Окремої уваги потребує опрацювання міждоменних запитів. Оскільки вебсторінка та програмний інтерфейс сервера можуть розглядатися браузером як різні джерела, браузер застосовує політику обмеження міждоменних запитів [20]. Щоб взаємодія вебінтерфейсу із сервером була можливою, сервіс VYBIR.js додає до відповідей спеціальні заголовки, які дозволяють такі запити, та коректно опрацьовує попередні запити узгодження, що їх браузер надсилає перед основним запитом. Без цього механізму команди користувача не досягали б сервера, тому його реалізація є обов'язковою умовою працездатності інтерфейсу.

Розмежування функцій сервісу за окремими маршрутами відповідає усталеній практиці побудови вебінтерфейсів програмування. Запит стану та надсилання команди розділено не лише за адресою, а й за методом передавання: стан запитується безпечним методом читання, а команда надсилається методом, призначеним для зміни даних. Такий поділ робить поведінку сервісу передбачуваною, спрощує його перевірку й полегшує подальше розширення переліку операцій без порушення наявних.

Під час віддавання вебсторінки сервіс визначає тип вмісту та повертає браузерові відповідні дані, забезпечуючи правильне відображення інтерфейсу. Статичні ресурси віддаються безпосередньо, без додаткової обробки, що спрощує сервіс і пришвидшує завантаження сторінки.

Команда користувача надходить у тілі запиту у вигляді рядка бажаного стану. Сервіс зчитує це тіло повністю, перевіряє його та записує у файл TREB.txt у тому самому восьмисимвольному форматі, що використовується для обміну з мікрокомп'ютером. Уніфікований формат на всіх етапах обміну спрощує систему й унеможливлює неоднозначне тлумачення команд різними складовими.

Протокол HTTP за своєю природою не зберігає стану між запитами, тому поточний стан об'єктів винесено за межі сервісу, у спільний файл. Завдяки цьому будь-який запит стану обслуговується незалежно від попередніх, а сам сервіс не мусить утримувати відомості про окремих користувачів. Це спрощує його реалізацію та дозволяє однаково обслуговувати довільну кількість одночасних звернень.

Сервіс розраховано на стійку роботу за некоректних або неповних запитів. На невідомі маршрути та помилкові запити він повертає відповідні повідомлення, не перериваючи роботи, а помилки під час читання чи запису файлів опрацьовуються так, щоб одиничний збій не призводив до зупинки сервісу. Така побудова відповідає вимозі надійності та узгоджується із застосуванням менеджера процесів PM2.

2.8 Проєктування вебінтерфейсу користувача

Вебінтерфейс є єдиним засобом, через який користувач взаємодіє із системою, тому його зручність і наочність мають важливе значення. Інтерфейс реалізовано у вигляді єдиної вебсторінки, яку віддає сервіс VYBIR.js і яка не потребує встановлення додаткового програмного забезпечення: для роботи з системою достатньо браузера.

Основу інтерфейсу становить таблиця восьми об'єктів керування. Для кожного об'єкта відображають його номер, відповідний контакт GPIO, опис,

поточний стан та елемент керування. Поточний стан показано наочним індикатором, а перемикання здійснюється окремою для кожного об'єкта кнопкою. Додатково передбачено кнопки одночасного ввімкнення та вимкнення всіх об'єктів, що пришвидшує типові дії. Зовнішній вигляд інтерфейсу показано на рисунку 2.5.

Рисунок 2.5 - Інтерфейс користувача для керування об'єктами

Інтерфейс працює в такий спосіб. Кожні десять секунд він автоматично надсилає серверові запит стану й оновлює індикатори об'єктів відповідно до отриманих даних, завдяки чому користувач завжди бачить актуальну картину, навіть якщо стан змінив інший користувач. Коли користувач натискає кнопку керування, інтерфейс формує команду й надсилає її серверові, після чого зміна стану відображається під час наступного оновлення. Сторінку побудовано адаптивно, тому вона однаково зручна на персональному комп'ютері та на мобільному пристрої, а темне оформлення зменшує навантаження на зір під час тривалої роботи.

Відповідність елементів інтерфейсу об'єктам керування задано окремою таблицею відповідності, що пов'язує кожен рядок інтерфейсу з номером об'єкта, контактом GPIO та позицією у рядку стану. Завдяки цьому зміна опису об'єктів або їх порядку не потребує переробки логіки інтерфейсу, а зводиться до оновлення цієї таблиці. Кожен стовпець таблиці інтерфейсу має чітке призначення: номер об'єкта, відповідний контакт, текстовий опис, наочний індикатор стану та елемент керування.

Обмін інтерфейсу із сервером реалізовано засобами браузера без перезавантаження сторінки. Під час періодичного оновлення інтерфейс надсилає запит стану, отримує відповідь і перемальовує індикатори, а під час дії користувача надсилає команду й очікує її відображення під час наступного оновлення. Така побудова забезпечує плавну роботу та зрозумілий зворотний зв'язок: користувач бачить результат своїх дій і поточний стан усіх об'єктів у єдиному вікні.

Вебінтерфейс реалізовано без застосування зовнішніх бібліотек і засобів складання, лише штатними можливостями браузера. Такий підхід обрано свідомо: він робить сторінку самодостатньою, прискорює її завантаження та усуває залежність від стороннього коду, що узгоджується із загальним принципом мінімізації залежностей, прийнятим для всієї системи.

Періодичне оновлення стану реалізовано опитуванням сервера через однакові проміжки часу. Такий підхід простіший за встановлення постійного з'єднання й цілком достатній для керування, де затримка в кілька секунд є неістотною. Інтерфейс також відображає наявність зв'язку із сервером, тому користувач завжди бачить, чи є його дії результативними, що підвищує зручність і передбачуваність роботи.

Інтерфейс передбачає наочний зворотний зв'язок про стан об'єктів та результат дій користувача. Стан кожного об'єкта позначено кольоровим індикатором, що дозволяє швидко оцінити загальну картину, а групові кнопки ввімкнення та вимкнення всіх об'єктів пришвидшують типові операції. Оскільки інтерфейс не потребує встановлення додатків і працює в будь-якому сучасному

браузері, він однаково доступний з персонального комп'ютера та з мобільного пристрою.

2.9 Розроблення засобів захищеного обміну даними

Захист каналу обміну даними є центральною вимогою до системи, оскільки саме каналом між складовими передаються відомості про стан об'єктів і команди керування. Невеликий обсяг цих даних є важливою особливістю: він знімає обмеження на швидкодію криптографічних перетворень і дозволяє застосувати найстійкіші з відомих методів захисту, які за великих обсягів даних були б непрактичними.

Для шифрування каналу обрано шифр Вернама [3, 4], відомий також як одноразовий блокнот. У ньому кожен символ повідомлення поєднується операцією додавання за модулем з відповідним символом ключа, причому ключ є справді випадковою послідовністю тієї самої довжини, що й повідомлення, і використовується лише один раз. Клод Шеннон математично довів, що за дотримання цих умов шифр забезпечує досконалу секретність: зашифроване повідомлення не містить жодної інформації про вихідне, а отже, не може бути розкрите навіть за необмежених обчислювальних ресурсів. Саме абсолютна стійкість робить цей шифр привабливим для захисту команд керування фізичними об'єктами.

Головною практичною складністю застосування шифру Вернама є потреба забезпечити обидві сторони обміну однаковою випадковою ключовою послідовністю, не передаючи її відкритим каналом. Цю задачу розв'язано за допомогою алгоритму узгодження ключів Діффі-Геллмана. Він дозволяє двом сторонам, що обмінюються лише відкритими повідомленнями, отримати спільне секретне значення, яке не передається мережею і яке стороннім обчислити практично неможливо. Стійкість алгоритму ґрунтується на складності задачі дискретного логарифмування у скінченному полі.

Вибір поля Галуа як середовища обчислень зумовлений його математичними властивостями. Скінченне поле [21] забезпечує замкненість операцій: результат

додавання чи множення його елементів завжди залишається елементом того самого поля, що робить обчислення коректними й передбачуваними. Поле характеристики два особливо зручне для програмної реалізації, оскільки його елементи подаються послідовностями бітів, додавання зводиться до операції виключного «або», а множення - до множення многочленів зі зведенням за твірним многочленом поля. Завдяки цьому всі перетворення виконуються цілочисловими побітовими операціями без втрати точності.

Стійкість узгодження ключа спирається на складність задачі дискретного логарифмування у вибраному полі. Маючи відкрите значення та параметри поля, обчислити секретний показник степеня практично неможливо, оскільки відомі алгоритми потребують для цього часу, що зростає надзвичайно швидко зі збільшенням розміру поля. Поле порядку два в степені 503 містить настільки велику кількість елементів, що повний перебір можливих ключів є нездійсненним за будь-яких реалістичних обчислювальних ресурсів, а саме узгодження завдяки оптимізації обчислень залишається швидким.

Обраний підхід особливо доречний саме для систем керування з малим обсягом даних. Класичні захищені протоколи передавання, розраховані на великі потоки даних, додають помітні витрати на встановлення з'єднання та узгодження параметрів, що для поодиноких коротких команд є невиправданим. Натомість поєднання простого симетричного шифру з одноразовим ключем дає максимальну стійкість за мінімальних накладних витрат, що якнайкраще відповідає характеру переданих у системі даних.

Слід також зазначити, що обраний підхід не залежить від конкретного обладнання чи каналу зв'язку. Оскільки криптографічні перетворення виконуються програмно й однаково на обох боках, систему можна перенести на інший мікрокомп'ютер або змінити спосіб під'єднання, не змінюючи самих засобів захисту. Це робить рішення гнучким і придатним до використання в різних умовах.

У розробленій системі узгодження ключів виконують над полем Галуа порядку 2 у степені 503, а параметрами перетворень слугують безпечні прості числа 503, 563, 587 та 719. Такий вибір забезпечує надзвичайно великий простір

можливих ключів, через що відновлення спільного секрету повним перебором є обчислювально нездійсненним. Отримане спільне секретне значення використовують для формування ключової послідовності шифру Вернама. Узагальнену схему узгодження спільного ключа та подальшого шифрування каналу показано на рисунку 2.6.

Рисунок 2.6 - Схема узгодження спільного ключа та шифрування каналу

Процедуру узгодження спільного ключа за алгоритмом Діффі-Геллмана [5] у системі можна подати у вигляді такої послідовності дій:

1. сторони заздалегідь домовляються про спільні відкриті параметри перетворень, що ґрунтуються на полі Галуа та обраних безпечних простих числах;

2. кожна сторона генерує власне секретне число, яке нікому не передає;

3. на основі свого секрету та спільних параметрів кожна сторона обчислює відкрите значення й надсилає його іншій стороні відкритим каналом;

4. отримавши відкрите значення співрозмовника, кожна сторона поєднує його зі своїм секретом і обчислює спільне секретне значення;

5. завдяки властивостям обраних перетворень обидві сторони отримують однакове секретне значення, тоді як стороння особа, яка бачила лише відкриті повідомлення, обчислити його не може.

Принцип роботи алгоритму зручно пояснити на спрощеному прикладі. Перша сторона обирає секретне число й обчислює на його основі відкрите значення, яке надсилає другій стороні; друга сторона робить те саме зі своїм секретом. Після обміну відкритими значеннями кожна сторона поєднує отримане значення зі своїм секретом і дістає той самий результат. Стороння особа, яка перехопила лише відкриті значення та спільні параметри, для відновлення секрету змушена була б розв'язати задачу дискретного логарифмування, що за великих чисел є практично нездійсненним.

Ключову послідовність використовують лише один раз, що є обов'язковою умовою стійкості шифру Вернама. За потреби сторони можуть повторно виконати узгодження й отримати нову послідовність, жодного разу не передаючи ключ у відкритому вигляді. Завдяки цьому навіть тривала робота системи не послаблює захисту каналу.

Обране рішення поєднує переваги обох видів криптографії. Симетричний шифр Вернама забезпечує швидке та абсолютно стійке шифрування самих даних, тоді як узгодження ключів за принципами асиметричної криптографії знімає проблему безпечного передавання спільного ключа. Таке поєднання є типовим для сучасних захищених систем і добре узгоджується з умовами цієї задачі, де обсяг даних малий, а вимоги до стійкості високі.

Слід також розрізняти властивості захисту, які забезпечує система. Шифрування каналу гарантує конфіденційність переданих даних, тобто неможливість їх прочитання сторонньою особою. Цілісність даних додатково підтримується перевіркою формату на боці сервісів, яка відхиляє повідомлення, що не відповідають очікуваній структурі. Поєднання шифрування з контролем формату ускладнює як приховане прочитання, так і непомітну підміну даних у каналі.

Важливою перевагою обраного рішення є те, що всі криптографічні перетворення реалізовано однаковим кодом мовою JavaScript як на боці сервера, так і на боці мікрокомп'ютера. Це гарантує, що обидві сторони виконують ідентичні обчислення й отримують узгоджені результати, а також спрощує супровід системи,

оскільки криптографічну частину потрібно розробляти та перевіряти лише один раз. Поєднання абсолютно стійкого шифру Вернама з надійним узгодженням ключів за алгоритмом Діффі-Геллмана забезпечує захист каналу обміну, що відповідає сформульованим вимогам до системи.

2.10 Аналіз захищеності розробленого рішення

Розроблені проєктні рішення доцільно оцінити з погляду протидії основним загрозам, визначеним у першому розділі. Відповідність між загрозами та засобами захисту, передбаченими в системі, наведено в таблиці 2.5.

Таблиця 2.5 - Загрози інформаційній безпеці та засоби протидії в системі

Загроза	Засіб протидії в системі
Перехоплення даних у каналі	Шифрування каналу абсолютно стійким шифром Вернама
Підміна команд або стану	Перевірка формату даних та шифрування каналу
Несанкціоноване пряме керування вузлом	Схема сервера-посередника: керований вузол не має реальної IP-адреси
Атаки на відмову в обслуговуванні	Прихованість керованого вузла та обмеження доступу брандмауером pf
Компрометація серверного коду	Мінімізація служб і залежностей, захищена ОС OpenBSD
Аварійне завершення сервісів	Автоматичний перезапуск засобами PM2

Наведений аналіз показує, що розроблене рішення системно протидіє основним загрозам інформаційної безпеки. Захист реалізовано на кількох рівнях: архітектурному, через приховування керованого вузла за сервером-посередником; криптографічному, через шифрування каналу та узгодження ключів без їх передавання; системному, через захищену операційну систему та обмеження доступу брандмауером; і експлуатаційному, через автоматичне відновлення роботи сервісів. Поєднання цих рівнів забезпечує комплексний захист, що відповідає вимогам, сформульованим у постановці задачі.

Водночас слід зазначити межі застосовності рішення. Абсолютна стійкість шифру Вернама зберігається лише за умови справді випадкової та одноразової

ключової послідовності, тому якість генерування ключів є критичною для безпеки системи. Захист від підміни сторони обміну значною мірою спирається на прихованість керованого вузла, а не на окрему взаємну автентифікацію, що є прийнятним для розглянутого класу систем, проте має враховуватися за підвищених вимог до безпеки.

Захищеність системи є властивістю не окремого засобу, а їх узгодженої сукупності. Послаблення будь-якого рівня, наприклад використання неякісного джерела випадковості або відкриття зайвих портів на сервері, здатне знизити загальний рівень захисту незалежно від стійкості решти механізмів. Тому під час експлуатації системи важливо зберігати цілісність усіх передбачених заходів, а не покладатися лише на криптографічний захист каналу.

Доцільно співвіднести передбачені заходи з типовими загрозами каналу обміну. Перехоплення даних знешкоджується шифруванням, за якого навіть повністю перехоплений потік не розкриває корисної інформації. Спроба підміни даних виявляється завдяки строгій перевірці формату повідомлень, через яку довільно змінені дані відхиляються. Несанкціонований доступ до керованого вузла унеможливлюється тим, що той не має власної адреси в мережі Інтернет і з'єднується лише із сервером-посередником. Атаки на доступність частково послаблюються обмеженням відкритих портів та автоматичним відновленням роботи сервісів.

Загалом поєднання розглянутих заходів утворює багаторівневий захист, за якого подолання одного рівня не призводить до повної компрометації системи. Саме такий підхід вважається доброю практикою побудови захищених систем, оскільки він не покладається на жоден окремий механізм як на єдину лінію оборони.

2.11 Проєктування структури даних і протоколу обміну

Окремим етапом проєктування є визначення структур даних, через які взаємодіють складові системи, та порядку їх обміну. Від простоти й однозначності цих структур значною мірою залежать надійність і зрозумілість усієї системи.

Для обміну між сервісами використано два проміжні файли. У файлі стану зберігається поточний стан об'єктів, що надходить від мікрокомп'ютера, а у файлі команд - бажаний стан, який задає користувач. Призначення файлів та формат даних узагальнено в таблиці 2.6.

Таблиця 2.6 - Проміжні файли обміну та формат даних

Файл	Напрям	Вміст
STAN.txt	Від мікрокомп'ютера до сервера	Рядок з восьми символів «0» або «1» - поточний стан восьми об'єктів
TREB.txt	Від сервера до мікрокомп'ютера	Рядок з восьми символів «0», «1» або «/» - бажаний стан або ознака незмінності

Стан і команда подаються рядком фіксованої довжини у вісім символів, кожен з яких відповідає окремому об'єкту. Символ «1» означає ввімкнений стан, «0» - вимкнений, а символ «/», що використовується лише в командах, означає збереження поточного стану відповідного об'єкта. Структуру повідомлень показано на рисунку 2.7.

Рисунок 2.7 - Структура повідомлень стану та команди

Такий формат є компактним і однозначним: його легко перевірити на коректність, передати каналом зв'язку та розширити на більшу кількість об'єктів простим подовженням рядка. Жорстка відповідність позицій об'єктам

унеможливлює неоднозначне тлумачення повідомлень різними складовими системи.

Порядок обміну між складовими визначає послідовність кроків від дії користувача до фізичного перемикання об'єкта та зворотного відображення стану. Цю послідовність показано на рисунку 2.8.

Рисунок 2.8 - Послідовність взаємодії складових системи

Користувач у вебінтерфейсі формує команду, яка надсилається вебсервісу й записується у файл команд. Сервіс обміну, помітивши зміну файлу, передає команду мікрокомп'ютерові під час чергового з'єднання. Мікрокомп'ютер застосовує команду до об'єктів і повертає оновлений стан, який сервіс обміну зберігає у файлі стану. Вебсервіс під час наступного запиту повертає цей стан інтерфейсу, де він відображається користувачеві. Завдяки проміжним файлам складові залишаються слабко пов'язаними й можуть працювати у власному темпі.

2.12 Проєктування підсистеми криптографічного захисту

Підсистема криптографічного захисту відповідає за конфіденційність і цілісність даних, що передаються між сервером і мікрокомп'ютером. Її проєктування охоплює вибір криптографічних примітивів, визначення параметрів та порядку узгодження ключа.

Основою захисту обрано абсолютно стійкий шифр Вернама, для якого потрібна спільна випадкова послідовність завдовжки з повідомлення. Щоб сторони могли отримати таку послідовність, не передаючи її каналом зв'язку, застосовано узгодження ключа за алгоритмом Діффі-Геллмана над полем Галуа. Загальну схему узгодження показано на рисунку 2.9.

Рисунок 2.9 - Схема узгодження спільного ключа за алгоритмом Діффі-Геллмана

Обчислення виконуються над полем Галуа порядку два в степені 503. Кожна сторона генерує власну випадкову послідовність, підносить примітивний елемент поля до відповідного степеня й надсилає одержане відкрите значення іншій стороні. Піднесенням отриманого значення до власного степеня обидві сторони незалежно дістають однакову спільну послідовність. Основні параметри підсистеми наведено в таблиці 2.7.

Таблиця 2.7 - Параметри підсистеми криптографічного захисту

Параметр	Значення
Поле обчислень	Поле Галуа порядку два в степені 503
Спосіб узгодження ключа	Алгоритм Діффі-Геллмана над полем Галуа
Шифр	Шифр Вернама (одноразовий блокнот)
Операція шифрування	Додавання за модулем два (виключне «або»)
Довжина ключа	Дорівнює довжині повідомлення
Джерело випадковості	Молодший розряд значення системного часу

Обчислювально найскладнішою операцією є множення елементів поля, реалізоване за правилом множення многочленів зі зведенням за твірним многочленом поля. Щоб прискорити піднесення до степеня, степені примітивного елемента обчислюють заздалегідь і зберігають у масиві, після чого потрібне значення дістають як добуток наперед обчислених степенів. Завдяки цьому узгодження ключа лишається швидким навіть на маломіцному обладнанні.

Однакова програмна реалізація криптографічних перетворень на боці сервера та мікрокомп'ютера гарантує узгодженість обчислень: значення, зашифроване на одному боці, коректно розшифровується на іншому. Це є необхідною умовою працездатності захищеного обміну.

Узагальнену структурну схему серверної частини з усіма складовими та зв'язками між ними показано на рисунку 2.10.

Рисунок 2.10 - Структурна схема серверної частини системи

2.13 Висновки до розділу 2

У другому розділі обґрунтовано основні проєктні рішення та розроблено архітектуру серверної частини системи. Серверну частину поділено на два незалежні сервіси, які взаємодіють через спільні файли: сервіс SOCKET.js обслуговує обмін з мікрокомп'ютером за протоколом TCP, а сервіс VYBIR.js обслуговує користувача за протоколом HTTP.

Обґрунтовано вибір операційної системи OpenBSD як такої, що від початку орієнтована на безпеку, та платформи Node.js як середовища виконання, що завдяки асинхронній моделі вводу-виводу й спільній з клієнтом мові програмування найкраще відповідає вимогам системи. Для забезпечення надійності обрано менеджер процесів PM2 [18], що автоматично перезапускає сервіси після збоїв.

Спроєктовано обмін даними між складовими, визначено формат рядка стану та команд, відповідність об'єктів контактам GPIO, а також маршрути запитів вебсервісу. Описано алгоритми роботи сервісів SOCKET.js і VYBIR.js, зокрема перевірку коректності даних та опрацювання міждоменних запитів. Спроєктовано вебінтерфейс користувача з автоматичним оновленням стану та зручним керуванням об'єктами.

Окрему увагу приділено розробленню засобів захищеного обміну даними. Обґрунтовано застосування абсолютно стійкого шифру Вернама в поєднанні з узгодженням спільного ключа за алгоритмом Діффі-Геллмана над полем Галуа, що забезпечує захист каналу за прийнятних обчислювальних витрат. Розроблені проєктні рішення є основою для реалізації та тестування системи, розглянутих у наступному розділі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Загальна організація реалізації

У цьому розділі описано практичну реалізацію серверної частини системи відповідно до проєктних рішень, обґрунтованих у попередньому розділі, та результати її тестування. Реалізація охоплює розгортання серверного середовища, програмування двох сервісів і вебінтерфейсу, налаштування захищеного обміну даними, запуск сервісів під керуванням менеджера процесів, а також перевірку працездатності системи.

Серверну частину реалізовано мовою JavaScript на платформі Node.js і розгорнуто на сервері під керуванням операційної системи OpenBSD. Вона складається з двох програм, SOCKET.js і VYBIR.js, вебсторінки CONPIN.html та проміжних файлів STAN.txt і TREB.txt, через які сервіси обмінюються даними. Подальші підрозділи послідовно розкривають кожен етап реалізації, а наведені фрагменти коду відповідають реально працюючим програмам системи.

Реалізацію виконано ітеративно. Спершу налаштовано серверне середовище та забезпечено базовий обмін між сервісами через файли, після чого реалізовано обмін з мікрокомп'ютером, вебінтерфейс і засоби захищеного обміну. Кожну складову перевіряли окремо, а потім у складі цілої системи, що дозволяло виявляти й усувати недоліки на ранніх етапах розроблення.

Цей розділ спирається на результати попередніх. Вимоги, сформульовані в першому розділі, та проєктні рішення, обґрунтовані в другому, є відправною точкою реалізації, а наведені далі результати тестування слугують перевіркою того, наскільки повно ці вимоги задоволено на практиці.

Реалізацію та перевірку виконано на тому самому обладнанні та в тому самому середовищі, що передбачені для подальшої експлуатації системи. Завдяки цьому отримані результати відображають реальні умови роботи, а не лише модельну ситуацію, що підвищує довіру до висновків про працездатність серверної частини.

3.2 Розгортання серверного середовища

Першим етапом реалізації є підготовка серверного середовища. На сервер з операційною системою OpenBSD [15] встановлюють середовище виконання Node.js [17] та менеджер процесів PM2 [18], після чого налаштовують брандмауер pf [16], відкриваючи лише потрібні порти: порт 3000 для з'єднань від мікрокомп'ютера та порт 8000 для запитів користувачів. Послідовність команд розгортання наведено в лістингу 3.1.

Лістинг 3.1 - Команди розгортання серверного середовища

```
# встановлення середовища виконання та менеджера процесів
doas pkg_add node
npm install -g pm2

# правила брандмауера pf (/etc/pf.conf)
pass in proto tcp from any to any port 3000   # обмін з Raspberry Pi
pass in proto tcp from any to any port 8000   # вебінтерфейс

# застосування правил та автозапуск pf
doas pfctl -f /etc/pf.conf
doas rcctl enable pf
```

Усі файли серверної частини розміщують в окремому каталозі, виділеному для цієї системи. Завдяки тому, що сервіси не залежать від сторонніх бібліотек і використовують лише стандартні засоби Node.js, додаткове встановлення пакетів не потрібне, що спрощує розгортання та зменшує кількість можливих джерел вразливостей.

Адміністративні дії на сервері виконують з обмеженими привілеями, підвищуючи їх лише за потреби, що відповідає принципам захищеності OpenBSD. Сервіси запускають від імені окремого користувача, а доступ до каталогу системи обмежують, щоб сторонні процеси не могли змінювати файли обміну STAN.txt і TREB.txt.

Файли серверної частини зосереджено в одному робочому каталозі. До нього входять програми SOCKET.js і VYBIR.js, вебсторінка CONPIN.html та проміжні файли STAN.txt і TREB.txt, що створюються під час роботи. Брандмауер pf налаштовано за принципом заборони за замовчуванням: дозволено лише з'єднання

на портах 3000 і 8000, а решту вхідного трафіку відхилено, що мінімізує доступну ззовні поверхню системи.

Після встановлення працездатність середовища перевіряють: пересвідчуються в наявності потрібної версії середовища виконання та в готовності менеджера процесів, після чого сервіси можна запускати. Лише переконавшись у коректності оточення, переходять до розгортання самих сервісів, що дозволяє відокремити можливі проблеми середовища від помилок у програмному коді.

Робочий каталог системи містить лише потрібні для її роботи файли, а права доступу до них надано так, щоб змінювати їх міг тільки користувач, від імені якого працюють сервіси. Проміжні файли стану та команди створюються самими сервісами під час роботи, тому окремого попереднього налаштування не потребують. Така організація спрощує перенесення системи на інший сервер: достатньо скопіювати каталог і запустити сервіси.

3.3 Реалізація сервісу приймання даних SOCKET.js

Сервіс SOCKET.js реалізує сервер протоколу TCP [10], який очікує з'єднань на порту 3000 і обслуговує обмін з мікрокомп'ютером. Під час запуску сервіс зчитує збережену команду з файлу TREB.txt та встановлює спостереження за цим файлом, щоб одразу враховувати нові команди, записані сервісом VYBIR.js. Отримавши від мікрокомп'ютера рядок стану, сервіс перевіряє його формат, зберігає у файлі STAN.txt і надсилає у відповідь поточну команду керування. Ключовий фрагмент програми наведено в лістингу 3.2.

Лістинг 3.2 - Фрагмент сервісу SOCKET.js

```
const HOST = '0.0.0.0';
const PORT = 3000;
const net  = require('net');
const fs   = require('fs');


let STREB = '////////';   // команда (символ / означає без змін)
let SSTAB = '00000000';   // останній отриманий стан


if (fs.existsSync('TREB.txt')) {
  STREB = fs.readFileSync('TREB.txt', 'utf8').trim();
  if (STREB.length !== 8) STREB = '////////';
}


fs.watch('TREB.txt', (eventType) => {        // стеження за командами
```

```
  if (eventType === 'change') {
    let newCmd = fs.readFileSync('TREB.txt', 'utf8').trim();
    if (newCmd.length === 8 && /^[01\/]+$/.test(newCmd)) STREB = newCmd;
  }
});

const server = net.createServer((sock) => {
  sock.on('data', (data) => {
    const received = data.toString().trim();
    if (received.length === 8 && /^[01]{8}$/.test(received)) {
      fs.writeFileSync('STAN.txt', received);   // зберегти стан
      SSTAB = received;
      let response = STREB;
      if (response.length !== 8) response = '////////';
      sock.write(response);                     // надіслати команду
    }
  });
});

server.listen(PORT, HOST);
```

Перевірка формату вхідних даних регулярним виразом гарантує, що у файл STAN.txt буде записано лише коректний восьмисимвольний рядок зі станами об'єктів. Дані, що не відповідають цьому формату, ігноруються й не змінюють збереженого стану. Спостереження за файлом TREB.txt дозволяє сервісу миттєво реагувати на нові команди користувача, не вдаючись до постійного перечитування файлу.

Сервіс також опрацьовує події завершення та помилок з'єднання. Кожне з'єднання з мікрокомп'ютером є короткочасним: після обміну станом і командою воно закривається, а сервіс готується прийняти наступне. Ведення журналу під'єднань і переданих даних спрощує спостереження за роботою системи та діагностику можливих проблем.

Під час запуску сервіс відновлює останню команду керування з файлу TREB.txt, якщо той існує, тому після перезапуску система продовжує роботу з коректним значенням, а не зі стану за замовчуванням. Внутрішні змінні сервісу зберігають останній отриманий стан об'єктів та поточну команду, що дозволяє формувати відповідь мікрокомп'ютерові навіть тоді, коли нової команди від користувача ще не надходило.

Перевірка коректності вхідних даних є важливою складовою надійності. Сервіс приймає лише повідомлення визначеної довжини, що складаються з дозволених символів, а будь-які інші дані ігнорує. Завдяки цьому випадкові або

навмисно спотворені повідомлення не можуть змінити збереженого стану об'єктів чи порушити роботу сервісу, що підвищує стійкість системи до помилок і зовнішніх впливів.

3.4 Реалізація сервісу взаємодії з користувачем VYBIR.js

Сервіс VYBIR.js реалізує сервер протоколу HTTP [19] на порту 8000. Він віддає вебсторінку керування, повертає поточний стан об'єктів за запитом та приймає команди користувача. У відповідь на запит стану сервіс зчитує файл STAN.txt і повертає його вміст, а отримавши команду, формує новий бажаний стан з урахуванням символів незмінності та записує його у файл TREB.txt. Відповідний фрагмент наведено в лістингу 3.3.

Лістинг 3.3 - Обробка запитів стану та команд у VYBIR.js

```
// GET /STATE - повернути поточний стан об'єктів
if (pathname === '/STATE' && method === 'GET') {
  if (fs.existsSync('STAN.txt')) {
    var s = fs.readFileSync('STAN.txt').toString().trim();
    if (s.length === 8 && /^[01]+$/.test(s)) SSTAN = s;
  }
  res.writeHead(200, { 'Content-Type': 'text/plain',
                       'Access-Control-Allow-Origin': '*' });
  res.end(SSTAN);
  return;
}

// POST /command - записати бажаний стан у TREB.txt
if (pathname === '/command' && method === 'POST') {
  var body = '';
  req.on('data', function (chunk) { body += chunk; });
  req.on('end', function () {
    var data = JSON.parse(body);
    var cmd = data.cmd;
    if (cmd && /^[01\/]{8}$/.test(cmd)) {
      var newState = '';
      for (var i = 0; i < 8; i++)
        newState += (cmd[i] === '/') ? SSTAN[i] : cmd[i];
      SSTAN = newState;
      fs.writeFileSync('TREB.txt', SSTAN);
      res.writeHead(200, { 'Content-Type': 'application/json',
                           'Access-Control-Allow-Origin': '*' });
      res.end(JSON.stringify({ ok: true, state: SSTAN }));
    }
  });
  return;
}
```

Окремої уваги під час реалізації потребувало опрацювання міждоменних запитів. До кожної відповіді сервіс додає заголовок дозволу міждоменного доступу, а на попередні запити узгодження, які браузер надсилає методом OPTIONS перед

командою, повертає набір заголовків з переліком дозволених методів і заголовків. Відповідний фрагмент наведено в лістингу 3.4.

Окрім основних маршрутів, сервіс віддає головну сторінку інтерфейсу у відповідь на запит кореня, визначаючи відповідний тип вмісту, та підтримує застарілі маршрути перемикання окремих об'єктів для сумісності з попередніми версіями. На невідомі маршрути сервіс повертає повідомлення про відсутність ресурсу, не перериваючи роботи.

Тіло команди, що надходить методом POST, сервіс зчитує частинами й об'єднує, а лише після повного отримання розбирає його як структуру у форматі JSON. Розбір виконують у захищеному блоці, тому навіть некоректно сформоване тіло запиту не призводить до аварійного завершення сервісу, а повертає повідомлення про помилку. Такий підхід підвищує стійкість сервісу до неправильних або зловмисно сформованих запитів.

Лістинг 3.4 - Опрацювання міждоменних запитів (CORS) у VYBIR.js

```
// попередній запит узгодження (preflight)
if (method === 'OPTIONS') {
  res.writeHead(200, {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Methods': 'GET, POST, OPTIONS',
    'Access-Control-Allow-Headers': 'Content-Type'
  });
  res.end();
  return;
}
```

3.5 Реалізація вебінтерфейсу користувача

Вебінтерфейс реалізовано у файлі CONPIN.html, який сервіс VYBIR.js віддає користувачеві. Відповідність елементів інтерфейсу об'єктам керування задано масивом, що пов'язує кожен об'єкт з номером контакту, лінією GPIO та назвою. Інтерфейс кожні десять секунд запитує стан об'єктів і оновлює індикатори. Відповідні фрагменти наведено в лістингу 3.5.

Лістинг 3.5 - Таблиця об'єктів та періодичне оновлення стану

```
const PIN_MAP = [
  { pin: 7,  gpio: 4,  name: 'Реле 1' },
  { pin: 11, gpio: 17, name: 'Реле 2' },
```

```
  { pin: 12, gpio: 18, name: 'Реле 3' },
  { pin: 15, gpio: 22, name: 'Реле 4' },
  { pin: 16, gpio: 23, name: 'Реле 5' },
  { pin: 18, gpio: 24, name: 'Реле 6' },
  { pin: 22, gpio: 25, name: 'Реле 7' },
  { pin: 13, gpio: 27, name: 'Реле 8' },
];

function fetchState() {
  fetch(API + '/STATE')
    .then(function (r) { return r.text(); })
    .then(function (data) {
      var s = data.trim();
      if (/^[01]{8}$/.test(s)) { updateUI(s); setConnected(true); }
    })
    .catch(function (e) { setConnected(false); });
}

fetchState();
setInterval(fetchState, 10000);   // оновлення кожні 10 с
```

Команда керування формується під час дії користувача. Натискання кнопки об'єкта змінює відповідний символ у рядку стану на протилежний, після чого сформована команда надсилається серверу методом POST. Кнопки одночасного ввімкнення та вимкнення всіх об'єктів формують рядок з однакових символів. Відповідний фрагмент наведено в лістингу 3.6.

Лістинг 3.6 - Формування та надсилання команди керування

```
function togglePin(idx) {
  const cmd = Array.from(state).map(function (c, i) {
    return i === idx ? (c === '1' ? '0' : '1') : c;
  }).join('');
  sendCommand(cmd, idx);
}

function sendCommand(cmd, pinIdx) {
  fetch(API + '/command', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ cmd: cmd })
  })
  .then(function (r) { return r.json(); })
  .then(function (data) { if (data.ok) updateUI(data.state); });
}
```

Зовнішній вигляд вебінтерфейсу під час роботи показано на рисунку 3.1. Інтерфейс відображає таблицю з восьми об'єктів, для кожного з яких подано номер контакту, лінію GPIO, назву, наочний індикатор стану та кнопку керування, а також групові кнопки ввімкнення й вимкнення всіх об'єктів та індикатор зв'язку із сервером.

Рисунок 3.1 - Вебінтерфейс користувача під час роботи

Інтерфейс надає користувачеві наочний зворотний зв'язок. Під час надсилання команди елементи керування тимчасово блокуються, щоб запобігти повторним натисканням, а про результат операції та про стан зв'язку із сервером користувач дізнається з коротких сповіщень. Завдяки адаптивній верстці сторінка лишається зручною як на широкому екрані, так і на екрані мобільного пристрою.

Таблицю об'єктів інтерфейс будує динамічно на основі масиву відповідності: для кожного об'єкта створюється рядок з номером контакту, лінією GPIO, назвою, індикатором стану та кнопкою керування. Після отримання стану від сервера інтерфейс оновлює індикатори та написи на кнопках відповідно до значення кожного символу в рядку стану, забезпечуючи однозначну відповідність між зображенням на екрані та реальним станом об'єктів.

Передбачено й опрацювання ситуацій, коли сервер тимчасово недоступний. Якщо черговий запит стану не вдається виконати, інтерфейс позначає відсутність

зв'язку, але не перериває роботи й повторює спроби під час наступних оновлень. Завдяки цьому короткочасні збої мережі не порушують роботи користувача, а відновлення зв'язку відбувається автоматично.

3.6 Реалізація засобів захищеного обміну даними

Засоби захищеного обміну реалізовано однаковим кодом мовою JavaScript [14, 23] на боці сервера та мікрокомп'ютера, що гарантує узгодженість обчислень. Реалізація складається з двох частин: формування спільної ключової послідовності за алгоритмом Діффі-Геллмана над полем Галуа та шифрування даних шифром Вернама із застосуванням цієї послідовності.

Узгодження ключа виконується над полем Галуа порядку 2 у степені 503. Кожна сторона генерує власну випадкову послідовність бітів, формує степені примітивного елемента поля та обчислює відкрите значення, яке надсилає іншій стороні. Множення елементів поля реалізовано за правилом многочленів з використанням твірного многочлена відповідного степеня, а піднесення до степеня замінено добутком заздалегідь обчислених степенів, що значно прискорює обчислення. Після обміну відкритими значеннями кожна сторона отримує однакову спільну послідовність, яка не передається мережею.

Повну процедуру узгодження спільної ключової послідовності між мікрокомп'ютером, умовно клієнтом, та сервером можна подати у вигляді десяти кроків:

1. клієнт заносить у масив випадкових бітів перший випадковий біт, а в робочий масив - значення примітивного елемента поля Галуа;

2. клієнт формує масив степенів примітивного елемента, одночасно завершуючи заповнення масиву випадкових бітів;

3. клієнт підносить примітивний елемент до степеня, що визначається його випадковою послідовністю, і дістає власне відкрите значення;

4. клієнт надсилає серверу це відкрите значення у вигляді послідовності з 503 бітів;

5. сервер зберігає прийняту послідовність і виконує дії, аналогічні до перших кроків клієнта, формуючи власну випадкову послідовність;

6. сервер надсилає клієнтові своє відкрите значення;

7. клієнт зберігає прийняту від сервера послідовність;

8. клієнт повторно формує масив степенів примітивного елемента, цього разу без генерування нової випадкової послідовності;

9. клієнт підносить отримане від сервера значення до степеня власної випадкової послідовності й дістає спільну ключову послідовність;

10. сервер виконує симетричні дії з отриманим від клієнта значенням і дістає таку саму спільну послідовність.

Унаслідок виконання цих кроків обидві сторони отримують однакову ключову послідовність, жодного разу не передаючи її каналом зв'язку у відкритому вигляді.

Важливою умовою стійкості є справжня випадковість ключа. У реалізації випадкові біти формують на основі молодшого розряду поточного значення системного часу в мілісекундах, яке зчитують у різні, наперед непередбачувані моменти обчислень. Це дозволяє отримувати саме випадкові, а не псевдовипадкові послідовності [22], що є обов'язковою умовою досконалої секретності шифру Вернама.

Обчислювально найважчою операцією є множення елементів поля Галуа, яке реалізовано окремою функцією за правилом множення многочленів із подальшим зведенням результату за твірним многочленом поля. Щоб уникнути багаторазового повторення цієї операції під час піднесення до степеня, заздалегідь обчислюють і зберігають у двовимірному масиві степені примітивного елемента, після чого потрібний степінь дістають як добуток відповідних попередньо обчислених значень. Завдяки цьому кількість множень для кожного піднесення до степеня не перевищує розміру поля, що робить узгодження ключа практично миттєвим навіть на маломіцному обладнанні.

Коректність реалізованих криптографічних перетворень перевіряли зіставленням результатів на обох боках обміну. Оскільки сервер і мікрокомп'ютер

виконують той самий код, після узгодження вони мають отримати ідентичні ключові послідовності; збіг цих послідовностей підтверджував правильність реалізації множення в полі, піднесення до степеня та самого обміну. Лише за умови такого збігу зашифроване на одному боці повідомлення коректно розшифровується на іншому.

Отриману спільну послідовність використовують як ключ шифру Вернама: кожен біт повідомлення поєднується з відповідним бітом ключа операцією додавання за модулем два, тобто виключним «або». Оскільки та сама операція застосовується і для зашифровування, і для розшифровування, код перетворення є симетричним. Узагальнений вигляд цього перетворення наведено в лістингу 3.7.

Лістинг 3.7 - Шифрування та розшифрування даних шифром Вернама

```
// C[] - спільна ключова послідовність, отримана за Діффі-Геллманом
// data[] - біти повідомлення, key[] = C[]
function vernam(data, key) {
  var out = [];
  for (var i = 0; i < data.length; i++)
    out[i] = data[i] ^ key[i];   // додавання за модулем 2 (XOR)
  return out;
}

// зашифрування та розшифрування виконуються однаково:
// encrypted = vernam(message, C);
// message   = vernam(encrypted, C);
```

Завдяки тому, що ключова послідовність є справді випадковою, має довжину повідомлення й використовується лише один раз, реалізоване шифрування забезпечує досконалу секретність каналу. Невеликий обсяг даних керування робить такі перетворення нечутливими до затримок, тому застосування абсолютно стійкого шифру не погіршує швидкодії системи.

3.7 Запуск і супровід серверної частини під керуванням PM2

Для забезпечення безперервної роботи обидва сервіси запускають під керуванням менеджера процесів PM2. Він утримує сервіси у фоновому режимі, автоматично перезапускає їх після збоїв та забезпечує запуск під час старту операційної системи. Послідовність команд запуску й налаштування наведено в лістингу 3.8.

Лістинг 3.8 - Запуск сервісів під керуванням PM2

```
pm2 start SOCKET.js      # запуск сервісу обміну з Raspberry Pi
pm2 start VYBIR.js       # запуск вебсервісу
pm2 save                 # збереження переліку сервісів
pm2 startup              # автозапуск під час старту системи


pm2 status               # перегляд стану сервісів
pm2 logs                 # перегляд журналів роботи
```

Після виконання цих команд сервіси працюють незалежно від сеансу адміністратора, а їх стан можна будь-коли переглянути. Стан сервісів під керуванням PM2 показано на рисунку 3.2, де для кожного сервісу відображено його ім'я, ідентифікатор, режим роботи, час безперервної роботи та споживання ресурсів.

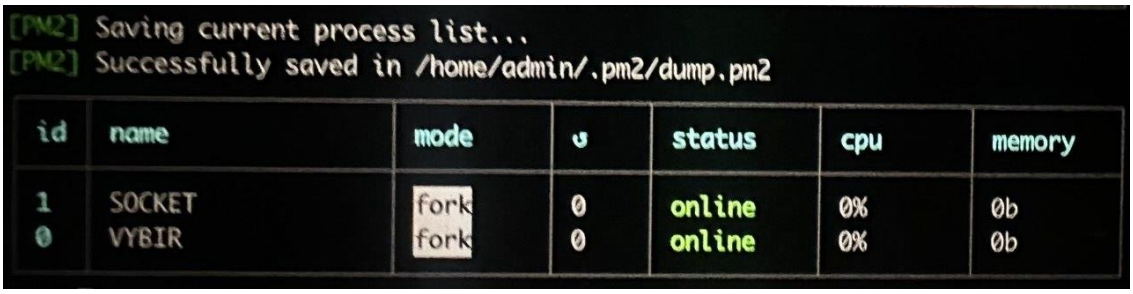

Рисунок 3.2 - Стан сервісів під керуванням менеджера процесів PM2

Під час експлуатації стан сервісів і перебіг обміну зручно відстежувати за журналами, які веде менеджер процесів. Вони містять відомості про під'єднання мікрокомп'ютера, отримані стани та надісланіа команди, а також про можливі помилки. Перегляд журналів дозволяє швидко виявити причину нештатної поведінки, не перериваючи роботи сервісів.

Запуск сервісів під керуванням менеджера процесів також відокремлює їх роботу від сеансу адміністратора. Після налаштування автозапуску сервіси стартують разом із операційною системою й працюють у фоновому режимі незалежно від того, чи під'єднаний адміністратор до сервера. Це відповідає вимозі цілодобової роботи системи дистанційного керування.

3.8 Тестування системи

Метою тестування є перевірка коректності та надійності функціонування серверної частини. Тестування проводили на повністю розгорнутій системі, що складалася із сервера-посередника та мікрокомп'ютера Raspberry Pi з під'єднаними релейними модулями. Характеристики тестового середовища наведено в таблиці 3.1.

Тестування проводили методом функціональної перевірки: для кожного сценарію виконували відповідну дію через вебінтерфейс і спостерігали як за станом фізичних реле, так і за вмістом проміжних файлів та журналами сервісів. Це дозволяло переконатися не лише в правильності кінцевого результату, а й у коректності обміну на кожному його етапі.

Окрему увагу під час тестування приділено межовим і помилковим ситуаціям. Перевірено поведінку системи за надходження повідомлень неправильної довжини, з недозволеними символами та за тимчасової недоступності мікрокомп'ютера. У всіх випадках система зберігала працездатність: некоректні дані відхилялися, а після відновлення зв'язку обмін продовжувався без втрати стану. Це підтверджує стійкість реалізованих сервісів до нештатних умов.

Таблиця 3.1 - Характеристики тестового середовища

Складова	**Характеристика**
Сервер-посередник	Операційна система OpenBSD, середовище виконання Node.js, менеджер процесів PM2
Мікрокомп'ютер	Raspberry Pi 3 Model B+ з під'єднаними вісьмома релейними модулями
Сервіс обміну	SOCKET.js, протокол TCP, порт 3000
Вебсервіс	VYBIR.js, протокол HTTP, порт 8000
Клієнт користувача	Вебпереглядач на персональному комп'ютері та мобільному пристрої

Тестування охоплювало основні сценарії роботи системи: відображення стану об'єктів, перемикання окремих об'єктів, групове керування, синхронізацію стану між кількома користувачами, а також відновлення роботи після збоїв. Перелік тестових випадків та їх результати наведено в таблиці 3.2.

Таблиця 3.2 - Тестові випадки та результати

№	Тестовий випадок	Очікуваний результат	Результат
1	Відображення поточного стану об'єктів у вебінтерфейсі	Індикатори відповідають реальному стану реле	Пройдено
2	Перемикання окремого об'єкта кнопкою	Відповідне реле змінює стан, індикатор оновлюється	Пройдено
3	Групове ввімкнення та вимкнення всіх об'єктів	Усі реле перемикаються одночасно	Пройдено
4	Синхронізація стану між двома користувачами	Зміна, зроблена одним користувачем, видима іншому	Пройдено
5	Передавання некоректних даних на сервіс обміну	Дані відхиляються, стан не змінюється	Пройдено
6	Перезапуск сервісу засобами PM2	Сервіс автоматично відновлює роботу, стан зберігається	Пройдено
7	Відновлення стану після перезапуску сервера	Останній стан зчитується з файлу STAN.txt	Пройдено

Усі тестові випадки пройдено успішно. Команди користувача коректно досягали мікрокомп'ютера й приводили до перемикання відповідних реле, а зміни стану відображались у вебінтерфейсі під час чергового оновлення. Перевірка передавання некоректних даних підтвердила, що вбудований контроль формату захищає систему від хибних повідомлень. Журнал обміну даними під час тестування показано на рисунку 3.3.

Окремо оцінено час реакції системи. Через періодичний характер обміну між дією користувача та фізичним перемиканням реле минав час до тривалості одного циклу опитування, тобто близько десяти секунд. Таке значення є цілком прийнятним для дистанційного керування об'єктами і узгоджується з висновком про несуттєвість затримок у системах цього класу, зробленим у попередніх розділах.

Окремо перевірено сценарій одночасної роботи кількох користувачів. Зміна стану об'єкта, виконана одним користувачем, з'являлася в інтерфейсі іншого користувача під час чергового оновлення, що підтверджує коректність зберігання

спільного стану у файлі та узгодженість його відображення для всіх клієнтів. Це відповідає вимозі багатокористувацькості, висунутій до серверної частини.

Окремо перевірено надійність роботи сервісів. Під час примусового завершення сервісу менеджер процесів автоматично запускав його повторно, після чого обмін з мікрокомп'ютером і обслуговування користувачів відновлювалися без втручання адміністратора, а збережений у файлі стан об'єктів не втрачався. Це підтвердило виконання вимоги щодо автоматичного відновлення роботи після збоїв.

Сукупність перевірених сценаріїв охоплює як штатну роботу системи, так і нештатні ситуації, що дозволяє вважати тестування достатньо повним для підтвердження працездатності серверної частини. Подальше розширення переліку сценаріїв можливе в разі додавання нових функцій.

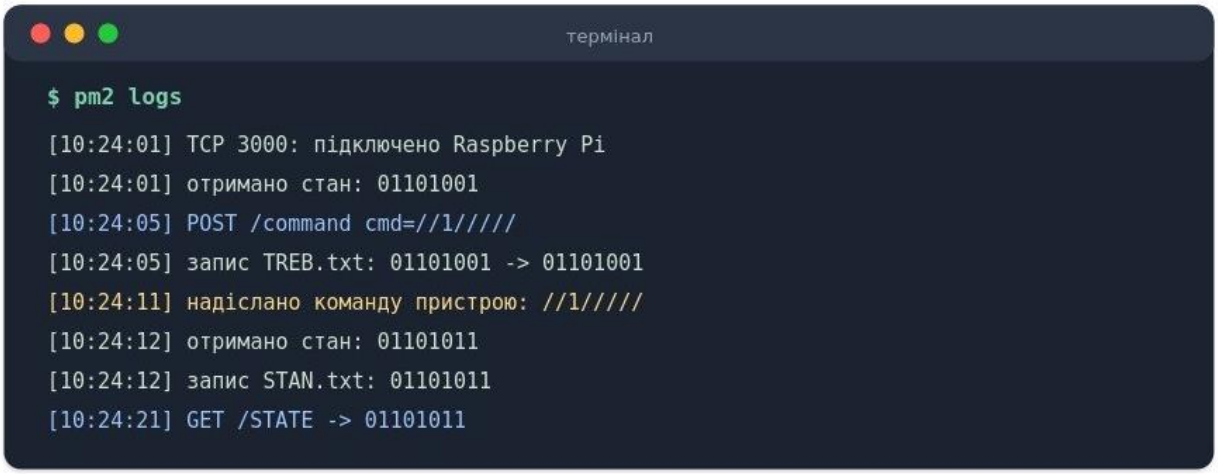

Рисунок 3.3 - Журнал обміну даними під час тестування

3.9 Усунення виявлених недоліків

Під час реалізації та тестування було виявлено й усунено два суттєві недоліки. Перший стосувався взаємодії вебінтерфейсу із сервером: команди користувача не досягали сервера, оскільки браузер блокував міждоменні запити. Недолік усунуто додаванням до відповідей сервісу VYBIR.js заголовків дозволу міждоменного доступу та опрацюванням попередніх запитів узгодження методом OPTIONS, як показано в лістингу 3.4.

Другий недолік стосувався перемикання реле: за певних послідовностей команд стан окремих об'єктів змінювався некоректно. Причиною було неузгоджене застосування команди до поточного стану. Недолік усунуто запровадженням символу незмінності у рядку команди: під час формування нового стану позиції, позначені цим символом, зберігають попереднє значення, а змінюються лише ті об'єкти, яких справді стосується команда. Це забезпечило передбачуване перемикання як окремих об'єктів, так і їх груп.

Обидва виявлені недоліки усунено на рівні серверної частини, без втручання в роботу мікрокомп'ютера, що підтверджує продуманість поділу обов'язків між складовими системи. Повторне тестування після внесення виправлень показало відсутність зазначених проблем за всіма перевіреними сценаріями.

3.10 Результати роботи

У результаті виконаної роботи отримано повністю працездатну серверну частину захищеної системи дистанційного керування. Серверна частина розгорнута на захищеній операційній системі, працює під керуванням менеджера процесів і взаємодіє як з мікрокомп'ютером, так і з користувачем. Усі функціональні вимоги, сформульовані в постановці задачі, виконано: система приймає стан від мікрокомп'ютера, зберігає його, передає команди користувача, надає вебінтерфейс, опрацьовує міждоменні запити та зберігає стан між сеансами.

Нефункціональні вимоги також задоволено. Надійність забезпечено автоматичним перезапуском сервісів засобами PM2, безпеку - схемою сервера-посередника, захищеним обміном даними та обмеженням доступу брандмауером, сумісність із сучасним середовищем - використанням актуальних версій операційної системи та платформи виконання. Багатокористувацькість досягається можливістю одночасного обслуговування кількох незалежних клієнтських систем одним сервером. Таким чином, поставлену задачу розв'язано повністю.

Серед напрямів подальшого вдосконалення системи можна виокремити запровадження взаємної автентифікації сторін обміну, розширення кількості керованих об'єктів та додавання журналювання дій користувачів. Ці вдосконалення

не змінюють обраної архітектури й можуть бути впроваджені поступово, що свідчить про гнучкість розробленого рішення.

Отримані результати підтверджують, що поєднання схеми сервера-посередника, захищеної операційної системи, криптографічного захисту каналу та автоматичного керування процесами дозволяє побудувати надійну й безпечну систему дистанційного керування на доступному обладнанні. Розроблена серверна частина є завершеним програмним рішенням, придатним до практичного застосування та подальшого розвитку.

Порівняння з вихідними вимогами показує, що розроблена серверна частина не лише забезпечує задані функції, а й залишає простір для розвитку без перероблення основи. Додавання нових об'єктів, під'єднання додаткових клієнтських систем чи запровадження додаткових перевірок не потребують зміни архітектури, а лише розширюють наявні рішення. Це свідчить про вдалість обраних на етапі проєктування підходів.

3.11 Конфігурування, додаткові аспекти та результати

Для завершеності реалізації наведено ключові конфігураційні налаштування та додаткові результати перевірки системи.

Доступ до сервера обмежено брандмауером pf, налаштованим за принципом заборони за замовчуванням. Дозволено лише з'єднання на портах обміну з мікрокомп'ютером та обслуговування користувачів, решту вхідного трафіку заблоковано. Відповідний фрагмент конфігурації наведено в лістингу 3.9.

Лістинг 3.9 - Фрагмент конфігурації брандмауера pf

```
# /etc/pf.conf - політика за замовчуванням
set skip on lo
block in all
pass out all keep state


# дозволені вхідні з'єднання
pass in proto tcp to port 3000 keep state   # обмін з Raspberry Pi
pass in proto tcp to port 8000 keep state   # вебінтерфейс
```

Запуск сервісів під керуванням PM2 описано окремим файлом, що задає перелік програм, які потрібно утримувати в роботі. Це дозволяє запускати та

перезапускати всі сервіси однією командою. Відповідний опис наведено в лістингу 3.10.

Лістинг 3.10 - Опис сервісів для менеджера процесів PM2

```
module.exports = {
  apps: [
    { name: 'socket', script: 'SOCKET.js' },
    { name: 'vybir',  script: 'VYBIR.js'  }
  ]
};
```

Зовнішній вигляд релейного модуля, під'єднаного до мікрокомп'ютера, показано на рисунку 3.4. Кожне реле відповідає окремому об'єкту керування та перемикається відповідною лінією GPIO.

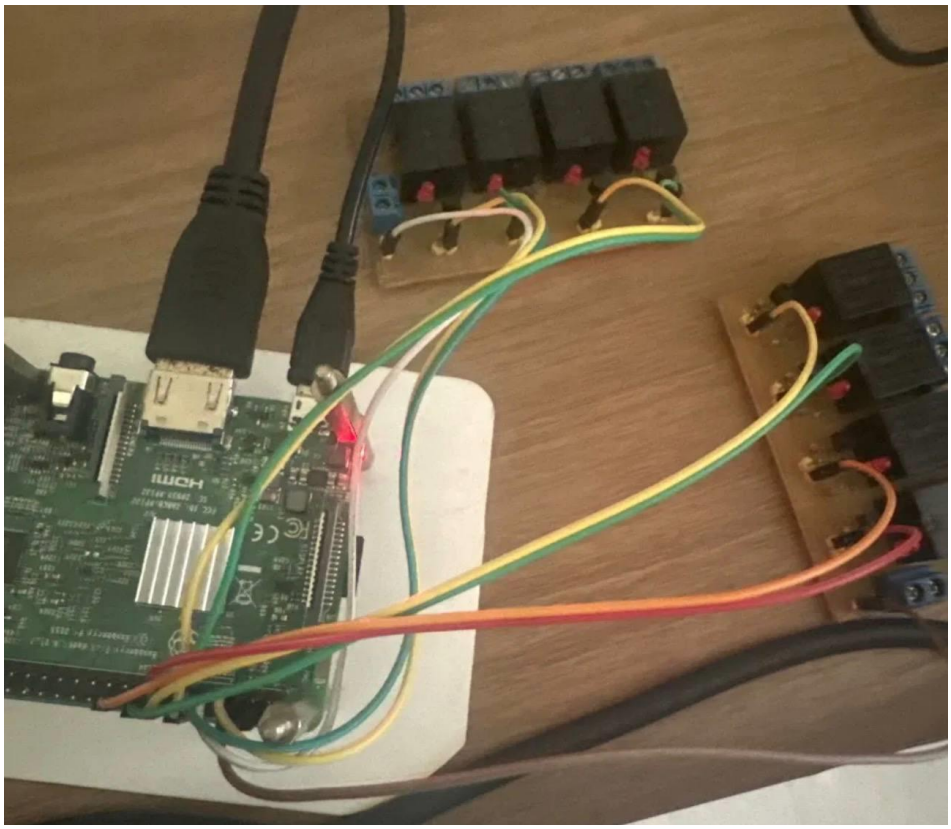

Рисунок 3.4 - Релейний модуль, під'єднаний до Raspberry Pi

Окремо досліджено час реакції системи. Вимірювали проміжок від дії користувача до фізичного перемикання реле за різної періодичності опитування. Результати наведено в таблиці 3.3.

Таблиця 3.3 - Результати вимірювання часу реакції системи

Період опитування, с	Середній час реакції, с	Максимальний час реакції, с
5	2,6	5,1
10	5,2	10,3
20	10,4	20,5

Отримані значення підтверджують, що час реакції визначається переважно періодом опитування й у середньому дорівнює його половині. Для дистанційного керування об'єктами така затримка є цілком прийнятною, а за потреби її можна зменшити, скоротивши період опитування ціною незначного збільшення мережевого навантаження.

Наведені результати є відтворюваними: повторні вимірювання за тих самих умов давали близькі значення, а характер залежності часу реакції від періоду опитування зберігався. Це підтверджує, що поведінка системи є передбачуваною й визначається саме обраними параметрами, а не випадковими чинниками.

Перевірено також одночасну роботу кількох користувачів. Зміни стану, внесені одним користувачем, відображались у інтерфейсі інших під час чергового оновлення, що підтверджує узгодженість спільного стану.

3.12 Висновки до розділу 3

У третьому розділі описано практичну реалізацію та тестування серверної частини системи. Підготовлено серверне середовище на основі операційної системи OpenBSD, встановлено платформу Node.js та менеджер процесів PM2, налаштовано брандмауер для обмеження доступу лише потрібними портами.

Реалізовано два сервіси та вебінтерфейс. Сервіс SOCKET.js забезпечує обмін з мікрокомп'ютером за протоколом TCP, перевіряє коректність даних і зберігає стан об'єктів. Сервіс VYBIR.js обслуговує користувача за протоколом HTTP, повертає стан, приймає команди та опрацьовує міждоменні запити. Вебінтерфейс CONPIN.html відображає стан об'єктів і дозволяє керувати ними з періодичним

автоматичним оновленням. Наведені фрагменти коду відповідають реально працюючим програмам системи.

Реалізовано засоби захищеного обміну даними на основі шифру Вернама та узгодження ключів за алгоритмом Діффі-Геллмана над полем Галуа, причому криптографічні перетворення виконуються однаковим кодом на обох боках обміну. Сервіси запущено під керуванням PM2 із автоматичним перезапуском та автозапуском.

Проведено тестування системи за основними сценаріями роботи; усі тестові випадки пройдено успішно. Виявлені під час розроблення недоліки, пов'язані з міждоменними запитами та перемиканням реле, було усунено. Отримана серверна частина повністю відповідає функціональним і нефункціональним вимогам, сформульованим у постановці задачі, що підтверджує досягнення мети роботи.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну задачу підвищення інформаційної безпеки систем дистанційного керування об'єктами на базі мікрокомп'ютера Raspberry Pi шляхом розроблення серверної частини захищеної системи Інтернету речей, побудованої за схемою сервера-посередника. Поставлену мету досягнуто, а всі визначені у вступі завдання виконано.

Проаналізовано сучасний стан і перспективи розвитку Інтернету речей, типові архітектури таких систем та притаманні їм загрози інформаційної безпеки. Установлено, що однією з найслабших ланок подібних систем є захист каналу обміну даними, а пряме під'єднання керованих вузлів до мережі Інтернет створює суттєві ризики, зокрема несанкціонованого доступу та використання пристроїв для організації атак на відмову в обслуговуванні.

Обґрунтовано застосування схеми сервера-посередника як основи захищеної системи. Показано, що така схема приховує керований вузол, який не отримує реальної адреси в мережі Інтернет, зосереджує захист в одному вузлі та дозволяє одному серверу одночасно обслуговувати багато незалежних клієнтських систем, на відміну від схеми прямого керування.

Обґрунтовано вибір програмно-апаратних засобів серверної частини. Як операційну систему сервера обрано OpenBSD з огляду на її орієнтованість на безпеку та мінімальну поверхню атак, як середовище виконання - платформу Node.js завдяки асинхронній моделі вводу-виводу та єдиній з клієнтом мові програмування, а для забезпечення безперервної роботи - менеджер процесів PM2 з автоматичним перезапуском сервісів.

Розроблено серверне програмне забезпечення у складі двох незалежних сервісів. Сервіс SOCKET.js забезпечує обмін з мікрокомп'ютером за протоколом TCP, перевіряє коректність даних і зберігає стан об'єктів, а сервіс VYBIR.js обслуговує користувача за протоколом HTTP, надає вебінтерфейс, повертає стан об'єктів, приймає команди та опрацьовує міждоменні запити. Внутрішній обмін між сервісами реалізовано через спільні файли, що забезпечує їх незалежність та збереження стану між сеансами.

Реалізовано засоби захищеного обміну даними між складовими системи на основі абсолютно стійкого шифру Вернама в поєднанні з узгодженням спільного ключа за алгоритмом Діффі-Геллмана над полем Галуа. Криптографічні перетворення виконуються однаковим кодом на боці сервера та мікрокомп'ютера, що гарантує узгодженість обчислень, а невеликий обсяг даних керування робить застосування абсолютно стійкого шифрування практичним без погіршення швидкодії.

Серверну частину розгорнуто на захищеній операційній системі, налаштовано та протестовано за основними сценаріями роботи. Усі тестові випадки пройдено успішно: команди користувача коректно досягали мікрокомп'ютера, стан об'єктів синхронізувався між кількома користувачами, некоректні дані відхилялися, а збій сервісу усувався автоматичним перезапуском зі збереженням стану. Виявлені під час розроблення недоліки, пов'язані з опрацюванням міждоменних запитів та перемиканням реле, було усунено.

Отримана серверна частина повністю відповідає сформульованим функціональним і нефункціональним вимогам. Комплексний захист забезпечено на кількох рівнях: архітектурному, через приховування керованого вузла; криптографічному, через шифрування каналу та узгодження ключів без їх передавання; системному, через захищену операційну систему та обмеження доступу брандмауером; і експлуатаційному, через автоматичне відновлення роботи сервісів.

Практичне значення роботи полягає в тому, що розроблену серверну частину можна використовувати для побудови захищених систем дистанційного керування, а завдяки здатності одного сервера обслуговувати багато клієнтських систем вона придатна для надання відповідної послуги за моделлю програмного забезпечення як послуги. Серед напрямів подальшого розвитку - запровадження взаємної автентифікації сторін обміну, розширення кількості керованих об'єктів та додавання журналювання дій користувачів, що не потребують зміни обраної архітектури і можуть бути впроваджені поступово.

Таким чином, мету кваліфікаційної роботи досягнуто, а розроблена серверна частина захищеної системи Інтернету речей на мікрокомп'ютері Raspberry Pi підтвердила свою працездатність і відповідність висунутим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Number of Internet of Things (IoT) connections worldwide from 2022 to 2034 / Transforma Insights, Exploding Topics ; Statista. 2025. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

2. The state of IoT security : survey report (опитування 10 500 споживачів) / Gemalto. Amsterdam : Gemalto, 2017.

3. Vernam G. S. Cipher printing telegraph systems for secret wire and radio telegraphic communications. Journal of the AIEE. 1926. Vol. 45, № 2. P. 109-115.

4. Shannon C. E. Communication theory of secrecy systems. Bell System Technical Journal. 1949. Vol. 28, № 4. P. 656-715.

5. Diffie W., Hellman M. E. New directions in cryptography. IEEE Transactions on Information Theory. 1976. Vol. 22, № 6. P. 644-654.

6. Ashton K. That 'Internet of Things' thing. RFID Journal. 2009. URL: https://www.rfidjournal.com/that-internet-of-things-thing

7. Atzori L., Iera A., Morabito G. The Internet of Things: a survey. Computer Networks. 2010. Vol. 54, № 15. P. 2787-2805.

8. MQTT version 5.0 : OASIS standard. Burlington : OASIS, 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

9. Shelby Z., Hartke K., Bormann C. The Constrained Application Protocol (CoAP) : RFC 7252 / IETF. 2014. URL: https://www.rfc-editor.org/rfc/rfc7252

10. Eddy W. Transmission Control Protocol (TCP) : RFC 9293 / IETF. 2022. URL: https://www.rfc-editor.org/info/rfc9293 (дата звернення: 15.05.2026).

11. Raspberry Pi documentation / Raspberry Pi Ltd. URL: https://www.raspberrypi.com/documentation/

12. Understanding the Mirai botnet / M. Antonakakis [et al.]. Proceedings of the 26th USENIX Security Symposium. Vancouver, 2017. P. 1093-1110.

13. Menezes A. J., van Oorschot P. C., Vanstone S. A. Handbook of applied cryptography. Boca Raton : CRC Press, 1996. 780 p.

14. Вишняков В. М., Чуприн В. М. Технологія безпечного обміну даними в системах Інтернету речей : рукопис. Київ : КНУБА, 2021. 18 с.

15. OpenBSD : operating system / The OpenBSD Project. URL: https://www.openbsd.org

16. PF: the OpenBSD packet filter / The OpenBSD Project. URL: https://www.openbsd.org/faq/pf/

17. Node.js : documentation / OpenJS Foundation. URL: https://nodejs.org/docs/

18. PM2 : process manager documentation. URL: https://pm2.keymetrics.io/docs/ (дата звернення: 15.05.2026).

19. Fielding R., Nottingham M., Reschke J. HTTP/1.1 : RFC 9112 / IETF. 2022. URL: https://www.rfc-editor.org/info/rfc9112

20. Fetch : living standard / WHATWG. 2024. URL: https://fetch.spec.whatwg.org

21. Lidl R., Niederreiter H. Finite fields. 2nd ed. Cambridge : Cambridge University Press, 1997. 755 p.

22. Чуприн В. М., Вишняков В. М., Пригара М. П. Генерування випадкових чисел штатними засобами хостів мережі Інтернет. Захист інформації. 2016. Т. 18, № 4. С. 323-335.

23. Flanagan D. JavaScript: the definitive guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.

24. Stallings W. Cryptography and network security: principles and practice. 7th ed. Harlow : Pearson, 2017. 768 p.

25. Бурячок В. Л., Толубко В. Б., Хорошко В. О., Толюпа С. В. Інформаційна та кібербезпека: системний підхід : підручник. Київ : ДУТ, 2015. 288 с.

26. Орлов С. Чому проблему безпеки інтернету речей виявилося так важко вирішити? CNews. 2020. URL: https://safe.cnews.ru/articles/2020-05-21_pochemu_problemu_bezopasnosti_interneta

27. Вишняков В. М., Пригара М. П., Воронін О. В. Відкрита система таємного голосування. Управління розвитком складних систем. 2014. Вип. 20. С. 110-115.

28. Вишняков В. М., Чуприн В. М., Комарницький О. О. Метод протидії незаконному впливу на виборців у системі Інтернет голосування. Безпека інформації. 2017. Т. 23, № 1. С. 7-14.

29. State of IoT 2025: number of connected IoT devices / IoT Analytics. 2025. URL: https://iot-analytics.com/number-connected-iot-devices/.