МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

СИСТЕМА ОДНОСТОРОННЬОГО ЗВ'ЯЗКУ ОДНОРАЗОВИХ СПОВІЩЕНЬ БЕЗ КЕШУВАННЯ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Владислав ОСТРАКОВ

«___»_____ 2025 р.

Керівник ст. викладач кафедри ІПЗ _____ Юрій РЕШЕТНІК

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем
_____ Юрій КОНДРАТЕНКО
«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Остраков Владислав Олександрович
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система одностороннього зв'язку одноразових сповіщень без кешування».
Керівник роботи: Решетнік Юрій Володимирович, ст. викладач кафедри ІПЗ.
(прізвище, ім’я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» червня 2025 р.

3. Очікуваний результат роботи: Планується створення чат-бота в Телеграмі з системою одностороннього зв’язку одноразових сповіщень без кешування даних на стороні користувача, де користувач зможе перейти в спеціально веб-форму і надіслати зашифроване повідомлення адміністраторам чату.

4. Перелік питань, що підлягають розробці:
- дослідження теоретичних засад;

- аналіз існуючих архітектур;
- обгрунтування вибору інструментальних засобів;
- реалізація застосунку на основі сучасних інструментів.

5. Перелік графічних матеріалів: презентація, рисунки.

Керівник роботи _____ Юрій РЕШЕТНІК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Владислав ОСТРАКОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Система одностороннього зв'язку одноразових сповіщень без кешування.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	22.12.2024	виконано
2	Аналіз предметної області та постановка задачі	23.12.2024	30.01.2025	виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем	31.01.2025	01.03.2025	виконано
4	Огляд існуючих архітектур одностороннього зв’язку для вирішення поставленої задачі	02.03.2025	01.04.2025	виконано
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2025	29.05.2025	виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	виконано
7	Корегування роботи за результатами попереднього захисту	17.05.2025	29.05.2025	виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	виконано
9	Доробка та остаточне оформлення КР	01.06.2025	10.06.2025	виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	17.06.2025	виконано

Керівник роботи _____ Юрій РЕШЕТНІК
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Здобувач _____ Владислав ОСТРАКОВ
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра
здобувача групи 402 ЧНУ ім. Петра Могили

Остраков Владислав Олександрович

“СИСТЕМА ОДНОСТОРОННЬОГО ЗВ’ЯЗКУ ОДНОРАЗОВИХ СПОВІЩЕНЬ БЕЗ КЕШУВАННЯ”

Захист інформації та своєчасне повідомлення користувачів у критичних ситуаціях є надзвичайно важливими завданнями сучасних інформаційних систем. У випадках, коли повідомлення мають бути доставлені один раз, будь-яке повторне відправлення або затримка може призвести до втрати актуальності даних або навіть до небезпечних наслідків.

Об'єктом роботи є інформаційно-комунікаційні технології, організації односпрямованих оповіщень у мережі, зокрема Telegram Bot API та супутні сервіси.

Предмет роботи є методи, алгоритми та програмно-технічні рішення для реалізації одноразової доставки повідомлень у Telegram-чат-боті без кешування, з урахуванням вимог до масштабованості, надійності та захисту даних.

Метою роботи є розробити архітектуру й реалізувати Telegram чат-бота для забезпечення одноразової системи оповіщень, яка гарантуватиме своєчасність, надійність і безпеку доставки інформації.

У першому розділі проведено аналіз існуючих рішень у сфері безпечного зв'язку комунікаційних технологій. Визначено основні вимоги до функціоналу системи, включаючи простоту використання, швидкість передачі інформації та механізми захисту даних. У другому розділі розглянуто архітектурне планування клієнтської частини та обрано технології за допомогою яких буде реалізовуватись проєкт. У третьому розділі описана структура системи, схеми взаємодіїї користувача з ботом та адміністраторами. У четвертому розділі знаходиться повна реалізація проєкту та його функціонал.

У ході дипломної роботи досліджується та реалізується процес розробки системи одностороннього зв'язку, призначеної для безпечної та конфіденційної

комунікації в умовах обмеженої безпеки. Система базується на використанні технологій одноразових сповіщень, що не зберігаються в кеші, що дозволяє гарантувати конфіденційність та захист даних користувачів, зокрема для військових служб.

Загальний обсяг роботи – 80 сторінок. Кваліфікаційна робота містить 1 додаток, 23 рисунки, та 30 джерел посилань.

ABSTRACT

to the qualification work by the student of the group 402
of Petro Mohyla Black Sea National University

Ostrakov Vladyslav

"ONE-WAY COMMUNICATION SYSTEM OF ONE-TIME NOTIFICATIONS WITHOUT CACHING"

Protecting information and notifying users in a timely manner in critical situations are extremely important tasks for modern information systems. In cases where messages must be delivered once, any resend or delay can lead to loss of data relevance or even dangerous consequences.

The object of the work is information and communication technologies, organization of unidirectional notifications in the network, in particular Telegram Bot API and related services.

The subject of the work is methods, algorithms, and software and hardware solutions for implementing one-time delivery of messages in a Telegram chatbot without caching, taking into account the requirements for scalability, reliability, and data protection.

The aim of the work is to develop an architecture and implement a Telegram chatbot to provide a one-time notification system that will guarantee timeliness, reliability, and security of information delivery.

The first section analyzes existing solutions in the field of secure communication technologies. The main requirements for the system's functionality, including ease of use, information transfer speed, and data protection mechanisms, are identified. The second section discusses the architectural planning of the client side and selects the technologies that will be used to implement the project. The third section describes the structure of the system, user interaction schemes with the bot and administrators. The fourth section describes the full implementation of the project and its functionality.

In the course of the thesis, the process of developing a one-way communication system designed for secure and confidential communication in a limited security environment is

investigated and implemented. The system is based on the use of one-time notification technologies that are not stored in the cache, which allows to guarantee the confidentiality and protection of user data, in particular for military services.

Total length of the thesis – 80 pages. The qualification work contains 1 appendix, 23 figures and 30 references.

ЗМІСТ

ВСТУП

Сьогодні цифрова комунікація охоплює всі сфери нашого життя, особливо сфери безпеки, інформування та приватності. В умовах постійного потоку інформації надзвичайно важливим є швидке, одноразове донесення критично важливих повідомлень до адресата.

Об'єктом роботи є інформаційно-комунікаційні технології, організації односпрямованих оповіщень у мережі, зокрема Telegram Bot API та супутні сервіси.

Предмет роботи є методи, алгоритми та програмно-технічні рішення для реалізації одноразової доставки повідомлень у Telegram-чат-боті без кешування, з урахуванням вимог до масштабованості, надійності та захисту даних.

Метою роботи є розробити архітектуру й реалізувати Telegram чат-бота для забезпечення одноразової системи оповіщень, яка гарантуватиме своєчасність, надійність і безпеку доставки інформації.

Основним завданням було розробити систему одностороннього зв’язку для одноразових сповіщень у формі Telegram-чат-бота. Метою проєкту було створення інструменту, здатного надсилати повідомлення лише один раз, без можливості їх повторного перегляду чи збереження у кеші, що дозволяє підвищити рівень конфіденційності переданої інформації та мінімізувати ризики несанкціонованого доступу до неї.

Було проведено дослідження основних принципів побудови одноразових повідомлень, вивчено специфіку Telegram API, обрано оптимальні підходи до реалізації логіки одностороннього зв’язку, а також здійснено пошук технологій, що дозволяють уникнути кешування або дублювання повідомлень. Окрему увагу приділено аналізу безпекових аспектів при передачі даних через Telegram-бота.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ У СФЕРІ ОДНОСТОРОННЬОГО ЗВ’ЯЗКУ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери програмного забезпечення систем одностороннього зв'язку одноразових сповіщень без кешування

Предметна сфера програмного забезпечення, пов’язаного із системами одностороннього зв’язку для одноразових сповіщень без кешування, охоплює розробку спеціалізованих рішень, які дозволяють надсилати повідомлення від одного учасника комунікації до іншого без можливості відповіді та без збереження переданої інформації в проміжних сховищах. Такі системи особливо актуальні в умовах, коли необхідно забезпечити високий рівень конфіденційності та безпеки передавання даних. Однією з головних характеристик цієї предметної області є реалізація функції одноразового перегляду, після якого повідомлення автоматично знищується або стає недоступним для користувача. Це виключає можливість повторного доступу до інформації, копіювання або збереження її у вигляді кешу на стороні клієнта.

Подібні програмні рішення стають дедалі популярнішими в сучасному цифровому середовищі, де значна кількість комунікацій відбувається в електронному форматі, а вимоги до конфіденційності постійно зростають. Зокрема, передавання одноразових кодів, важливих службових повідомлень або приватних інструкцій часто потребує механізмів, які б гарантували, що інформація буде використана лише один раз і не залишиться доступною в жодному вигляді після її прочитання. Це дозволяє зменшити ризики, пов’язані з несанкціонованим доступом до даних, витоком інформації чи перехопленням повідомлень.

Одним із найефективніших інструментів є месенджер Telegram, який має відкрите API для розробки ботів із широким функціоналом. Завдяки підтримці асинхронної комунікації, можливості автоматичного видалення повідомлень та налаштуванню обмежень на рівні доступу, Telegram дозволяє реалізовувати подібні системи з високим рівнем безпеки та надійності. Вибір саме цієї платформи

пояснюється також її популярністю серед користувачів та зручністю інтеграції з іншими цифровими сервісами.

1.2 Огляд останніх публікацій та існуючих аналогічних систем

У багатьох публікаціях, присвячених цифровій безпеці та обміну повідомленнями, розглядається необхідність використання інструментів, які мінімізують можливість витоку або несанкціонованого доступу до інформації. Окрему увагу дослідники приділяють месенджерам, які реалізують функції самознищення повідомлень, наскрізного шифрування, обмеженого часу доступу та заборони збереження.

Telegram, Signal, AWS Wickr це лише декілька із сучасних застосунків, що частково або повністю реалізують концепцію одноразовості повідомлень.

Крім загальнодоступних месенджерів, також можна знайти реалізації подібних систем у форматі окремих ботів або мікросервісів. Наприклад, є багато статей на GitHub, де розробники описують створення Telegram-ботів із функціями одноразових сповіщень для внутрішнього використання в компаніях або стартапах. У таких рішеннях зазвичай використовується Telegram API, Python або Node.js, та тимчасове сховище даних без постійного збереження [1].

Архітектура таких ботів і подібних систем зазвичай включає просту логіку:

- ввід повідомлення адміністратором або іншим користувачем;
- генерація одноразового посилання або коду;
- збереження повідомлення у тимчасовій базі даних (Redis);
- передача отримувачу;
- знищення повідомлення після відкриття або по таймеру.

На етапі формування повідомлення відправник вводить текст або дані, які мають бути передані. У більшості систем ці дані одразу шифруються або обгортаються у спеціальну тимчасову логіку (наприклад, додається унікальний ID повідомлення, час життя або контроль доступу). Якщо використовується Telegram-бот, то це може бути просте текстове повідомлення, яке зберігається лише у RAM

або у базі типу Redis з автоматичним знищенням через певний час.

Обробка на сервері може включати перевірку прав доступу, таймер самознищення або обмеження кількості переглядів. У класичному випадку повідомлення можна переглянути лише один раз після чого воно або видаляється фізично, або мітиться як прочитане та недоступне повторно.

Передача повідомлення виконується через захищений канал, наприклад, TLS-шифрування, HTTP-з'єднання або за допомогою Telegram API, яке також підтримує наскрізне шифрування в секретних чатах. Протягом передачі важливо уникати кешування як на стороні клієнта, так і на стороні сервера [1, 2, 4].

1.3 Огляд та аналіз наявних рішень систем одноразових сповіщень

1.3.1 Наявні сервіси

У процесі вивчення предметної області одноразових сповіщень без кешування було проаналізовано кілька реальних реалізацій, які вже функціонують на ринку та мають широку базу користувачів. Станом на 2025 рік найбільш показовими прикладами стали месенджери Telegram, Signal та AWS Wickr, які реалізують різні підходи до забезпечення конфіденційності, контролю за повідомленнями та знищення даних після доставки. Нижче вони розглянуті більш детально.

1. Telegram реалізує систему одноразових повідомлень у вигляді секретних чатів, де використовується наскрізне шифрування з самознищенням повідомлень. Повідомлення в таких чатах існують лише на пристроях учасників і не зберігаються на серверах Telegram. Функція самознищення дозволяє налаштувати таймер після відкриття повідомлення воно автоматично зникає через заданий проміжок часу від однієї секунди до тижня. Також Telegram дозволяє створювати ботів, які можуть самостійно видаляти повідомлення одразу після їх прочитання, що ідеально підходить для систем одноразових сповіщень. Telegram використовує власний протокол шифрування MTProto, який поєднує симетричне шифрування AES-256,

RSA-2048 та обмін ключами за схемою Diffie-Hellman. Попри те, що більшість чатів у Telegram не є наскрізно зашифрованими, саме секретні чати відповідають вимогам безпечного одноразового повідомлення [2].

Рисунок 1.1 – Логотип застосунку та інтерфейс застосунку

2. Signal це open-source месенджер, який вважається одним із найнадійніших інструментів для захищеного спілкування. Однією з ключових функцій є disappearing messages, повідомлення, що автоматично зникають після заданого часу. Окрім цього, Signal використовує протокол Signal Protocol, який забезпечує як forward secrecy, так і backward secrecy. Це означає, що навіть якщо один з ключів буде скомпрометований, інші повідомлення залишаться зашифрованими. У Signal можливо налаштувати зникнення повідомлення як для приватних, так і для групових чатів. Таймер встановлюється як на рівні чату, так і окремо для кожного повідомлення. Хоча система не обмежує кількість переглядів, як у Telegram, знищення повідомлення після встановленого часу гарантує, що воно не буде

збережене чи кешоване. Signal також попереджає користувачів про створення скріншотів і в Android-версії навіть блокує можливість зробити їх [3].

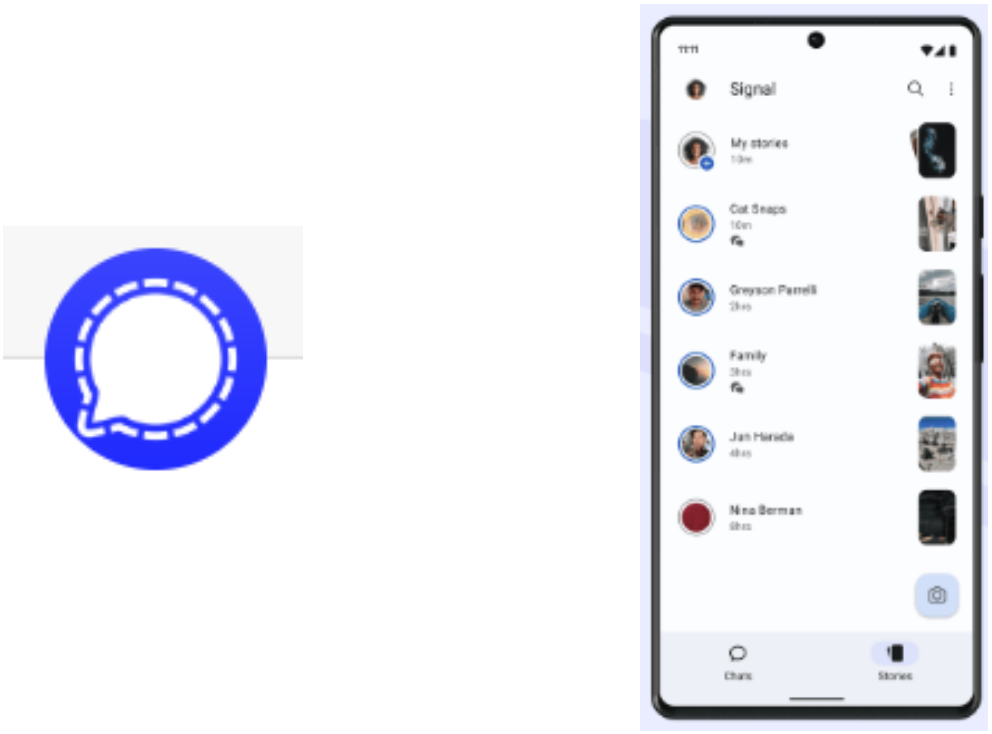

Рисунок 1.2 – Логотип застосунку та інтерфейс застосунку

3. AWS Wickr – це сучасна захищена платформа обміну повідомленнями, створена для корпоративного та урядового використання. AWS Wickr є комерційним рішенням, яке пропонує розширені функції безпеки, масштабованість, інтеграцію з хмарною інфраструктурою Amazon та повну відповідність сучасним стандартам збереження й обробки даних. Система підтримує наскрізне шифрування для всіх типів повідомлень. Усі вони шифруються на пристрої відправника та розшифровуються лише на пристрої одержувача. Унікальною функцією AWS Wickr є автоматичне самознищення повідомлень після прочитання або через заданий інтервал часу. Це дозволяє будувати системи одноразового інформування, де дані не залишаються у кеші та не зберігаються на жодному сервері після їх використання. Ще однією важливою функцією AWS Wickr є можливість повного контролю за політиками збереження, де адміністратор

організації може налаштувати таймери видалення, заборонити або дозволити знімки екрана, запис розмов чи переадресацію повідомлень. Крім того, для великих організацій важливою є інтеграція з іншими сервісами AWS, що дозволяє масштабувати систему та забезпечити відповідність нормам безпеки, як-от HIPAA, FINRA, GDPR, CJIS тощо. AWS Wickr також підтримує анонімність, не зберігає метадані, не має централізованого журналу активності користувачів, що дозволяє забезпечити високу конфіденційність у спілкуванні навіть у великих компаніях [5, 6].

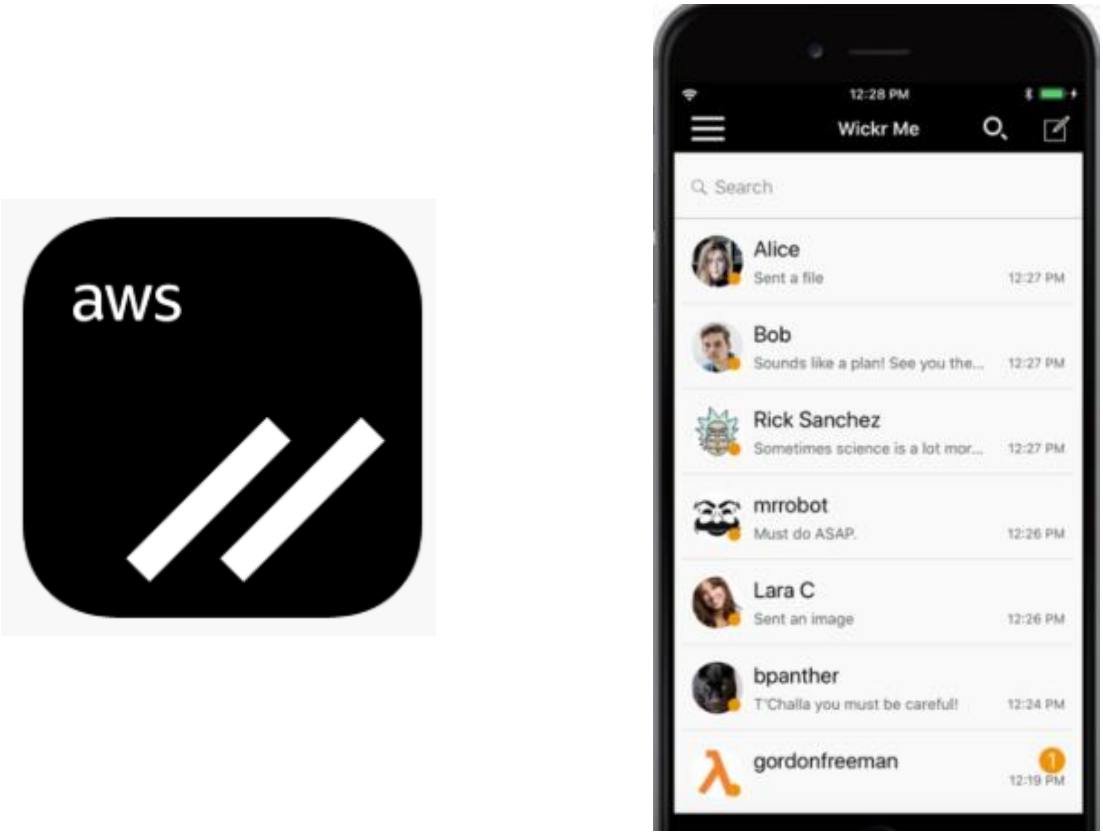

Рисунок 1.3 – Логотип застосунку та інтерфейс застосунку

1.3.2 Порівняння Telegram, Signal та AWS Wickr

Кожна з них має свої особливості, переваги та обмеження, що визначає сферу їх застосування та рівень придатності для створення сервісу одностороннього інформування.

Telegram – це популярна платформа обміну повідомленнями, яка дозволяє створювати ботів, канали та групи. Її перевагою є гнучкий API, який дає можливість створювати кастомізованих чат-ботів, зокрема для розсилки одноразових повідомлень. Однак важливо зазначити, що Telegram не забезпечує наскрізного шифрування за замовчуванням у звичайних чатах або ботах. Навіть у секретних чатах, де шифрування є доступним, дані можуть тимчасово зберігатися у кеші пристрою або сервері, залежно від конфігурації. Боти, як правило, функціонують у контексті хмарної інфраструктури Telegram, і хоч вони можуть видаляти повідомлення, це не є гарантією абсолютного знищення без сліду. Проте Telegram зручний і простий у впровадженні.

Signal є одним з найвідоміших месенджерів, орієнтованих на конфіденційність. Він підтримує наскрізне шифрування за замовчуванням для всіх повідомлень і не зберігає метадані. Signal також дозволяє вмикати таймери для зникнення повідомлень, що частково реалізує функцію одноразовості. Проте, на відміну від AWS Wickr, у Signal немає відкритого API для створення повноцінних ботів або серверних сповіщень, що ускладнює інтеграцію для задач одностороннього інформування. Таким чином, хоча Signal є ідеальним для особистого захищеного спілкування, він не настільки гнучкий для реалізації кастомізованих систем повідомлень.

AWS Wickr позиціонується як корпоративне рішення для захищеного обміну інформацією. На відміну від Telegram і Signal, ця система глибоко інтегрована з хмарною інфраструктурою Amazon Web Services, що дозволяє адмініструвати користувачів, керувати політиками збереження повідомлень, а також автоматизувати процеси безпеки. Wickr забезпечує повну відсутність кешування, підтримує автоматичне видалення повідомлень, і має механізми захисту від збору метаданих. Повідомлення шифруються на клієнтському боці та не можуть бути відновлені після знищення. Крім того, ця система дозволяє керувати таймерами самознищення на рівні адміністратора, що підвищує контроль за інформаційним потоком. Вона ідеально підходить для реалізації систем одноразових сповіщень у

середовищах з високими вимогами до безпеки.

1.4 Постановка задачі

У межах кваліфікаційної роботи поставлено завдання – розробити систему одностороннього зв’язку для одноразових сповіщень без кешування, реалізовану у вигляді Telegram-бота.

Згідно з оглядом існуючих рішень, таких як AWS Wickr, Signal та Telegram, було встановлено, що найбільш придатною платформою для реалізації односторонньої комунікації є саме Telegram. Це зумовлено тим, що Telegram надає документоване API для створення ботів, дозволяє надсилати повідомлення, видаляти їх автоматично через певний час або після прочитання, а також має велику аудиторію та просту інфраструктуру для швидкого розгортання рішення. Хоча Telegram не гарантує абсолютної відсутності кешування, але завдяки правильній конфігурації бота та обмеженню функціональності з боку користувача, наприклад, заборона відповіді, обмеження пересилання, можна наблизитися до моделі одноразової доставки.

1.4.1 Планування технологій для реалізації чат-боту

Для реалізації поставленої задачі обрано мову програмування Python, оскільки вона має велику кількість готових бібліотек для роботи з API Telegram, зокрема, python-telegram-bot, aiogram, pyrogram, а також зручний синтаксис, який дозволяє швидко створювати та тестувати веб-сервіси. Python також підтримує роботу з асинхронними викликами, що є критично важливим при побудові ботів, які повинні оперативно реагувати на події.

Також буде використано локальну базу даних або in-memory рішення для збереження станів користувача, якщо потрібно відстежувати, чи отримував користувач певне повідомлення. Однак жоден з вмістів повідомлень не буде збережений це принципова умова реалізації. Крім того, буде застосовано обмеження на повторну доставку, тобто повідомлення знищуватиметься одразу

після його надсилання або після одержання підтвердження прочитання [11].

Система також передбачає реалізацію таких функціональних компонентів:

– адміністративна панель або конфігураційний файл для задання змісту повідомлень;

– механізм одноразового генератора токенів або кодів, які дають змогу користувачеві отримати повідомлення лише один раз;

– логіка самознищення повідомлення після взаємодії користувача з ботом;

– захист від повторних звернень або зловживань через збереження ID користувача або тимчасового ключа.

Для тестування та розгортання планується використання хмарного хостингу, а також віртуального середовища Python. Telegram API працює через HTTPS-запити, тому також може бути використано бібліотеку requests або aiohttp для низькорівневої взаємодії з сервером, якщо не буде обрано готове SDK [2, 4, 9].

1.4.2 Вимоги до програмної реалізації

У процесі розробки Telegram-бота для одноразової доставки повідомлень висуваються певні вимоги до програмної частини системи, які необхідно дотриматися для забезпечення її коректної та безпечної роботи. Перш за все, програмна реалізація повинна гарантувати стабільну взаємодію з Telegram API, зокрема забезпечувати безперебійний прийом запитів від користувачів і своєчасне надсилання відповідей. Оскільки бот працює у режимі одноразової комунікації, критично важливо, щоб повідомлення не зберігались у жодному вигляді після їх надсилання, а також щоб не було можливості повторно їх отримати або переглянути.

Важливим аспектом є використання надійної логіки контролю доступу до повідомлень. Це означає, що бот повинен вміти перевіряти, чи має користувач право на отримання конкретного повідомлення, а також ідентифікувати, чи вже було ним відкрито вміст. З цією метою реалізується система токенів або одноразових кодів, які перевіряються на етапі взаємодії з ботом. Після отримання

повідомлення бот повинен самостійно його видалити або зробити його недоступним, аби забезпечити концепцію одноразовості.

Крім того, реалізація має враховувати питання захисту від зловживань і повторних спроб доступу. Для цього доцільно зберігати службову інформацію, таку як ідентифікатори користувачів, час запиту або ID повідомлення, однак без збереження самого тексту повідомлення. Система повинна вміти швидко реагувати на події, тому бажано використовувати асинхронну архітектуру, яка дозволяє обробляти кілька запитів одночасно.

Оскільки планується розгортання бота у хмарному середовищі, програмне забезпечення повинно бути адаптоване до роботи у таких умовах, а також підтримувати розмежування середовищ розробки, тестування і продакшну. Це означає, що конфігураційні параметри, ключі доступу та інші чутливі дані не повинні бути зашиті в код, а мають зчитуватись із змінних середовища або зовнішніх файлів конфігурації.

На рівні коду очікується структурованість, модульність і наявність документації. Код повинен бути написаний у відповідності до стилістичних стандартів Python (наприклад, PEP 8), з коментарями, які пояснюють логіку ключових функцій. Це дозволить легше підтримувати та масштабувати систему у майбутньому.

Висновки до розділу 1

У розглянутому розділі було детально проаналізовано основні аспекти розробки системи одностороннього зв’язку для одноразових сповіщень без кешування, зокрема, розглянуто специфіку створення Telegram-бота для цієї мети. Спершу було розглянуто предметну сферу програмного забезпечення, що включає створення системи, яка забезпечує безпечну та ефективну доставку повідомлень, які неможливо зберегти чи повторно отримати.

Також було здійснено огляд останніх публікацій та існуючих аналогічних систем таких як Telegram, Signal та AWS Wickr, що дозволило порівняти наявні

рішення, визначити їх переваги та обмеження, а також обрати найбільш відповідну платформу для реалізації нашої системи. Серед них Telegram був обраний через доступне API, можливість інтеграції та популярність серед користувачів.

Було сформульовано постановку задачі, в якій зазначено, що основною метою розробки є створення Telegram-бота, який забезпечить одноразову доставку повідомлень без їх збереження. У цьому контексті також були визначені основні технології та інструменти для реалізації проекту, зокрема мова програмування Python, а також бібліотеки для роботи з Telegram API. Також були окреслені принципи побудови системи та її ключові компоненти.

Вимоги до програмної реалізації детально описують необхідні умови для безперебійної роботи системи, такі як надійна взаємодія з Telegram API, безпека обробки запитів, реалізація механізмів одноразового доступу до повідомлень, використання асинхронної архітектури для забезпечення ефективності роботи, а також забезпечення безпечного зберігання конфіденційних даних. Важливою вимогою є забезпечення відсутності кешування та надійного самознищення повідомлень після їх прочитання або після певного часу.

Отже, в результаті розгляду цього розділу було створено чітке уявлення про те, як має виглядати система для одноразової доставки повідомлень, які технології та принципи повинні бути використані для її реалізації, а також про важливі аспекти безпеки та ефективності системи. Розробка цієї системи сприятиме створенню безпечного та ефективного інструменту для передачі важливих повідомлень, що є актуальним у багатьох сферах діяльності.

2 АРХІТЕКТУРНІ ПІДХОДИ, ТЕХНОЛОГІЇ, ЩО ВИКОРИСТОВУЮТЬСЯ ДЛЯ РОЗРОБКИ ЧАТ-БОТУ

2.1 Архітектурні підходи

Розробка чат-ботів для одноразових сповіщень вимагає вибору правильних архітектурних підходів, які дозволять забезпечити ефективну, масштабовану та безпечну роботу системи. Оскільки чат-боти мають специфічні вимоги до масштабованості та обробки даних в реальному часі, для їх реалізації найчастіше використовуються мікросервісна та подієва архітектури. Розглянемо ці архітектурні підходи більш детально.

2.1.1 Мікросервісна архітектура

Мікросервісна архітектура є одним із найбільш популярних підходів у розробці сучасних вебдодатків і чат-ботів, особливо коли йдеться про масштабовані системи з високим навантаженням. Вона передбачає розбиття великої системи на декілька незалежних, малих сервісів, кожен з яких відповідає за виконання певної функції. Кожен сервіс є автономним і може розвиватися окремо від інших, що значно полегшує обслуговування, оновлення та масштабування системи.

Мікросервісна архітектура чат-ботів дозволяє розподілити функціональність на окремі компоненти, кожен із яких працює самостійно і при потребі легко масштабується. Наприклад, коли навантаження на відправлення повідомлень зростає, достатньо підсилити тільки відповідний сервіс, не зачіпаючи інші частини системи. Завдяки такому підходу можна також використовувати різні мови програмування для різних завдань: хтось пише систему взаємодії з Telegram API на Python, а обробку повідомлень у реальному часі – на Go чи Node.js. Це дає змогу обирати найпридатніші інструменти та швидко адаптуватися до нових вимог.

Окрім того, ізольованість мікросервісів спрощує процес оновлень та підтримки: можна змінити логіку роботи з одноразовими токенами або внести корекції в алгоритми обробки повідомлень без простою всього чат-бота. В

окремому модулі легко реалізувати генерацію, валідацію й видалення таких токенів, гарантуючи їх послідовне використання. А ще є сенс виділити окремий сервіс для збирання логів і метрик – це допоможе відстежувати роботу системи, аналізувати статистику й забезпечувати безпеку через аудит дій користувачів. Такий модульний підхід робить чат-бот не лише гнучким і надійним, а й простішим у подальшій підтримці та розвитку [7].

2.1.2 Event-Driven архітектура

Подієва архітектура Event-Driven Architecture, EDA є одним із важливих підходів для розробки чат-ботів, особливо тих, які потребують високої взаємодії в реальному часі. В цій архітектурі система реагує на події – зміни стану або настання конкретних умов. Наприклад, для чат-бота це може бути подія, коли користувач відправляє повідомлення, або коли бот отримує одноразовий код для доступу до сповіщення.

Event-Driven архітектура дозволяє чат-ботам обробляти події асинхронно, завдяки чому система справляється з великою кількістю запитів без затримок і миттєво реагує на надходження повідомлень. У ній легко додавати або змінювати типи подій, не змінюючи всю структуру, а чітке визначення ключових подій, наприклад отримання повідомлення або створення одноразового токена, допомагає розподіляти відповідальність між обробниками. Для надійного обміну інформацією зазвичай використовують черги повідомлень, як-от RabbitMQ або Kafka, що забезпечує централізовану взаємодію між компонентами. Кожний обробник події виконує свою задачу, наприклад надсилає відповідь користувачеві після обробки запиту, що робить всю систему гнучкою, масштабованою і високоефективною [8].

2.1.3 Система зберігання станів у реальному часі

Для розробки чат-ботів, що працюють з одноразовими повідомленнями, критично важливо використовувати ефективні системи зберігання даних, які гарантують швидке і безпечне збереження станів користувачів в реальному часі. Це

необхідно для таких завдань, як обробка одноразових токенів, зберігання ідентифікаторів користувачів і часу взаємодії з ботом, а також забезпечення високої швидкості доступу до цих даних. Оскільки в системах одноразових повідомлень дані не повинні зберігатися тривалий час, необхідно застосовувати технології, які дозволяють зберігати їх лише на короткий період і оперативно очищати після завершення сеансу. Одними з найкращих варіантів для цього є Redis.

Redis – це система зберігання даних у пам'яті, яка підтримує різноманітні типи структур даних, зокрема рядки, списки, множини, хеші та багато інших. Вона має здатність до швидкої обробки запитів, що робить її ідеальним варіантом для застосувань, де важлива низька затримка та висока швидкість доступу до даних. Redis дозволяє зберігати тільки необхідну інформацію, наприклад, ідентифікатори користувачів або токени, з фіксацією часу запиту. Після того як повідомлення було надіслано, ці дані можуть бути автоматично видалені завдяки можливостям Redis для встановлення терміну придатності для кожного елемента. Це ідеально підходить для чат-ботів, де повідомлення не повинні зберігатися після того, як вони були надіслані, для запобігання витоку інформації.

Однією з важливих переваг є її здатність працювати в умовах високих навантажень та забезпечувати високу доступність. Вона ідеально підходить для сценаріїв, коли потрібно швидко зберігати дані і забезпечити зворотну взаємодію користувача з ботом в реальному часі. Наприклад, коли користувач отримує одноразове повідомлення, бот може згенерувати унікальний токен, який буде збережений у Redis. Після використання токен автоматично видаляється, що відповідає вимогам щодо одноразовості та конфіденційності повідомлень [9].

2.2 Технології для розробки чат-боту

2.2.1 Telegram API

Telegram API є потужним інструментом для розробки чат-ботів та інтеграції їх з платформою Telegram. Це програмний інтерфейс, який надає можливість створювати ботів, що можуть взаємодіяти з користувачами через Telegram. З

використанням API можна реалізувати різноманітні функції, починаючи від простого надсилання повідомлень до складніших задач, таких як інтерактивні опитування або обробка мультимедійного контенту. Однією з основних переваг Telegram API є його зручність і можливість забезпечення безпечного та ефективного зв’язку з користувачами, що робить його чудовим вибором для створення чат-ботів.

Однією з основних особливостей Telegram API є підтримка двох основних механізмів для отримання повідомлень: webhooks та polling. Webhooks дозволяють чат-боту отримувати повідомлення про події в реальному часі через HTTP-запити, які надсилаються сервером Telegram на вказану URL-адресу кожного разу, коли нове повідомлення або інша подія відбувається в чаті з ботом. Цей метод є дуже зручним для ботов, які потребують миттєвої реакції, оскільки дані надходять безпосередньо до сервера бота. З іншого боку, polling дозволяє боту періодично запитувати сервер Telegram на наявність нових повідомлень, що може бути корисним для менш чутливих до затримок задач.

Telegram API також дозволяє працювати з різними типами повідомлень, включаючи текстові, мультимедійні і кнопки для створення інтерактивних взаємодій. Наприклад, можна створювати кнопки, за допомогою яких користувачі можуть відповідати на запитання або вибирати варіанти без необхідності вводити текст вручну. Така інтерактивність дає змогу створювати більш складні і цікаві користувацькі інтерфейси для чат-ботів.

Ще однією важливою перевагою Telegram API є наявність потужних механізмів для забезпечення безпеки і конфіденційності. Telegram використовує шифрування повідомлень як в каналі зв’язку, так і при зберіганні даних на серверах. Крім того, Telegram дозволяє використовувати різні рівні доступу до бота, що дозволяє налаштовувати його функціонування таким чином, щоб тільки авторизовані користувачі могли взаємодіяти з ним. Це є особливо важливим для чат-ботів, які працюють з конфіденційною інформацією або вимагають захисту від несанкціонованого доступу.

API Telegram також надає широкий набір функцій для роботи з командними запитами, такими як /start, /help та інші. Це дозволяє користувачам легко взаємодіяти з ботом, отримувати інструкції або запускати певні дії. Можливість обробки команд і запитів дозволяє створювати гнучкі й інтуїтивно зрозумілі інтерфейси для чат-ботів, що значно підвищує їх ефективність і зручність використання.

Одним із найпоширеніших випадків використання Telegram API є створення чат-ботів для автоматизації сповіщень. Такі боти можуть надсилати одноразові або повторювані повідомлення користувачам, здійснювати моніторинг і реагувати на події в реальному часі. У контексті одноразових повідомлень важливо, щоб система була налаштована на автоматичне видалення повідомлень після їх відправки, що дозволяє забезпечити конфіденційність інформації та її захист [2, 10].

2.2.2 Python і бібліотеки для чат-ботів

Python є однією з найпопулярніших мов програмування для створення чат-ботів завдяки своїй простоті, гнучкості та широкій екосистемі бібліотек. Оскільки Python має багатий набір інструментів для роботи з веб-технологіями та обробкою даних, він є ідеальним вибором для розробки чат-ботів, зокрема для Telegram. У Python є кілька бібліотек, які дозволяють швидко створювати чат-боти та інтегрувати їх з різними платформами, включаючи Telegram. Розглянемо найбільш популярні бібліотеки, які використовуються для створення чат-ботів.

python-telegram-bot це одна з найпопулярніших бібліотек для розробки чат-ботів для Telegram. Вона надає зручний інтерфейс для взаємодії з Telegram API і дозволяє розробникам створювати функціональні боти без необхідності безпосередньо працювати з HTTP-запитами або низькорівневими деталями API. Ця бібліотека дозволяє легко обробляти повідомлення, додавати команди, створювати кнопки, працювати з мультимедійним контентом та організовувати взаємодію з користувачами. python-telegram-bot підтримує багато зручних функцій, таких як асинхронне оброблення запитів в поєднанні з бібліотекою asyncio, обробка команд

наприклад, /start, /help, а також можливість підключення різних хендлерів для обробки різних типів повідомлень. Вона також дозволяє реалізовувати складніші логічні структури чат-ботів, використовуючи, наприклад, машини станів або обробку запитів на кількох рівнях. З цією бібліотекою розробка чат-бота стає набагато простішою та швидшою.

aiogram це асинхронна бібліотека для створення чат-ботів, яка побудована на основі асинхронного програмування за допомогою Python бібліотеки asyncio. Вона є однією з найпопулярніших альтернатив для розробки масштабованих чат-ботів, особливо коли йдеться про високі навантаження та одночасну обробку великої кількості запитів. Завдяки асинхронному підходу, бот може одночасно обробляти кілька запитів від користувачів без блокування основного потоку, що значно підвищує продуктивність. aiogram ідеально підходить для проектів, де потрібно обробляти величезну кількість запитів в реальному часі, зокрема для систем із високими вимогами до швидкості реакції, таких як чат-боти для одноразових сповіщень. Вона дозволяє створювати безшовні інтерактивні процеси, такі як обробка повідомлень у режимі реального часу, що є критично важливим для чат-ботів із одноразовими повідомленнями [4, 14].

Flask і FastAPI це веб фреймворки для Python, які дозволяють створювати вебдодатки і API для інтеграції з чат-ботами. Обидва фреймворки є популярними серед розробників завдяки своїй простоті, високій продуктивності та зручності для створення RESTful API. Вони дозволяють організувати додаткові функції для взаємодії з іншими сервісами або для зберігання даних.

Flask є більш легким та простим у використанні фреймворком, який надає базовий набір функцій для створення вебдодатків. Він ідеально підходить для невеликих проектів, де не потрібна висока продуктивність або складна архітектура [12].

FastAPI, у свою чергу, є більш новим фреймворком, який орієнтований на високу продуктивність і підтримує асинхронне програмування. Завдяки цьому він дозволяє створювати дуже швидкі і ефективні API для чат-ботів, що є дуже

важливим для додатків, які повинні працювати з високими навантаженнями та обробляти численні запити одночасно [13].

Оскільки чат-боти часто потребують взаємодії з іншими сервісами або базами даних для зберігання тимчасових даних (наприклад, одноразових токенів), створення API за допомогою Flask або FastAPI є дуже корисним. Вони дозволяють швидко створювати кінцеві точки для взаємодії між ботом і іншими частинами системи, забезпечуючи гнучкість та масштабованість.

2.2.3 Redis і кешування даних

Redis є однією з найбільш популярних баз даних для роботи з даними, що потребують швидкого доступу та обробки в режимі реального часу. Оскільки Redis є базою даних, яка зберігається в пам'яті (in-memory), вона пропонує неймовірно високу швидкість читання та запису, що робить її ідеальним рішенням для зберігання тимчасових даних, таких як одноразові токени, сесії користувачів або маркери часу.

Для системи одноразових сповіщень Redis може бути використаний для тимчасового зберігання токенів або ID користувачів, що дозволяє оперативно перевіряти статус доступу до повідомлень. Перевагою Redis є те, що дані зберігаються в оперативній пам'яті, що забезпечує мінімальні затримки при отриманні інформації та дозволяє здійснювати швидку перевірку прав доступу. Після того, як одноразове повідомлення було переглянуте або після того, як сплинув термін дії токену, дані можуть бути автоматично видалені, що також відповідає вимогам конфіденційності.

Крім того, Redis має підтримку різних структур даних, таких як строки, списки, множини, хеші, що дозволяє зберігати дані у зручному форматі в залежності від потреб. Наприклад, для зберігання ID користувачів можна використовувати Redis хеші, а для обробки одноразових токенів – списки чи множини. Таке зберігання дозволяє максимально ефективно працювати з малими обсягами даних, що є необхідним для даної системи.

Однією з головних переваг Redis є підтримка TTL (Time To Live), що дозволяє налаштувати автоматичне видалення даних через певний проміжок часу. Це особливо корисно для одноразових токенів, оскільки вони повинні бути доступні лише обмежений час, а потім автоматично видаляються з пам'яті [9].

2.2.4 Асинхронні технології

Для розробки високопродуктивних чат-ботів з низькою затримкою важливо використовувати асинхронні технології. Асинхронне програмування дозволяє обробляти запити паралельно, без блокування основного потоку виконання, що значно підвищує швидкість та ефективність роботи бота.

Asyncio це стандартна бібліотека Python для асинхронного програмування, яка дозволяє ефективно управляти асинхронними операціями. За допомогою asyncio можна запускати корутини (функції, які виконуються асинхронно) та управляти подіями в програмі без необхідності створювати окремі потоки чи процеси. Це дозволяє значно зменшити накладні витрати на ресурси системи, оскільки асинхронні операції використовують одне ядро для виконання кількох задач одночасно, зберігаючи ефективність навіть при високих навантаженнях. Асинхронний підхід особливо важливий для чат-ботів, оскільки вони можуть обробляти велику кількість запитів одночасно. Наприклад, при використанні асинхронного програмування можна обробляти кілька запитів від різних користувачів одночасно, не блокуючи виконання програми для кожного окремого користувача. Це забезпечує швидку реакцію бота на команди та дозволяє ефективно працювати з великою кількістю запитів [14].

Aiohttp це асинхронна бібліотека Python для роботи з HTTP-запитами, яка дозволяє розробляти вебдодатки та API, що працюють асинхронно. Ця бібліотека є особливо корисною для чат-ботів, оскільки вона дозволяє обробляти запити від користувачів без блокування основного потоку. aiohttp дозволяє реалізовувати ефективні веб-сервери та клієнти, які можуть обробляти тисячі одночасних підключень, що є важливим для чат-ботів із високими вимогами до продуктивності.

Використовуючи aiohttp, чат-бот може надсилати та отримувати повідомлення без затримок, що особливо важливо для реалізації системи одноразових сповіщень, де кожна операція має бути виконана швидко та без помилок. Асинхронне оброблення HTTP-запитів дозволяє знизити навантаження на сервер, оскільки він може обробляти кілька запитів одночасно, не чекаючи відповіді від кожного з них. Це дозволяє чат-боту швидко реагувати на дії користувачів і відправляти сповіщення у реальному часі [15].

2.2.5 Шифрування

Шифрування є важливим аспектом забезпечення безпеки в системах передачі даних, зокрема в чат-ботах, що працюють з одноразовими повідомленнями. Для цього зазвичай застосовуються різні методи шифрування, які дозволяють гарантувати конфіденційність і цілісність даних під час їх передачі та зберігання.

Основним способом захисту даних під час передачі між клієнтами та сервером є використання TLS (Transport Layer Security) або його попередника SSL (Secure Sockets Layer). Ці протоколи забезпечують шифрування з’єднання, що гарантує, що передані дані не можуть бути перехоплені або змінені третіми особами. Telegram API підтримує використання TLS/SSL, що дозволяє чат-ботам безпечно взаємодіяти з платформою без ризику витоку чутливих даних. Для цього всі з'єднання між клієнтами та серверами Telegram шифруються через HTTPS, що використовує TLS/SSL для безпечної передачі даних. Це захищає не тільки текстові повідомлення, але й інші дані, такі як файли та медіа, які можуть передаватися через API.

Крім стандартного шифрування під час передачі, важливо також застосовувати додаткові методи шифрування для захисту одноразових токенів та повідомлень, що використовуються в чат-ботах. Одноразові токени служать для підтвердження автентичності користувача та доступу до конкретних повідомлень, тому вони повинні бути захищені, щоб запобігти несанкціонованому доступу.

Для шифрування одноразових токенів і повідомлень у системі можна використовувати такі підходи:

Симетричне шифрування. Цей метод передбачає використання одного і того ж ключа для шифрування та дешифрування повідомлень. Протокол AES (Advanced Encryption Standard) є найпоширенішим методом симетричного шифрування і застосовується для захисту даних у багатьох комунікаційних системах. В Python для цього можна використовувати бібліотеку cryptography, яка надає інтерфейси для роботи з AES і іншими алгоритмами шифрування.

Асиметричне шифрування. Цей метод використовує пару ключів – публічний і приватний. Публічний ключ використовується для шифрування даних, а приватний – для їх дешифрування. Асиметричне шифрування є більш безпечним у випадках, коли потрібно передавати дані через незахищені канали, оскільки навіть якщо публічний ключ потрапить у чужі руки, він не дозволить розшифрувати повідомлення без приватного ключа. Бібліотека PyCryptodome в Python надає інструменти для роботи з асиметричним шифруванням, зокрема з алгоритмами RSA або ECC (Elliptic Curve Cryptography).

Telegram також використовує end-to-end шифрування для своїх секретних чатів. Це означає, що лише відправник і отримувач можуть бачити вміст повідомлення, а Telegram не має доступу до нього. Для шифрування повідомлень Telegram використовує алгоритм MTProto. Цей алгоритм поєднує симетричне шифрування для передачі даних і асиметричне для обміну ключами, що забезпечує високий рівень безпеки та конфіденційності [1, 4].

Висновки до розділу 2

Розробка чат-боту для одноразових сповіщень є комплексним процесом, що вимагає ретельного вибору архітектурних підходів та технологій для забезпечення ефективної роботи, безпеки та зручності користування. У цьому контексті мікросервісна архітектура є оптимальним вибором, оскільки вона дозволяє розділити функціональність на окремі сервіси, що полегшує масштабування та

покращує підтримку системи. Підхід Event-Driven архітектури дозволяє оптимізувати обробку подій та зменшити затримки при передачі повідомлень.

Інструменти, такі як Telegram API, Python та різні бібліотеки, забезпечують ефективну інтеграцію з платформою, полегшуючи створення чат-ботів і їхнє функціонування в реальному часі. Використання Redis для кешування даних дозволяє швидко зберігати необхідні дані без зайвих затримок, а асинхронні технології, як asyncio та aiohttp, забезпечують високу продуктивність та масштабованість системи.

Забезпечення безпеки даних є одним із найважливіших аспектів, зокрема через застосування шифрування TLS/SSL для захисту з’єднання з Telegram API, а також додаткових методів шифрування для одноразових токенів і повідомлень. Оновлення програмного забезпечення та регулярні перевірки на вразливості допомагають зберегти високий рівень безпеки.

Таким чином, правильно підібрані технології та архітектурні підходи дозволяють створити безпечний, масштабований та ефективний чат-бот для одноразових сповіщень, що відповідає вимогам до швидкості, надійності та захищеності даних.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних та структури системи

Telegram-бот виконує функцію посередника між користувачем та адміністратором системи, однак не зберігає повідомлення і не підтримує діалогової комунікації. Його основне функціональне призначення полягає в наданні можливості користувачеві залишити зашифроване повідомлення через вебінтерфейс, яке потім надсилається безпосередньо адміністраторам у Telegram. Особливістю цієї системи є односторонність зв’язку, користувач не отримує відповіді в межах того ж самого сеансу або каналу. Це дозволяє повністю уникнути створення історії переписки, кешування, або збереження будь-яких повідомлень у чаті, що забезпечує максимальний рівень конфіденційності та безпеки для обох сторін. Користувач не взаємодіє з ботом у класичному розумінні – він не надсилає текстові повідомлення напряму в чат, а натомість отримує посилання на спеціальну веб-форму. Таким чином, Telegram-бот виконує роль навігатора до захищеного каналу надсилання, де вже відбувається шифрування повідомлення на стороні користувача.

Серед основних вхідних даних, що використовуються в системі, ключовим є текст повідомлення. Це довільний текст, який користувач вводить у веб-формі. Повідомлення може містити як контактні дані, так і опис проблеми, звернення або іншу інформацію, яка, на думку користувача, повинна бути доставлена адміністраторам системи. Особливість полягає в тому, що цей текст ще до надсилання до сервера шифрується безпосередньо у браузері користувача за допомогою публічного ключа RSA, що є доступним у коді сторінки.

Другим важливим вхідним елементом є публічний ключ, що генерується і зберігається на сервері, а під час завантаження вебсторінки автоматично вбудовується у HTML-шаблон. Саме завдяки йому можливо реалізувати асиметричне шифрування, за якого повідомлення може бути зашифроване будь-

ким, але розшифроване лише за допомогою приватного ключа, що знаходиться у захищеному середовищі на сервері. Ключ не передається в інші частини системи, не кешується, і не доступний для користувача у зворотному напрямку, що повністю відповідає принципам безпечної односторонньої передачі повідомлень.

Таким чином, обидва ці вхідні елементи, текст повідомлення і публічний ключ, взаємодіють на рівні браузера користувача, що дозволяє максимально зменшити ризик перехоплення або маніпуляцій з даними до моменту надсилання. Увесь процес організовано так, щоб навіть сервер не мав доступу до відкритого тексту повідомлення до моменту його дешифрування строго визначеним алгоритмом, із залученням приватного ключа.

3.1.1 Структура інформаційної системи

Архітектура створеної інформаційної системи складається з декількох взаємопов’язаних, але чітко відокремлених компонентів, кожен з яких виконує свою унікальну роль у забезпеченні безпечного та односпрямованого зв’язку між користувачем і адміністраторами. Особливістю реалізації є те, що компоненти системи розміщені в хмарному середовищі (Render), що дозволяє забезпечити постійну доступність та масштабованість без потреби підтримувати фізичну інфраструктуру.

Вхідною точкою взаємодії у всій системі виступає Telegram-бот, логіка якого реалізована у модулі main.py. Його головне завдання – реагувати на команди /start та /help, надсилаючи користувачеві посилання на веб-форму для заповнення повідомлення. Важливо підкреслити, що жодне текстове повідомлення, надіслане в бот вручну, не обробляється – бот завжди відповідає стандартним повідомленням з тим самим посиланням на веб-форму. Це реалізовано через обробку усіх вхідних повідомлень як “невідомих команд”, аби уникнути будь-яких залишків інформації в історії чату. Таким чином, Telegram-бот виступає не як месенджер, а як диспетчер безпечного доступу до зашифрованого каналу.

Другим і більш складним компонентом є серверна частина, реалізована за допомогою фреймворку Flask у файлі app.py. Саме вона обробляє запити від користувачів веб-форми. Серверна логіка включає дві основні функції: перша – видача HTML-форми разом із вбудованим публічним ключем шифрування, а друга – обробка POST-запиту з зашифрованим повідомленням. Після приймання повідомлення сервер виконує розшифрування за допомогою приватного RSA-ключа, зчитаного з файлу у форматі PEM, і одразу надсилає розшифрований текст адміністраторам через Telegram Bot API. На завершення, сервер перенаправляє користувача на сторінку подяки без жодного збереження чи логування самого тексту звернення, що гарантує однократність передачі.

Щодо Redis, то хоча система в поточній реалізації принципово не використовує кешування для повідомлень, технологія Redis все ж згадується у контексті платформи Render, яка надає її як частину інфраструктури. Його можна було б використати для логування активності, контролю доступу або таймінгів, але задля дотримання умов про повну відсутність кешу, цей механізм наразі свідомо не задіяний. Його наявність лише демонструє можливість масштабування системи в майбутньому, якщо буде потреба, наприклад, в обмеженні частоти запитів або введенні одноразових токенів доступу. Візуалізація структури системи представлена на рис. 3.1.

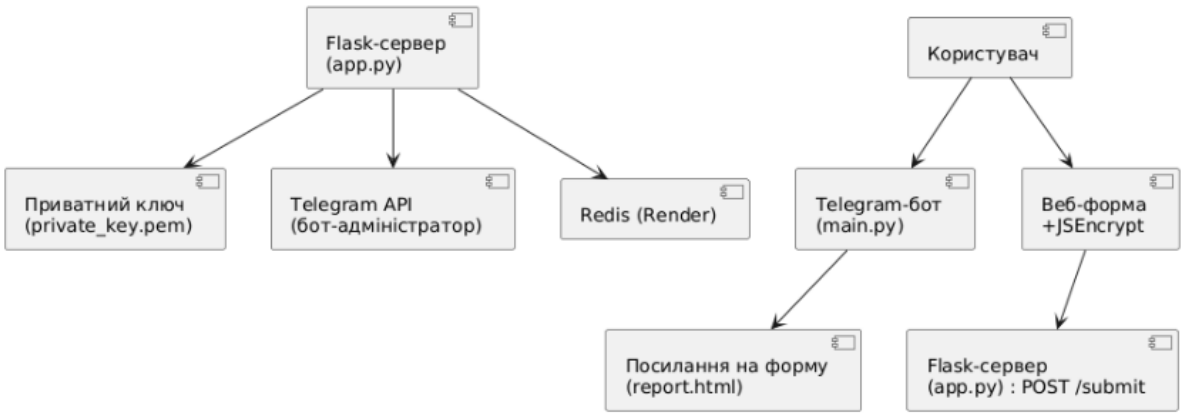

Рисунок 3.1 – Логічна структура системи

Таким чином, структура системи забезпечує чітке розмежування відповідальності між модулями, а також можливість майбутнього масштабування, якщо згодом буде вирішено додати додаткові функції або рівні контролю.

3.1.2 Компоненти безпеки

У центрі розробленої інформаційної системи лежить не лише функціональна простота, а й продуманий захист даних, що передаються. Оскільки система орієнтована на передачу чутливої інформації (зокрема повідомлень, що можуть містити особисті дані, прохання або звернення), безпека стає її базовим фундаментом. Реалізація захисту побудована на двох основних компонентах: використанні сучасного асиметричного шифрування та повному виключенні кешування на всіх етапах взаємодії. Ключову роль у забезпеченні конфіденційності відіграє алгоритм RSA, реалізований на стороні клієнта за допомогою бібліотеки JSEncrypt, яка працює безпосередньо у браузері користувача. Коли користувач відкриває веб-форму, сервер автоматично вбудовує в HTML-документ публічний ключ у форматі PEM. Він зчитується з локального файлу public_key.pem, що зберігається на сервері, і передається у шаблон сторінки. Після цього весь процес шифрування відбувається на клієнтському боці, без передачі відкритого тексту або навіть участі сервера до моменту надсилання зашифрованого повідомлення.

Шифрування працює за схемою, де публічний ключ використовується для зашифрування, а розшифрувати повідомлення можливо виключно за допомогою приватного ключа, який зберігається на сервері у файлі private_key.pem. Цей ключ ніколи не передається у браузер, не доступний ззовні і використовується виключно на боці серверної логіки під час обробки запиту. Така модель гарантує, що навіть у випадку перехоплення HTTP-запиту (наприклад, у відкритій мережі), зміст повідомлення не буде доступним стороннім особам без приватного ключа, який фізично відокремлений від клієнта.

Другим важливим аспектом безпеки є заборона кешування. Повідомлення, яке шифрується у браузері, не повинно залишити сліду ні в кеші браузера, ні в

логах, ні в локальному сховищі. Для цього на рівні HTML та серверної відповіді були реалізовані спеціальні HTTP-заголовки. Конкретно, заголовки Cache-Control: no-store, no-cache, must-revalidate, Pragma: no-cache та Expires: 0 передаються з кожною відповіддю від Flask-сервера. Вони змушують браузер щоразу завантажувати нову копію форми і не зберігати попередні введення. Це важливо не лише з точки зору безпеки, а й з погляду односторонньої моделі зв’язку – система не повинна мати "пам’яті" про попередні дії користувача.

На додачу до цього, архітектура побудована так, щоб взагалі не мати жодного зворотного каналу або логіки відповіді на повідомлення користувача. Користувач отримує лише статичну сторінку для вводу та коротке підтвердження після відправки. Будь-яке збереження інформації, повторне відображення, або тим паче створення історії – виключено з принципів функціонування.

3.2 Моделювання системи

У центрі архітектури системи лежить простий, але логічно завершений сценарій взаємодії між кінцевим користувачем та адміністративною частиною, реалізований через Telegram і вебінтерфейс. Кожен етап побудований з урахуванням того, щоб користувачеві не потрібно було розуміти складності шифрування або веббезпеки – натомість усі дії автоматизовані і приховані за інтуїтивним інтерфейсом. Водночас усі технічні процеси узгоджуються між собою, створюючи узгоджену логіку, яка мінімізує людські помилки й унеможливлює витік даних.

Взаємодія починається зі стандартної Telegram-команди /start. Користувач вводить її у чаті з ботом, і той миттєво відповідає, надсилаючи коротке інформативне повідомлення, у якому розписано, як залишити повідомлення адміністрації. Головним елементом відповіді є гіперпосилання на вебсторінку форми, яка буде використана для написання звернення. Важливо, що бот не пропонує прямого діалогу чи введення тексту в самому чаті – незалежно від того, що саме напише користувач після цього, бот реагуватиме однаково: знову надсилає

посилання на форму, чітко дотримуючись моделі односпрямованого каналу. Таким чином, Telegram використовується виключно як точка старту, а не середовище обміну повідомленнями.

Після переходу за посиланням користувач потрапляє на спеціально створену веб-форму, де йому пропонується ввести текст свого повідомлення. У цьому полі може бути будь-яка інформація: ім’я, ситуація, запит чи звернення. Сам інтерфейс форми є мінімалістичним і зосереджений на головному – поле для вводу, коротка інструкція та кнопка «Надіслати». Саме на цьому етапі, ще до натискання кнопки, браузер користувача завантажує публічний ключ у фоновому режимі – цей ключ вбудовується в сторінку ще на етапі її завантаження, тож користувач не бачить цього процесу.

Коли користувач натискає кнопку надсилання, у браузері автоматично виконується шифрування введеного повідомлення за допомогою бібліотеки JSEncrypt. Повідомлення перетворюється у зашифрований текст у форматі base64, який потім записується у приховане поле форми. Лише після цього форма стандартним способом (через method="POST") надсилається на сервер, не розкриваючи відкритий текст ні в HTTP-трафіку, ні в логах, ні в браузері.

Усе це від старту через Telegram до натискання кнопки відбувається без залучення сторонніх сервісів або посередників. Інтерфейс створено таким чином, щоб навіть найменш досвідчений користувач міг залишити повідомлення без помилок або труднощів, тоді як технічна частина забезпечує, щоб усі дії були виконані правильно та безпечно. Сценарій передбачає одне-єдине повідомлення в одному напрямку, без історії, без збереження сесій, і без можливості повторного використання введеної інформації, що повністю узгоджується з поставленими вимогами про односторонній та одноразовий характер комунікації. Візуалізація даного сценарію представлена на рис. 3.2.

Рисунок 3.2 – Візуалізація взаємодії користувача з системою

Архітектура інформаційної системи ґрунтується на чіткому поділі на клієнтську та серверну частини, кожна з яких реалізує свій набір функцій. Такий підхід забезпечує високу безпеку, масштабованість і простоту обслуговування системи, оскільки будь-яку її частину можна оновлювати чи замінювати незалежно від інших.

Клієнтська частина зводиться до вебсторінки з формою, реалізованою в HTML та JavaScript. Центральною ланкою клієнтської логіки виступає бібліотека JSEncrypt, яка дозволяє шифрувати повідомлення на стороні браузера перед його надсиланням на сервер. Це означає, що текст повідомлення користувача ніколи не передається в незашифрованому вигляді, ні через інтернет, ні через сам веб-сервер. Уся обробка, пов’язана із шифруванням, відбувається повністю локально в браузері. Це забезпечує критично важливу властивість системи – незмінність і таємність даних під час передачі.

Публічний ключ, необхідний для цього шифрування, динамічно вбудовується в HTML-сторінку при її генерації сервером. Код JSEncrypt інтегрується у вигляді окремого JavaScript-файлу (jsencrypt.min.js), який підключається до сторінки, а логіка скрипта пов’язана з кнопкою «Надіслати» у формі. Перед сабмітом, користувацький текст зчитується з поля textarea, шифрується, результат записується у приховане поле, і лише тоді дані

надсилаються. У такий спосіб відбувається передача виключно зашифрованої інформації без участі самого тексту в HTTP-запиті.

Серверна частина реалізована за допомогою Python-фреймворку Flask, що забезпечує обробку HTTP-запитів, розшифрування повідомлень та їх пересилання адміністраторам. Основні логічні блоки зосереджено у файлі app.py. Тут обробляються два основні типи запитів: GET-запит для показу форми (/report) та POST-запит при її сабміті (/submit). При надходженні зашифрованого повідомлення, сервер виконує його декодування з base64, після чого розшифровує його за допомогою приватного ключа, зчитаного з захищеного файлу private_key.pem. Розшифрування виконується з використанням криптографічної бібліотеки cryptography, яка гарантує відповідність сучасним стандартам безпеки (PKCS#1 v1.5).

Після успішної розшифровки, сервер відразу надсилає повідомлення через Telegram Bot API до обраних ID адміністраторів, використовуючи метод send_message. Увесь процес побудований на асинхронній взаємодії з Telegram API через бібліотеку python-telegram-bot, що дозволяє зменшити затримки у відповіді сервера. Якщо повідомлення було успішно доставлене, користувача автоматично перенаправляють на сторінку з підтвердженням надсилання.

Таким чином, клієнтська частина бере на себе функцію збору і шифрування даних, а серверна – їх дешифрування та безпечну доставку. Весь шлях повідомлення, від введення до отримання адміністратором, побудований так, щоб уникнути будь-якого посередника або небезпечного проміжного етапу. У результаті формується цілісна архітектура з чітким розподілом ролей, у якій безпека і простота використання взаємно підсилюють одна одну. Візуалізація архітектури системи представлена на рис. 3.3.

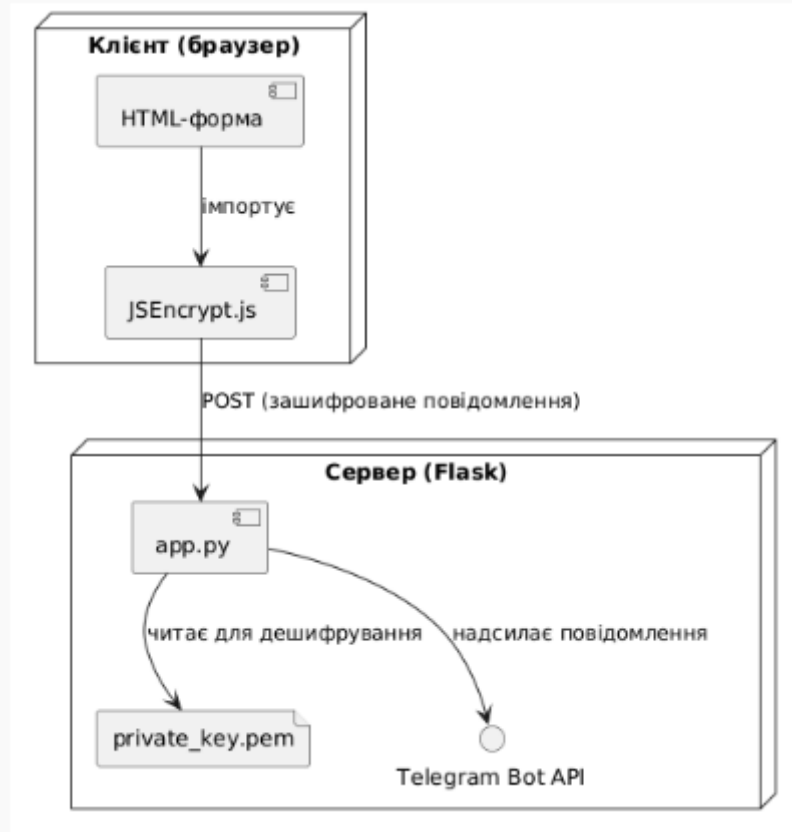

Рисунок 3.3 – Архітектура системи

Усі функціональні можливості розробленої системи реалізовано у трьох ключових модулях, кожен з яких виконує окрему роль у ланцюжку взаємодії. Завдяки цьому структура проєкту залишається логічно прозорою, а підтримка коду простою і передбачуваною.

Першим і з погляду користувача, головним є модуль report.html. Це клієнтська веб-сторінка, яка надає користувачу інтерфейс для написання та надсилання повідомлення. Візуально вона виглядає як звичайна форма з полем для тексту та кнопкою «Надіслати», проте її внутрішня логіка набагато складніша. Перед відправкою повідомлення, воно обов’язково проходить етап асиметричного шифрування у браузері користувача за допомогою бібліотеки JSEncrypt, яка підключена як зовнішній JavaScript-файл. Публічний ключ у форматі PEM вбудовується динамічно, сервер під час формування сторінки в Flask-шаблон вставляє вміст файла public_key.pem, який потім використовується JSEncrypt для шифрування. Як тільки користувач натискає кнопку, скрипт зчитує вміст поля,

шифрує його, і результат поміщає у приховане поле <input type="hidden">. Саме це поле надсилається на сервер, а не відкритий текст. Таким чином, report.html поєднує в собі одночасно і зовнішню частину взаємодії, і критично важливу частину безпеки – шифрування повідомлення.

Другим модулем є app.py, який представляє собою основний серверний компонент системи. Написаний з використанням фреймворку Flask, він реалізує дві головні функції: генерацію форми (GET-запит до /report) та обробку зашифрованих повідомлень (POST-запит до /submit). У першому випадку сервер зчитує публічний ключ і вбудовує його в шаблон сторінки. У другому – приймає повідомлення у base64-форматі, розшифровує його за допомогою приватного ключа, і після успішної дешифрації надсилає повідомлення адміністраторам через Telegram Bot API. Доступ до приватного ключа здійснюється шляхом читання локального PEM-файла private_key.pem, який не передається в мережу і не виходить за межі серверного середовища. Окрім цього, у app.py реалізовано ще одну важливу деталь – заборона кешування. Через декоратор @app.after_request усім HTTP-відповідям сервер додає заголовки Cache-Control, Pragma та Expires, які змушують браузер не зберігати сторінку, її вміст або історію введення, що особливо важливо для гарантування одноразовості та конфіденційності повідомлення.

Третім завершальним модулем є main.py, який відповідає за запуск і логіку Telegram-бота. Реалізований на базі бібліотеки python-telegram-bot, бот обробляє лише дві команди: /start і /help, надсилаючи у відповідь лінк на форму. Усі інші повідомлення чи команди інтерпретуються як «невідомі» та обробляються окремим хендлером, який також надсилає лише посилання на веб-форму. Таким чином, незалежно від дій користувача в чаті, бот не вступає в переписку і не зберігає історії. Фактично, main.py виконує лише роль диспетчера, який перенаправляє користувача з Telegram до безпечного середовища введення повідомлення.

Ця модульна організація дозволяє кожному з елементів системи залишатися простим, ізольованим і легко тестованим. При цьому всі вони працюють як частини

єдиного механізму, що забезпечує односторонню, зашифровану, одноразову передачу повідомлень від користувача до адміністратора, без історії, без кешування і без прямого контакту.

Окрему увагу під час розробки було приділено аспектам захисту інформації в момент її передачі між клієнтом і сервером. Оскільки система оперує потенційно конфіденційними повідомленнями, що можуть містити персональні дані або чутливу інформацію, критично важливо було не лише зашифрувати вміст повідомлення, а й забезпечити безпечний канал його транспортування. У цьому контексті на продакшн-середовищі використовуються додаткові рівні захисту, які доповнюють криптографічну модель шифрування.

Найважливішим елементом є використання HTTPS, захищеного протоколу передачі даних, що гарантує шифрування всього HTTP-трафіку між браузером користувача і сервером. У випадку деплойменту через Render ця функціональність надається автоматично: усі запити, які надходять до серверного застосунку, автоматично перенаправляються з HTTP на HTTPS. Завдяки цьому вся взаємодія між браузером та сервером, включно з завантаженням форми, публічного ключа, а також надсиланням зашифрованого повідомлення, відбувається у зашифрованому каналі TLS. Це виключає можливість «прослуховування» даних третіми особами навіть на рівні мережевого трафіку.

Окрім HTTPS, реалізовано ще один критично важливий рівень захисту – повна відсутність кешування на стороні клієнта. Для цього серверна частина (модуль app.py) налаштована так, щоб у кожну HTTP-відповідь автоматично додавались заголовки, які забороняють браузеру зберігати будь-який вміст у кеші. Зокрема, використано такі заголовки:

– Cache-Control: no-store, no-cache, must-revalidate, max-age=0 – вказує браузеру не зберігати жодної копії сторінки або відповідей;

– Pragma: no-cache – забезпечує сумісність зі старішими версіями браузерів;

– Expires: 0 – примушує вважати сторінку негайно «протермінованою» з моменту завантаження.

Ці заголовки встановлюються через спеціальний декоратор Flask @after_request, який автоматично додає їх до кожної відповіді сервера. У результаті, навіть якщо користувач повернеться на сторінку назад через кнопку браузера, форма завантажиться заново, з новим сеансом і оновленим публічним ключем. Жодне введене повідомлення не зберігається в автозаповненні чи історії введення браузера, де після завершення вона залишає нульовий цифровий слід.

Таким чином, комбінація HTTPS-тунелю та агресивної політики no-cache створює надійний бар'єр для потенційного перехоплення чи витоку інформації. Навіть якщо користувач працює у публічній Wi-Fi мережі, або використовує загальний пристрій, шанси на доступ до повідомлення будь-ким стороннім практично нульові. Система не лише не читає і не зберігає дані, вона навіть не дозволяє браузеру це зробити. Саме така агресивна безпека і відповідає ідеології цього проєкту – передача одного повідомлення, один раз, у єдиному напрямку.

3.2 Аналіз отриманих результатів

Оцінка функціональності створеної інформаційної системи починається з порівняння її фактичної поведінки з вимогами, що були сформульовані на етапі технічного завдання. Основною метою проєкту було створення Telegram-бота, який забезпечує одноразову, односторонню та захищену передачу повідомлень, без збереження історії, без кешування та без двосторонньої взаємодії.

Перший критерій – одноразовість повідомлення, був досягнутий через архітектуру, у якій повідомлення не зберігається в базі даних або файлах, а обробляється виключно у пам’яті на момент надсилання. Після розшифрування і пересилання в Telegram повідомлення не залишає після себе жодного цифрового сліду. Таким чином, кожне звернення існує лише один раз, у момент, коли воно читається й відправляється адміністраторам.

Другий фундаментальний принцип – односторонність комунікації. Реалізація цього принципу проявляється у відсутності будь-якої відповіді користувачеві на зміст повідомлення. Користувач не може продовжити діалог у тому ж каналі, оскільки Telegram-бот не приймає і не обробляє текстові повідомлення як контент. Будь-яка спроба написати щось боту, чи то текст, чи команда, приводить лише до повторного надсилання посилання на веб-форму. Тим самим бот не виконує функцію чат-інтерфейсу, а діє виключно як посередник для перенаправлення у безпечне середовище. Адміністратор, у свою чергу, отримує повідомлення без можливості відповіді через той самий канал, що унеможливлює повторне використання сеансу.

Ще одним важливим аспектом відповідності технічному завданню є відсутність кешу та історії повідомлень, як у Telegram, так і у браузері користувача. На Telegram-боці повідомлення не зберігаються в чаті користувача, оскільки взаємодія обмежується виключно надсиланням інформаційного посилання. Сам текст повідомлення не вводиться в сам Telegram, а лише у веб-форму, яка відкривається у зовнішньому браузері. Це означає, що ні Telegram, ні сервер не мають доступу до відкритого тексту, а отже не можуть його архівувати чи зберігати.

Кешування у браузері було повністю вимкнене на рівні HTTP-заголовків, що робить неможливим автоматичне збереження вмісту форми. Навіть якщо користувач випадково оновить сторінку чи повернеться назад, усе введене зникне. Завдяки цьому забезпечується гарантована "одноразовість" сеансу де після натискання кнопки "Надіслати" не залишається нічого, що можна було б відновити.

Для оцінки надійності та практичної працездатності створеної інформаційної системи було проведено низку тестувань, спрямованих на перевірку її роботи в реальних умовах. Метою цього етапу було переконатися, що система стабільно функціонує на різних пристроях, правильно виконує шифрування та дешифрування повідомлень, а також забезпечує гарантовану доставку повідомлень адміністраторам Telegram.

Першим етапом було тестування на різних типах пристроїв і браузерів. Повідомлення надсилались із ноутбуків, планшетів та мобільних телефонів, використовуючи сучасні браузери (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge). Усі пристрої однаково коректно завантажували веб-форму, отримували публічний ключ і виконували шифрування повідомлення у браузері. Незалежно від платформи, результатом була успішна генерація зашифрованого повідомлення у форматі base64, що свідчить про універсальність рішення. Особливо важливо було перевірити поведінку на мобільних пристроях, де обмеження пам’яті та кешування можуть впливати на роботу скриптів, але навіть у цих умовах форма працювала коректно і не зберігала жодного введеного тексту при перезавантаженні.

Другим кроком стала перевірка криптографічної цілісності системи. Для цього проводився контроль як процесу шифрування, так і розшифрування повідомлень. Було протестовано десятки повідомлень з різною довжиною та складністю (включно з українськими, латинськими літерами, спецсимволами, емодзі). Усі повідомлення, які були зашифровані у браузері, успішно дешифрувались на сервері за допомогою приватного ключа без втрати вмісту або помилок кодування. В разі навмисного спотворення повідомлення (наприклад, при передачі некоректного base64-рядка), сервер реагував коректно – видавав помилку з відповідним HTTP-кодом 400 та не продовжував обробку, що свідчить про наявність надійної валідації вхідних даних.

Третім і завершальним аспектом тестування була перевірка фактичної доставки повідомлень адміністраторам у Telegram. У кожному тестовому випадку, після надсилання форми, у логах сервера (app.logger.info) фіксувалась інформація про успішну розшифровку повідомлення, а також про спробу відправки через Telegram Bot API. Telegram-бот надсилав повідомлення на конкретний chat_id адміністратора, і кожне повідомлення з'являлось у Telegram негайно, без затримок або помилок. У разі виникнення будь-яких проблем (наприклад, відсутність інтернету, помилка API, некоректний токен), система фіксувала відповідні

помилки у логах із точним описом винятку, що дозволяє швидко ідентифікувати джерело проблеми.

Попри високу ефективність і відповідність заявленим вимогам, розроблена система не позбавлена певних обмежень, які логічно випливають із обраної моделі безпеки, технічної архітектури, а також залежності від сторонніх сервісів. Ці обмеження не є критичними, проте їх необхідно враховувати при використанні системи в довготривалих чи масштабних сценаріях.

Насамперед, слід згадати відсутність зворотного зв’язку з користувачем, яка закладена у саму концепцію односторонньої передачі. Система навмисно не передбачає можливості відповіді на повідомлення або побудови діалогу. З одного боку, це дозволяє забезпечити максимальну конфіденційність і безпеку, користувач передає звернення і більше ніде не фігурує. Але з іншого боку, у випадках, коли адміністраторам потрібно було б уточнити деталі чи надати допомогу у відповідь, система не дозволяє зробити цього напряму. Це обмеження є концептуальним і може бути подолане лише за рахунок зміни моделі, наприклад, додаванням опцій добровільного зворотного каналу (що суперечить поточному дизайну).

Іншим важливим обмеженням є залежність усієї криптографічної логіки від наявності й цілісності ключів. Без публічного ключа система не зможе шифрувати повідомлення у браузері, а без приватного – розшифровувати їх на сервері. Обидва ключі мають зберігатися в недоступному середовищі та не бути скомпрометованими. Якщо хоча б один із них буде втрачено або змінено неузгоджено, система втратить працездатність: повідомлення не будуть прийматися або дешифруватися. Це означає, що підтримка системи передбачає чітку процедуру зберігання ключів, регулярне резервне копіювання, а також контроль за їхнім форматом і доступом.

Крім цього, система має зовнішні технічні залежності, передусім, від хостинг-платформи Render, а також від Telegram Bot API. У випадку тимчасової недоступності Render (наприклад, через оновлення, збої або зміни в тарифах) серверна частина системи не зможе обробляти повідомлення або видавати форму.

Аналогічно, якщо Telegram API змінить протокол, тимчасово відключиться або введе обмеження (наприклад, на кількість повідомлень на день), система втратить здатність доставляти звернення адміністраторам. На щастя, і Render, і Telegram сервіси з високою стабільністю, однак залежність від них означає, що система не є повністю автономною.

Висновки до розділу 3

Модель системи, що базується на асиметричному шифруванні та суворій політиці «без історії», демонструє потенціал до створення надійного інструменту захищеного одноразового зв’язку. Розділення клієнтської та серверної частин, реалізація шифрування безпосередньо у браузері користувача за допомогою JSEncrypt, а також відсутність збереження повідомлень у Telegram дозволяють забезпечити високий рівень конфіденційності та анонімності.

Попередній аналіз показує, що система має низку суттєвих переваг: зручність використання, мінімалістичний інтерфейс, криптографічний захист на основі публічного/приватного ключа, а також захист від кешування за рахунок HTTP-заголовків. Усе це сприяє створенню середовища, де передане повідомлення існує виключно в момент доставки і більше ніде не зберігається.

Водночас виявлено й ряд обмежень, які потрібно враховувати у подальших етапах розробки та тестування. Відсутність зворотного зв’язку може бути критичною у певних сценаріях використання, а залежність від збереження ключів та зовнішніх API (Render, Telegram) вимагає додаткових заходів щодо відмовостійкості.

Таким чином, проведений аналіз підтверджує доцільність розробки такої системи в межах поставленого завдання. Подальші етапи проєкту мають зосереджуватись на практичній реалізації, валідації безпеки, перевірці поведінки у граничних випадках та потенційному розширенні функціональності.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Опис програмної реалізації

Розроблений програмний комплекс складається з двох основних компонентів, що тісно взаємодіють між собою, Telegram-бота та веб-додатку на Flask. Усе програмне забезпечення побудовано згідно з архітектурою «клієнт-сервер», де Telegram-бот виступає в ролі інтерфейсу для користувача, а Flask-сервер, як бекенд, що відповідає за логіку прийому, розшифрування та надсилання повідомлень.

Файлова структура проєкту є лаконічною, що забезпечує зручність у підтримці та масштабуванні. Основні файли проєкту:

– main.py – файл, що містить реалізацію Telegram-бота;
– app.py – основний веб-сервер на Flask, який обробляє запити від форми;
– report.html – HTML-шаблон для форми повідомлення;
– jsencrypt.min.js – підключена JS-бібліотека для шифрування повідомлення у браузері;
– public_key.pem, private_key.pem – пара RSA-ключів для шифрування/дешифрування;
– requirements.txt – список залежностей Python-проєкту.

Telegram-бот реалізований з використанням бібліотеки python-telegram-bot версії 20 або вище. Основне його завдання – бути проміжним звеном між користувачем та адміністраторами. Він не зберігає повідомлень у чаті, а лише надає лінк на веб-форму. Цим досягається повна конфіденційність, оскільки повідомлення шифруються вже у браузері користувача.

Налаштування токена бота та ID адміністратора задаються в main.py і показані на рис. 4.1.

```
TELEGRAM_BOT_TOKEN = "7888461204:AAEf1X2YtlV4-DMc6A5LQuQAqMU7bTJ4Tdg"
ADMIN_USER_IDS = [797316319]
REPORT_PAGE_URL = "https://web-app-d8fd.onrender.com/report"
```

Рисунок 4.1 – Налаштування токену, ID адміністраторів та посилання на web-форму

У цьому фрагменті вказується токен Telegram-бота, список ID користувачів, яким буде надсилатися розшифроване повідомлення (тобто адміністратори), а також URL сторінки форми.

Бот обробляє лише три типи взаємодії:

– команду /start;

– команду /help;

– усі інші повідомлення.

При цьому всі вони ведуть до одного результату – відправлення посилання на веб-форму. Приклад коду для виводу повідомлення показано на рис. 4.2.

```
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:  1 usage  Unluckich *
    user = update.effective_user
    welcome_message = (
        "Вітаю! Якщо ви хочете написати нам повідомлення, перейдіть за цим посиланням:\n\n"
        f"{REPORT_PAGE_URL}\n\n"
        "Тут ви зможете зашифрувати і відправити текст, і він надійде напряму адміністраторам."
    )
    await update.message.reply_text(welcome_message)
    logger.info(f"User {user.id} opened /start, sent link to web-form.")
```

Рисунок 4.2 – Приклад коду для виводу повідомлення після команди /start

Усі інші команди (наприклад, якщо користувач надрукує щось випадкове або помилково) обробляються функцією unknown_command, яка теж відсилає посилання на форму:

```python
async def unknown_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:  1 usage  Unluckich *

    info = (
        "Ви можете залишити нам повідомлення лише через веб-форму, щоб історія не залишалася в чаті:\n\n"
        f"{REPORT_PAGE_URL}"
    )
    await update.message.reply_text(info)
```

Рисунок 4.3 – Приклад коду для виводу при помилковій команді

Приклад реагування бота на команду /start продемонстровано на рис.4.4.

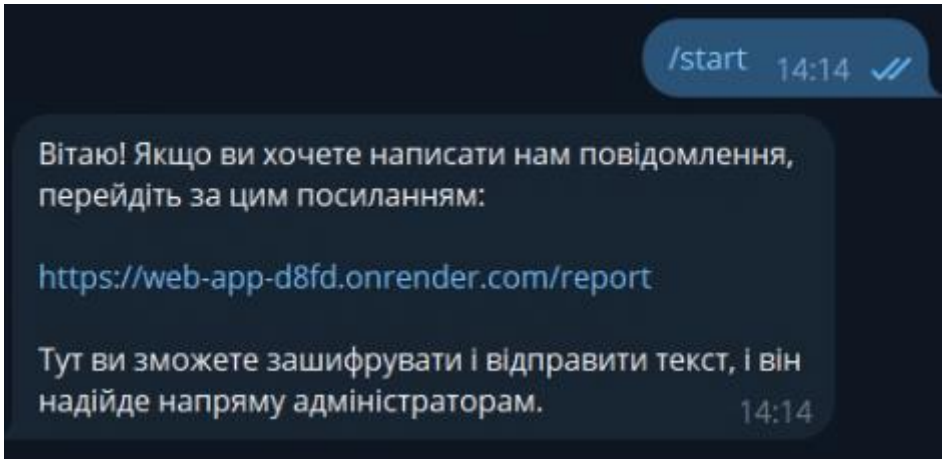

Рисунок 4.4 – Реагування бота на повідомлення

Уся основна логіка для запуску бота зосереджена у функції main(), яка додає відповідні хендлери і зображена на рис. 4.5.

```python
application.add_handler(CommandHandler("start", start))
application.add_handler(CommandHandler("help", help_command))
application.add_handler(
    MessageHandler(filters.TEXT | filters.COMMAND, unknown_command)
)
```

Рисунок 4.5 – Логіка запуску бота

Це означає, що бот фактично не зберігає жодної історії, не обробляє повідомлення напряму, а діє виключно як направляючий, користувач має залишити повідомлення через веб-форму.

Веб-частина системи реалізована за допомогою мікрофреймворку Flask, що дозволяє швидко та просто створювати веб-додатки на Python. Цей компонент відповідає за генерацію вебсторінки з формою для користувача, прийом зашифрованих повідомлень та передачу їх адміністраторам через Telegram.

Основні маршрути Flask-додатку описані в app.py. Їх усього три:

– /report – повертає HTML-сторінку з формою;
– /submit – обробляє POST-запит із зашифрованим повідомленням;
– /thankyou – показується після успішного надсилання.

При переході за маршрутом /report, сервер відкриває шаблон report.html, в який підставляється публічний ключ шифрування у PEM-форматі, приклад наведено на рис. 4.6.

```
@app.route("/report", methods=["GET"])  Unluckich
def report_form():
    try:
        with open(PUBLIC_KEY_PATH, "r") as f:
            public_pem = f.read()
    except FileNotFoundError:
        abort(500, description="Публічний ключ не знайдено на сервері.")
    return render_template("report.html", public_key=public_pem)
```

Рисунок 4.6 – Відкриття веб-шаблону

Цей фрагмент показує, що ключ динамічно читається з файла і передається до HTML-шаблону. Таким чином браузер отримує актуальний ключ щоразу при завантаженні сторінки.

Після того як користувач заповнює форму та натискає кнопку "Надіслати", браузер шифрує повідомлення та надсилає його як поле encrypted_message у POST-запиті на /submit.

Код показаний на рис. 4.7, показує процес, де спочатку виконується декодування з base64, потім розшифровка за допомогою приватного RSA-ключа, а далі, надсилання адміністраторам через Telegram API.

```python
@app.route("/submit", methods=["POST"])  Unluckich
def submit_report():
    encrypted_b64 = request.form.get("encrypted_message")
    if not encrypted_b64:
        abort(400, description="Не надійшло поле encrypted_message.")

    try:
        encrypted_bytes = b64decode(encrypted_b64)
        private_key = load_private_key()
        decrypted = private_key.decrypt(encrypted_bytes, padding.PKCS1v15())
        plain_text = decrypted.decode("utf-8")
```

Рисунок 4.7 – Обробка повідомлень

Оскільки Flask є синхронним фреймворком, а Telegram Bot API працює асинхронно, надсилання повідомлення відбувається за допомогою asyncio.run(...), що дозволяє виконати асинхронну функцію bot.send_message(...) у синхронному маршруті Flask.

Після успішного надсилання користувача перекидує на сторінку подяки де написано що його повідомлення успішно надіслано і дійшло до адміністраторів. Веб-форма показана на рис. 4.8 та сторінка після успішного надсилання на рис. 4.9.

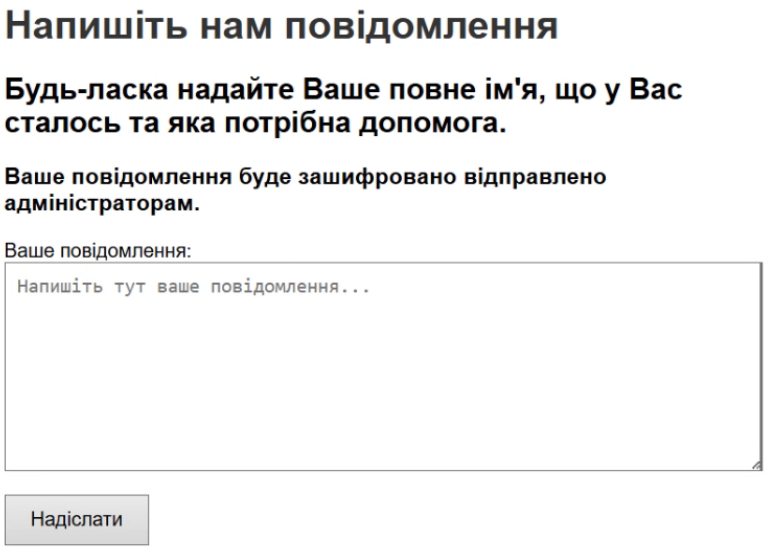

Рисунок 4.8 – Вигляд веб-форми

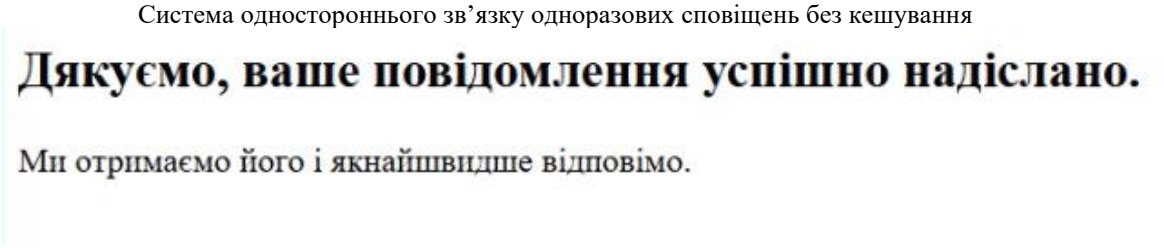

Рисунок 4.9 – Успішне надсилання повідомлення

Шифрування в реалізованому проєкті виконується на стороні клієнта, тобто в браузері користувача, що є головною перевагою з погляду конфіденційності. Для цього використовується бібліотека JSEncrypt, яка дозволяє легко шифрувати текст за алгоритмом RSA.

Публічний ключ publicKeyPEM передається динамічно з Flask-сервера і підставляється у шаблон за допомогою Jinja2. Користувач вводить звичайний текст, але в момент натискання кнопки «Надіслати», його повідомлення перетворюється на зашифровану строку.

У результаті вміст поля encrypted_message – це шифрований base64-текст, який жодним чином не може бути прочитаний на сервері без приватного ключа.

Приватний ключ зберігається на сервері у файлі private_key.pem і ніколи не передається клієнту. Тільки він дозволяє виконати розшифрування повідомлення на бекенді. Схема шифрування зображена на рис. 4.10.

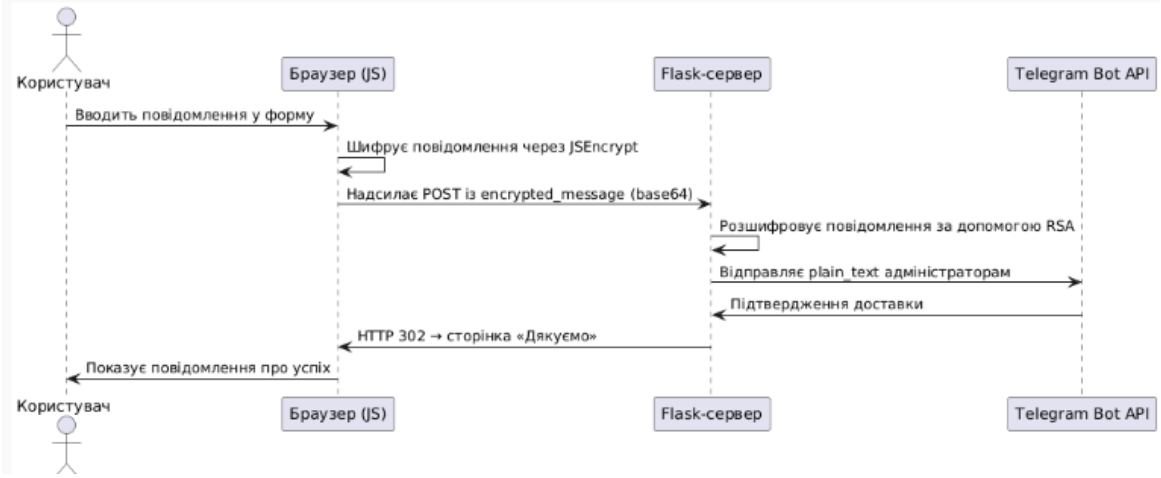

Рисунок 4.10 – Схема шифрування

Комунікація між користувачем та адміністраторами організована таким чином, щоб забезпечити повну анонімність та односторонню передачу повідомлення. Telegram-бот та вебінтерфейс розділені функціонально, але взаємодіють опосередковано через API Telegram. На рис. 4.11, продемонстровано приклад повідомлення, що отримують адміністратори.

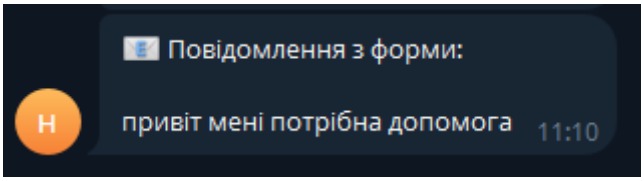

Рисунок 4.11 – Повідомлення з форми, що отримують адміністратори

Початковий контакт починається в Telegram. Коли користувач надсилає /start чи /help, бот повертає посилання на веб-форму, розміщену на Flask-сервері.

Далі відбувається перехід у браузер, де користувач заповнює форму. Уся логіка збору повідомлення та його шифрування реалізована на стороні клієнта. Завдяки цьому бот не бачить незашифрованого тексту.

POST-запит із зашифрованим повідомленням надсилається до Flask-додатку, який розшифровує його і відправляє текст адміністраторам напряму в Telegram через Telegram Bot API.

Таким чином Telegram-бот працює у двох режимах, як пасивний провідник для користувача (дає посилання на форму), та як активний, що доставляє повідомлення адміністраторам. Це дозволяє поєднати вебінтерфейс, зручний для введення тексту, з Telegram як платформою для миттєвого сповіщення.

Проєкт використовує відкриті бібліотеки, які роблять реалізацію простою та надійною:

– Flask – для створення веб-сервера та обробки HTTP-запитів;
– python-telegram-bot (версія 20+) – для реалізації Telegram-бота;
– cryptography – для роботи з RSA-ключами та дешифрування повідомлень;

– gunicorn – рекомендований для запуску Flask-додатку у production-режимі;

– JSEncrypt – легка бібліотека, яка реалізує RSA у браузері. Вона дозволяє шифрувати повідомлення перед тим, як вони залишають браузер користувача.

4.2 Запуск та використання бота

Щоб скористатися системою повідомлень, адміністратору необхідно переконатися, що Telegram-бот та веб-застосунок активні. Для цього необхідно попередньо налаштувати їх запуск, використовуючи файл main.py та app.py. Ці файли міститять основну логіку обробки команд від користувача в Telegram та спрямування на форму повідомлення.

Користувач у Telegram знаходить бота за його іменем, відкриває чат і надсилає команду /start. У відповідь бот надішле посилання на веб-форму для залишення повідомлення. Таким чином, запуск бота – це однократна дія, після якої він може постійно працювати у фоновому режимі та приймати нових користувачів.

Після отримання посилання користувач натискає на нього, і в браузері відкривається сторінка веб-форми. Вона має простий, зрозумілий інтерфейс із полем для введення тексту повідомлення та кнопкою «Надіслати». Вона створена спеціально для зручного, швидкого та захищеного залишення повідомлень. На сторінці користувач бачить грубо кажучи інструкцію, яка повідомляє, що: «Ваше повідомлення буде зашифровано у браузері перед надсиланням. Адміністратори отримають лише зашифрований варіант, який розшифровується лише на сервері.». Таким чином, користувачеві рекомендується не просто залишити загальний коментар, а вказати своє ім’я або псевдонім, суть ситуації або проблеми, очікувану допомогу. Форма не вимагає авторизації чи введення електронної пошти – це частина принципу анонімності.

Кнопка «Надіслати» активна лише після заповнення поля. Після натискання виконується шифрування та надсилання повідомлення.

Після успішної відправки користувач автоматично перенаправляється на сторінку подяки. Це свідчить про те, що повідомлення було передано адміністраторам.

Такий підхід не вимагає від користувача жодних технічних знань, усе працює інтуїтивно.

Однією з ключових особливостей системи є те, що введене повідомлення шифрується у браузері користувача ще до того, як потрапляє на сервер. Це забезпечує високий рівень безпеки і конфіденційності.

Для шифрування використовується JavaScript-бібліотека JSEncrypt, яка реалізує RSA (алгоритм з відкритим/закритим ключем). Публічний ключ передається зі сторони сервера під час завантаження сторінки. Після того як користувач натискає кнопку «Надіслати», виконується код, який зображений на рис. 4.12.

```
const plainText = document.getElementById("message").value;
console.log("Plain text:", plainText);

document.getElementById("encrypted_message").value = encrypted;
console.log("Записали в hidden поле.");
```

Рисунок 4.12 – Шифрування повідомлення перед надсиланням

Таким чином, введений текст не покидає браузера у відкритому вигляді, а сервер отримує лише шифрований текст (base64). Розшифрувати його можна тільки за допомогою приватного ключа, який є лише на сервері.

Після того як повідомлення було зашифроване та відправлене на сервер, відбувається його обробка, розшифрування, перевірка, доставка в Telegram. Якщо всі кроки проходять успішно, сервер перенаправляє користувача на спеціальну сторінку з інформуванням, що повідомлення надіслано. Це сигналізує

користувачеві, що процес завершено, і йому більше нічого робити не потрібно. Він може закрити сторінку.

Цей підхід – інтуїтивний, прозорий і не потребує жодних спеціальних знань. Водночас він гарантує, що користувач не залишає слідів повідомлення ні в Telegram, ні у браузері.

4.3 Тестування

У процесі створення програмного комплексу було проведено всебічне тестування ключових компонентів системи, зокрема функцій Telegram-бота, шифрування повідомлень у браузері, їх передачі на сервер та розшифрування.

Telegram-бот є першою точкою контакту для користувача, тому його стабільність і передбачуваність критично важливі.

Команда /start

Бот відповідає повідомленням з посиланням на веб-форму. Посилання формувалося коректно і було клікабельним.

Команда /help

Відповідь ідентична до /start, що є допустимим для простого інтерфейсу. Повідомлення інформує про конфіденційність і шифрування.

Будь-які інші повідомлення чи команди

Наприклад, якщо користувач надсилає Привіт, або /somethingwrong, бот не намагається обробити це як текст. Він знову повертає посилання на форму, пояснюючи, що листування через Telegram не підтримується.

Приклад не правильної команди та відповіді бота показано на рис. 4.13.

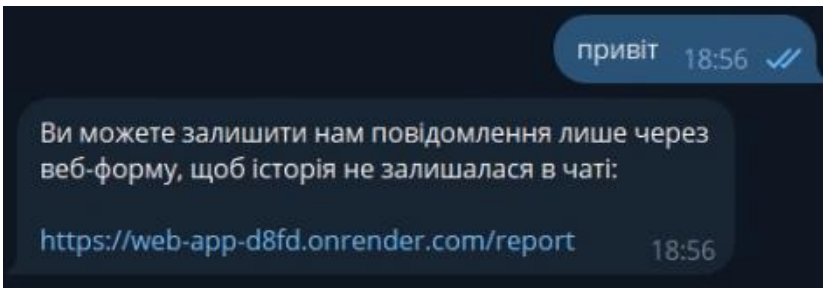

Рисунок 4.13 – Не правильна команда боту та його відповідь

Під час тестування було виявлено, що всі повідомлення від бота не містять конфіденційної інформації, не зберігаються в історії, і відразу спрямовують на захищену веб-форму.

Другим важливим елементом є веб-форма бо саме тут створюється, шифрується і надсилається повідомлення.

Для перевірки, що шифрування справді виконується було внесено зміну у код report.html, щоб показати результат. Після введення тестового повідомлення у консолі було видно два результати plainText що містить відкритий текст та encrypted, довгий рядок у форматі base64, що не піддається читанню. Це свідчить про те, що шифрування дійсно виконується до надсилання.

На сервері, повідомлення розшифровується за допомогою приватного RSA-ключа. Тому була перевірена функція зображена на рис. 4.14.

```
decrypted = private_key.decrypt(encrypted_bytes, padding.PKCS1v15())
plain_text = decrypted.decode("utf-8")
```

Рисунок 4.14 – Функція декодування

Усі тести показали, що повідомлення дійсно зашифроване валідним публічним ключем і воно розшифровується без помилок. Якщо змінити повідомлення вручну або пошкодити encrypted_message, сервер повертає 400 Bad Request.

Оскільки система обробляє чутливу інформацію, надзвичайно важливо передбачити можливі помилки як на стороні клієнта, так і на сервері. Під час тестування моделювалися різні сценарії відмов.

Якщо користувач натискає «Надіслати», не заповнивши форму, JavaScript у браузері не дасть надіслати порожній запит. У коді є перевірка зображена на рис. 4.15.

```
if (!plainText) {
  alert("Будь ласка, введіть повідомлення.");
  return;
}
```

Рисунок 4.15 – Перевірка на заповнення форми текстом

Це зупиняє обробку ще до шифрування. На практиці тестування показало, що неможливо натиснути «Надіслати» без введеного тексту.

Була змодельована ситуація, коли на сервері видалено файл public_key.pem. У такому випадку відкриття сторінки /report викликає помилку 500 (внутрішня помилка сервера). Код показано на рис. 4.16.

```
except FileNotFoundError:
    abort(500, description="Публічний ключ не знайдено на сервері.")
```

Рисунок 4.16 – Перевірка на відсутність ключа

Адміністратор бачить повідомлення про помилку і це дозволяє виявити проблему під час відлагодження.

У випадку, якщо поле encrypted_message відсутнє або має неправильний формат (наприклад, зіпсований base64), сервер повертає помилку 400. Це перевірено вручну через підставлення зіпсованого значення у форму. Повідомлення не обробляється, і в логах записується причина.

Для забезпечення зручності налагодження та моніторингу система веде повноцінне логування дій як у Telegram-боті, так і у Flask-сервері. Це логування реалізоване через модуль logging.

Логування у main.py показано на рис. 4.17.

```
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)
```

Рисунок 4.17 – Модуль logging в main.py

Такі логи фіксують чи було розшифровано повідомлення, кому воно було надіслане, чи виникли помилки. Це критично важливо для адміністраторів, які хочуть переконатися, що повідомлення доходять, не порушуючи конфіденційності тексту.

Завершальною частиною тестування є перевірка того, що повідомлення, яке прийшло з форми, дійсно доходить до адміністратора через Telegram. Для цього використовувався Telegram-бот із реальним ID адміністратора, вказаним у ADMIN_USER_IDS. Після розшифрування сервер відправляє повідомлення кожному адміну через асинхронний API.

Було перевірено чи отримує адміністратор повідомлення миттєво після надсилання, чи зберігається структура (розриви рядків, розділові знаки) та чи приходить воно у відповідному форматі.

Результати тестування підтвердили, що повідомлення надходять без затримок і втрат контенту. Жодного випадку помилки відправлення зафіксовано не було.

Висновки до розділу 4

У цьому розділі було детально розглянуто процес розробки, реалізації та тестування програмного комплексу, призначеного для анонімного та захищеного надсилання повідомлень через Telegram-бота з використанням веб-форми.

Під час опису програмної реалізації було показано, як система побудована з двох незалежних, але взаємодіючих компонентів, Telegram-бота та Flask-додатку. Бот виконує функцію з’єднання користувача з вебінтерфейсом, тоді як серверна частина відповідає за обробку, розшифрування та надсилання повідомлень

адміністраторам. Архітектура проєкту забезпечує високий рівень ізоляції та безпеки.

Було показано, що шифрування повідомлення відбувається на стороні клієнта, тобто у браузері користувача, ще до надсилання його на сервер. Це гарантує, що повідомлення залишається конфіденційним навіть у разі перехоплення мережевого трафіку. Сервер, у свою чергу, обробляє лише зашифровані дані та зберігає ключ розшифрування локально, без стороннього доступу.

Під час тестування було перевірено всі ключові функції системи: взаємодія з ботом, заповнення та обробка форми, правильність шифрування та дешифрування повідомлень, доставлення повідомлень у Telegram, обробка типових помилок. Результати тестування підтвердили працездатність і стабільність усіх компонентів.

Таким чином, можна зробити висновок, що реалізований програмний комплекс повністю виконує поставлену функцію – надає можливість залишити зашифроване повідомлення анонімно та безпечно, з доставкою його до адресатів через сучасні цифрові інструменти.

ВИСНОВКИ

У процесі розробки проєкту кваліфікаційної роботи я створив систему, яка дозволяє безпечно й анонімно надсилати повідомлення адміністраторам через Telegram, використовуючи веб-форму з шифруванням на стороні клієнта. Це рішення поєднує в собі телеграм-бота та веб-сервер, який реалізовано на Flask, з інтегрованою криптографією на основі RSA.

Основною метою було забезпечити приватність і безпеку користувача при передачі чутливих повідомлень. Для цього на стороні клієнта, в браузері, використовується бібліотека JSEncrypt, яка шифрує повідомлення публічним ключем, а на сервері повідомлення розшифровується приватним ключем та надсилається адміністраторам через Telegram API.

Окрім технічної реалізації, важливо було врахувати такі аспекти, як обхід кешування браузером, щоб дані не зберігались локально, обробка помилок при декодуванні та розшифруванні, а також організація взаємодії з користувачем у Telegram через зручні команди бота.

Таким чином було реалізувано просту, але водночас ефективну систему захищеного зворотнього зв’язку. Проєкт виявився практичним і досить гнучким для подальшого розширення, наприклад, можна додати базу даних, логування повідомлень чи підтримку кількох ролей користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bot examples. GitHub. URL: https://github.com/search?q=one-time%20message%20bot&type=repositories (last accessed: 05.05.2025).

2. Telegram Documentation. URL: https://telegram.org/faq (last accessed: 27.05.2025).

3. Signal. URL: https://signal.org/docs/ (last accessed: 10.05.2025).

4. Python Documentation. Python-telegram-bot. URL: https://docs.python-telegram-bot.org/en/stable/ (last accessed: 10.05.2025).

5. AWS Wickr. URL: https://wickr.com/product/me/ (last accessed: 01.04.2025).

6. AWS Wickr Security. URL: https://aws.amazon.com/ru/wickr/security/ (last accessed: 01.05.2025).

7. Мікросервісна архітектура. URL: https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/ (дата звернення: 30.05.2025).

8. Event-driven architecture. URL: https://dou.ua/forums/topic/38182/ (дата звернення: 06.05.2025).

9. Redis Documentation. URL: https://redis.io/docs/latest/ (last accessed: 30.04.2025).

10. Telegram API Documentation. URL: https://core.telegram.org/api (last accessed: 07.05.2025).

11. Documentation aiogram in Python. URL: https://docs.aiogram.dev/en/v3.20.0.post0/ (last accessed: 02.05.2025).

12. Flask User’s Guide. URL: https://flask.palletsprojects.com/en/stable/ (last accessed: 03.05.2025).

13. FastAPI. URL: https://fastapi.tiangolo.com (last accessed: 03.05.2025).

14. Python’s asincio. URL: https://docs.python.org/3/library/asyncio.html (last accessed: 03.05.2025).

15. Aiohttp Documentation. URL: https://docs.aiohttp.org/en/stable/ (last accessed: 04.05.2025).

16. JSEncrypt GitHub. URL: https://github.com/travist/jsencrypt (last accessed: 15.05.2025).

17. JSEncrypt. URL: https://www.npmjs.com/package/jsencrypt (last accessed: 15.05.2025).

18. Python Cryptography Documentation. URL: https://cryptography.io/en/latest/ (last accessed: 13.05.2025).

19. RSA Algorithm. URL: https://www.di-mgt.com.au/rsa_alg.html (last accessed: 20.05.2025).

20. Flask Deployment. URL: https://flask.palletsprojects.com/en/stable/deploying/ (last accessed: 21.05.2025).

21. Python base64 Data Encoding. URL: https://docs.python.org/3/library/base64.html (last accessed: 19.05.2025).

22. HTTP headers. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers (last accessed: 25.05.2025).

23. Gunicorn Documentation. URL: https://docs.gunicorn.org/en/stable/run.html (last accessed: 30.05.2025).

24. Public Key Encription. URL: https://www.geeksforgeeks.org/public-key-encryption/ (last accessed: 15.05.2025).

25. Python logging. URL: https://docs.python.org/3/library/logging.html (last accessed: 20.05.2025).

26. HTML Forms. URL: https://austingil.com/how-to-build-html-forms-right-security/ (last accessed: 28.05.2025).

27. Redis. URL: https://redis.io (last accessed: 30.05.2025).

28. Using async. URL: https://flask.palletsprojects.com/en/stable/async-await/ (last accessed: 05.05.2025).

29. Symmetric and Asymmetric Key Encryption. URL: https://www.geeksforgeeks.org/difference-between-symmetric-and-asymmetric-key-encryption/ (last accessed: 10.05.2025).

30. Web Crypto API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API (last accessed: 15.05.2025).

ДОДАТОК А

Код програмної реалізації

Файл main.py

```
import logging
from telegram import Update
from telegram.ext import (
    Application,
    CommandHandler,
    MessageHandler,
    ContextTypes,
    filters,
)
TELEGRAM_BOT_TOKEN = "7888461204:AAEf1X2YtlV4-DMc6A5LQuQAqMU7bTJ4Tdg"
ADMIN_USER_IDS = [797316319]
# URL Flask-сервера
REPORT_PAGE_URL = "https://web-app-d8fd.onrender.com/report"
# Налаштування логування
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logger = logging.getLogger(__name__)

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """/start — надсилаємо посилання на веб-форму."""
    user = update.effective_user
    welcome_message = (
        "Вітаю! Якщо ви хочете написати нам повідомлення, перейдіть за цим
посиланням:\n\n"
        f"{REPORT_PAGE_URL}\n\n"
        "Тут ви зможете зашифрувати і відправити текст, і він надійде напряму
адміністраторам."
    )
    await update.message.reply_text(welcome_message)
    logger.info(f"User {user.id} opened /start, sent link to web-form.")

async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """/help — теж просто даємо лінк на веб-форму."""
    help_text = (
        "Щоб надіслати нам повідомлення, скористайтеся нашою веб-формою:\n\n"
        f"{REPORT_PAGE_URL}\n\n"
        "Повідомлення буде зашифроване у вашому браузері, а потім доставлене
адміністраторам."
    )
    await update.message.reply_text(help_text)
async def unknown_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    info = (
        "Ви можете залишити нам повідомлення лише через веб-форму, щоб історія не
залишалася в чаті:\n\n"
        f"{REPORT_PAGE_URL}"
    )
    await update.message.reply_text(info)
def main() -> None:
    if TELEGRAM_BOT_TOKEN == "YOUR_TELEGRAM_BOT_TOKEN":
        logger.error("Будь ласка, встановіть TELEGRAM_BOT_TOKEN у main.py")
        return
```

```
    application = Application.builder().token(TELEGRAM_BOT_TOKEN).build()

    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))
    # Усе інше (text чи commands) — перекидаємо на форму
    application.add_handler(
        MessageHandler(filters.TEXT | filters.COMMAND, unknown_command)
    )
    logger.info("Bot starting...")
    application.run_polling()
if __name__ == "__main__":
    main()
```

Файл app.py

```
import os
import logging
import asyncio
import threading
import time
import redis
from flask import Flask, render_template, request, redirect, url_for, abort, make_response
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.fernet import Fernet
from telegram import Bot
from base64 import b64decode
from asyncio import run_coroutine_threadsafe

# Налаштування
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
KEYS_DIR = os.path.join(BASE_DIR, "keys")
PRIVATE_KEY_PATH = os.path.join(KEYS_DIR, "private_key.pem")
PUBLIC_KEY_PATH = os.path.join(KEYS_DIR, "public_key.pem")
FERNET_KEY_PATH = os.path.join(KEYS_DIR, "fernet.key")

TELEGRAM_BOT_TOKEN = os.getenv("TELEGRAM_BOT_TOKEN")
if not TELEGRAM_BOT_TOKEN:
    raise RuntimeError("TELEGRAM_BOT_TOKEN is not set in environment variables")

ADMIN_USER_IDS = [int(uid) for uid in os.getenv("ADMIN_USER_IDS", "").split(",") if uid]

# Redis settings
REDIS_URL = os.getenv("REDIS_URL")
redis_client = None

def init_redis():
    global redis_client
    try:
        if REDIS_URL:
            redis_client = redis.from_url(REDIS_URL, decode_responses=False)
            redis_client.ping()
            logging.info("✅ Connected to Redis via REDIS_URL.")
        else:
            logging.warning("⚠️ REDIS_URL not set, Redis will not be used.")
    except Exception as e:
        logging.error(f"❌ Failed to connect to Redis: {e}")
        redis_client = None
```

```
init_redis()
QUEUE_KEY = "telegram_queue"

app = Flask(__name__)
app.logger.setLevel(logging.INFO)

# Load or generate Fernet key
def load_fernet_key():
    if not os.path.exists(FERNET_KEY_PATH):
        key = Fernet.generate_key()
        with open(FERNET_KEY_PATH, "wb") as f:
            f.write(key)
    else:
        with open(FERNET_KEY_PATH, "rb") as f:
            key = f.read()
    return key

fernet = Fernet(load_fernet_key())

# Initialize Telegram Bot
bot = Bot(token=TELEGRAM_BOT_TOKEN)
bot_loop = asyncio.new_event_loop()


def load_private_key():
    with open(PRIVATE_KEY_PATH, "rb") as key_file:
        return serialization.load_pem_private_key(
            key_file.read(), password=None
        )

@app.after_request
def add_no_cache_headers(response):
    response.headers.update({
        "Cache-Control": "no-store, no-cache, must-revalidate, max-age=0",
        "Pragma": "no-cache",
        "Expires": "0"
    })
    return response

@app.route("/report", methods=["GET"])
def report_form():
    try:
        with open(PUBLIC_KEY_PATH, "r") as f:
            public_pem = f.read()
    except FileNotFoundError:
        abort(500, description="Публічний ключ не знайдено на сервері.")
    return render_template("report.html", public_key=public_pem)

@app.route("/submit", methods=["POST"])
def submit_report():
    encrypted_b64 = request.form.get("encrypted_message")
    if not encrypted_b64:
        abort(400, description="Не надійшло поле encrypted_message.")

    try:
        encrypted_bytes = b64decode(encrypted_b64)
        private_key = load_private_key()
        decrypted = private_key.decrypt(encrypted_bytes, padding.PKCS1v15())
        plain_text = decrypted.decode("utf-8")
    except Exception as e:
```

```
        app.logger.error(f"Помилка розшифровки: {e}")
        abort(400, description="Не вдалося розшифрувати повідомлення.")

    if not redis_client:
        app.logger.warning("Redis недоступний, повідомлення не збережено.")
        abort(503, description="Сервіс тимчасово недоступний (немає Redis).")

    try:
        encrypted_payload = fernet.encrypt(plain_text.encode('utf-8'))
        redis_client.lpush(QUEUE_KEY, encrypted_payload)
        app.logger.info("📥 Повідомлення додано в Redis-чергу.")
    except Exception as e:
        app.logger.error(f"❌ Помилка запису в Redis: {e}")
        abort(500, description="Помилка внутрішнього сервісу.")

    return redirect(url_for("thank_you"))

@app.route("/thankyou", methods=["GET"])
def thank_you():
    html = ("""
    <html><head><meta charset="utf-8"><title>Дякуємо!</title></head><body>
      <h2>Дякуємо, ваше повідомлення успішно надіслано.</h2>
      <p>Ми отримаємо його і якнайшвидше відповімо.</p>
    </body></html>
    """)
    return make_response(html)
# Воркер
def telegram_worker():
    global redis_client
    if not redis_client:
        app.logger.warning("Redis недоступний. Worker не запущений.")
        return

    app.logger.info("👷 Telegram worker запущено.")
    while True:
        try:
            item = redis_client.brpop(QUEUE_KEY, timeout=5)
            if not item:
                continue
            _, encrypted_payload = item
            text = fernet.decrypt(encrypted_payload).decode('utf-8')
            for admin_id in ADMIN_USER_IDS:
                future = run_coroutine_threadsafe(
                    bot.send_message(chat_id=admin_id, text=f"📩 Повідомлення з
форми:\n\n{text}"),
                    bot_loop
                )
                future.result(timeout=10)
                app.logger.info(f"✅ Надіслано до {admin_id}")
        except Exception as e:
            app.logger.error(f"❌ Worker помилка: {e}")
        time.sleep(0.1)

if __name__ == "__main__":
    threading.Thread(target=bot_loop.run_forever, daemon=True).start()
    if redis_client:
        threading.Thread(target=telegram_worker, daemon=True).start()
    else:
        app.logger.warning("⚠️ Redis не ініціалізований. Воркери не запущено.")
```

```
    app.run(host="0.0.0.0", port=int(os.getenv("PORT", 5000)), debug=True)
```

Файл report.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate" />
  <meta http-equiv="Pragma" content="no-cache" />
  <meta http-equiv="Expires" content="0" />
  <title>Залишити повідомлення</title>

  <!-- Підключаємо JSEncrypt із папки static/js -->
  <script src="{{ url_for('static', filename='js/jsencrypt.min.js') }}"></script>

  <style>
    body { font-family: Arial, sans-serif; margin: 30px; max-width: 600px; }
    h1 { color: #333; }
    textarea { width: 100%; height: 150px; font-size: 1rem; padding: 10px; margin-bottom:
15px; }
    button { padding: 10px 20px; font-size: 1rem; cursor: pointer; }
    .note { color: #555; font-size: 0.9rem; }
  </style>
</head>
<body>
  <h1>Напишіть нам повідомлення</h1>
  <h2>Будь-ласка надайте Ваше повне ім'я, що у Вас сталось та яка потрібна допомога.</h2>
  <h3>Ваше повідомлення буде зашифровано відправлено адміністраторам.</h3>

  <form id="report-form" method="POST" action="/submit">
    <label for="message">Ваше повідомлення:</label><br />
    <textarea id="message" placeholder="Напишіть тут ваше повідомлення..."
required></textarea><br />
    <input type="hidden" id="encrypted_message" name="encrypted_message" />
    <button type="submit">Надіслати</button>
  </form>

  <script>
    // Сюди Flask передає публічний ключ у PEM-форматі
    const publicKeyPEM = `{{ public_key | safe }}`.trim();
    console.log("PUBLIC KEY:", publicKeyPEM);

    document.addEventListener("DOMContentLoaded", function() {
      const form = document.getElementById("report-form");
      if (!form) {
        console.error("Не знайдено форму #report-form");
        return;
      }

      form.addEventListener("submit", function(event) {
        event.preventDefault();
        console.log("Натиснули submit…");

        const plainText = document.getElementById("message").value;
        console.log("Plain text:", plainText);

        if (!plainText) {
```

```
        alert("Будь ласка, введіть повідомлення.");
        return;
      }

      if (!publicKeyPEM) {
        alert("Помилка: публічний ключ не знайдено.");
        return;
      }

      // Використовуємо JSEncrypt (PKCS#1 v1.5) для шифрування
      const jsEncrypt = new JSEncrypt();
      jsEncrypt.setPublicKey(publicKeyPEM);
      const encrypted = jsEncrypt.encrypt(plainText);
      console.log("Encrypted:", encrypted);

      if (!encrypted) {
        alert("Помилка шифрування. Спробуйте ще раз.");
        return;
      }

      // Записуємо зашифрований base64 у приховане поле
      document.getElementById("encrypted_message").value = encrypted;
      console.log("Записали в hidden поле.");

      // Тепер сабмітимо форму звичайним POST
      form.submit();
    });
  });
</script>
</body>
</html>
```

Файл private_key.pem

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCjGHC39PKeeWOB
otZ4aTrHeLR+KRSYeGAXDqaQMd6BWlj+OeNU3RnFvHXEVyqQ7/eiNDZKor7SkSQZ
PhFkvlqWwTlVP+ZAJ9Ag2tQ95e2ndZd/CtqmwN6MmSddcoL6as7uUHhASbJUPhds
zVDawFCVAlSOBukDdEtX1Tm6THw8SvxKrUVowE9NeT0hsX1UZjNDsfQxktx9U6cb
23L0rptkZpuKRF45aJX5BxaUcqSi41OPryB3KgciRKNCiuwvWSbn6y3OptP4qeqA
6tXZGWMNou/muTCzyqtP/j4RcBMtZSqyOvj4ZQ7AGiYNzOjHdJVvAGaXq3Vu2nmt
aAp/KrY/AgMBAAECggEAUWNyaJJvep7oBbq9nUQiBYXaSHT2pqm3hfoVK7L/Lvzn
mSJJLEHTySXlfQ/kmYRuxr6Yc0FoyYqNgZ1EL8qASWfuVPj4zX4wshyRurvhj2Yy
rj6742gvBumsx8jLzOx9DFcS0z4TpE+CABUraqZ68AlEKcYQBN3FdXTeBHaDrtAO
PydneGd8SMiaO5m4pBFPYUOlKad5uV5UNrna7fCN/WAfvXZ/rP/uyxQQSbagkdWx
868Cg6dRzUu0WY2+Msud1zYksr1u9aL7mOapUFkj329SOGKM6QESHhd7iaqQUyUo
sI9JxBwiK7Eoe1PT8TtUvWl8BnYLCuUg2x1N+jUBsQKBgQDb4L8nlCWlX/AKqMHl
oPEJuk1LX0x38ocrL1grgA8vr7cO44iySalikFShBv5BNwn1s3v4yMQCi3Y+nQ4y
I8xX/kCWiSLggwsPSJeoxH/Wg0Tajsp9qeG2UD5EkiaTKNS7kNEtlaaIXD41YOjb
RZ9mnrA2kvvVRFKkIcjanIt80QKBgQC946Iw6nSXqe/jYktzyhB5sQJDJz1EQhhP
YE2VgcD3fSvIDQbxAAMNe9xAjZuiUyGTG9vP9I0nF7FR7q/a58ZuOgt/hvgoteXb
+e60zuoSC3K7RtatiBGU6MFd7dAg0uvWzmipK/mOHQomQH+pKe8HsWtppmK0//1j
XH17+1mGDwKBgDxhWnz27OOeFeAHeb+iU67VL1asY+Zofn/4b2D6uLDiswVurGIx
YkeD1QmnTvj/sUd8s2WdVF5Sh/Un0fco53uegYQVQrOeqNHYoHUjCOwSfjz6i1a4
6RfUr1TLQbaVtt2PJjI7b1Dw69BivW9BjkVy/HbwJCHrsjoKYQrPhGqRAoGASw+a
Y1+qavE/5FBMpx6u2IvB6y2tqDfASFA3GuxJjQsrrU6I+ecZTdEyRUo0xTu7P915
TIDM8zEIZOzHEu8fnx7OwMBnhCAzFnt7gnlwSWXBGopr/fiViplfJnQFG5SEqUq6
UHUTlruf8XL5g7MRy6a0YPTme8ndLdO3vrSitL0CgYEAhAzVGwNgzE1m1kPcqgiU
```

```
h+wkkxPlEj6XX971kc0U1ySfHmGZY3rooh32DOaoTK7KZBRa3q7LOb/CWiAvhlXo
xGZ6OxdjPsFgz09IwbK7Hatm3sKPMeYBeE6VzCH/Z88H6zgZBozhUw3muwhLcVTW
4nLcVE4KpPVtDcreVrElm40=
-----END PRIVATE KEY-----
```

Файл public_key.pem

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoxhwt/TynnljgaLWeGk6
x3i0fikUmHhgFw6mkDHegVpY/jnjVN0Zxbx1xFcqkO/3ojQ2SqK+0pEkGT4RZL5a
lsE5VT/mQCfQINrUPeXtp3WXfwrapsDejJknXXKC+mrO7lB4QEmyVD4XbM1Q2sBQ
lQJUjgbpA3RLV9U5ukx8PEr8Sq1FaMBPTXk9IbF9VGYzQ7H0MZLcfVOnG9ty9K6b
ZGabikReOWiV+QcWlHKkouNTj68gdyoHIkSjQorsL1km5+stzqbT+KnqgOrV2Rlj
DaLv5rkws8qrT/4+EXATLWUqsjr4+GUOwBomDczox3SVbwBml6t1btp5rWgKfyq2
PwIDAQAB
-----END PUBLIC KEY-----
```

Файл requirements.txt

```
python-telegram-bot>=20.0
Flask
cryptography
gunicorn
Redis
```

Файл jsencrypt.min.js

```
!function(t,e){"object"==typeof exports&&"object"==typeof
module?module.exports=e():"function"==typeof
define&&define.amd?define([],e):"object"==typeof
exports?exports.JSEncrypt=e():t.JSEncrypt=e()}(window,(()=>(()=>{var t={155:t=>{var
e,i,r=t.exports={};function n(){throw new Error("setTimeout has not been
defined")}function s(){throw new Error("clearTimeout has not been defined")}function
o(t){if(e===setTimeout)return setTimeout(t,0);if((e===n||!e)&&setTimeout)return
e=setTimeout,setTimeout(t,0);try{return e(t,0)}catch(i){try{return
e.call(null,t,0)}catch(i){return e.call(this,t,0)}}}!function(){try{e="function"==typeof
setTimeout?setTimeout:n}catch(t){e=n}try{i="function"==typeof
clearTimeout?clearTimeout:s}catch(t){i=s}}();var h,a=[],u=!1,c=-1;function
f(){u&&h&&(u=!1,h.length?a=h.concat(a):c=-1,a.length&&l())}function l(){if(!u){var
t=o(f);u=!0;for(var e=a.length;e;){for(h=a,a=[];++c<e;)h&&h[c].run();c=-
1,e=a.length}h=null,u=!1,function(t){if(i===clearTimeout)return
clearTimeout(t);if((i===s||!i)&&clearTimeout)return
i=clearTimeout,clearTimeout(t);try{i(t)}catch(e){try{return i.call(null,t)}catch(e){return
i.call(this,t)}}}(t)}}function p(t,e){this.fun=t,this.array=e}function
g(){}r.nextTick=function(t){var e=new Array(arguments.length-
1);if(arguments.length>1)for(var i=1;i<arguments.length;i++)e[i-1]=arguments[i];a.push(new
p(t,e)),1!==a.length||u||o(l)},p.prototype.run=function(){this.fun.apply(null,this.array)}
,r.title="browser",r.browser=!0,r.env={},r.argv=[],r.version="",r.versions={},r.on=g,r.add
Listener=g,r.once=g,r.off=g,r.removeListener=g,r.removeAllListeners=g,r.emit=g,r.prependLi
stener=g,r.prependOnceListener=g,r.listeners=function(t){return[]},r.binding=function(t){t
hrow new Error("process.binding is not
supported")},r.cwd=function(){return"/"},r.chdir=function(t){throw new
Error("process.chdir is not supported")},r.umask=function(){return 0}},e={};function
i(r){var n=e[r];if(void 0!==n)return n.exports;var s=e[r]={exports:{}};return
t[r](s,s.exports,i),s.exports}i.d=(t,e)=>{for(var r in
e)i.o(e,r)&&!i.o(t,r)&&Object.defineProperty(t,r,{enumerable:!0,get:e[r]})},i.o=(t,e)=>Obj
ect.prototype.hasOwnProperty.call(t,e);var r={};return(()=>{"use strict";function
```

```
t(t){return"0123456789abcdefghijklmnopqrstuvwxyz".charAt(t)}function e(t,e){return
t&e}function n(t,e){return t|e}function s(t,e){return t^e}function o(t,e){return
t&~e}function h(t){if(0==t)return-1;var e=0;return
0==(65535&t)&&(t>>=16,e+=16),0==(255&t)&&(t>>=8,e+=8),0==(15&t)&&(t>>=4,e+=4),0==(3&t)&&(t
>>=2,e+=2),0==(1&t)&&++e,e}function a(t){for(var e=0;0!=t;)t&=t-1,++e;return
e}i.d(r,{default:()=>ot});var
u,c="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";function f(t){var
e,i,r="";for(e=0;e+3<=t.length;e+=3)i=parseInt(t.substring(e,e+3),16),r+=c.charAt(i>>6)+c.
charAt(63&i);for(e+1==t.length?(i=parseInt(t.substring(e,e+1),16),r+=c.charAt(i<<2)):e+2==
t.length&&(i=parseInt(t.substring(e,e+2),16),r+=c.charAt(i>>2)+c.charAt((3&i)<<4));(3&r.le
ngth)>0;)r+="=";return r}function l(e){var
i,r="",n=0,s=0;for(i=0;i<e.length&&"="!=e.charAt(i);++i){var
o=c.indexOf(e.charAt(i));o<0||(0==n?(r+=t(o>>2),s=3&o,n=1):1==n?(r+=t(s<<2|o>>4),s=15&o,n=
2):2==n?(r+=t(s),r+=t(o>>2),s=3&o,n=3):(r+=t(s<<2|o>>4),r+=t(15&o),n=0))}return
1==n&&(r+=t(s<<2)),r}var p,g={decode:function(t){var e;if(void 0===p){var i="=
\f\n\r\t \u2028\u2029";for(p=Object.create(null),e=0;e<64;++e)p["ABCDEFGHIJKLMNOPQRSTUVWXY
Zabcdefghijklmnopqrstuvwxyz0123456789+/".charAt(e)]=e;for(p["-
"]=62,p._=63,e=0;e<i.length;++e)p[i.charAt(e)]=-1}var
r=[],n=0,s=0;for(e=0;e<t.length;++e){var o=t.charAt(e);if("="==o)break;if(-
1!=(o=p[o])){if(void 0===o)throw new Error("Illegal character at offset
"+e);n|=o,++s>=4?(r[r.length]=n>>16,r[r.length]=n>>8&255,r[r.length]=255&n,n=0,s=0):n<<=6}
}switch(s){case 1:throw new Error("Base64 encoding incomplete: at least 2 bits
missing");case 2:r[r.length]=n>>10;break;case
3:r[r.length]=n>>16,r[r.length]=n>>8&255}return r},re:/-----BEGIN [^-]+-----([A-Za-z0-
9+\/=\s]+)-----END [^-]+-----|begin-base64[^\n]+\n([A-Za-z0-
9+\/=\s]+)====/,unarmor:function(t){var
e=g.re.exec(t);if(e)if(e[1])t=e[1];else{if(!e[2])throw new Error("RegExp out of
sync");t=e[2]}return g.decode(t)}},d=1e13,v=function(){function
t(t){this.buf=[+t||0]}return t.prototype.mulAdd=function(t,e){var
i,r,n=this.buf,s=n.length;for(i=0;i<s;++i)(r=n[i]*t+e)<d?e=0:r-
=(e=0|r/d)*d,n[i]=r;e>0&&(n[i]=e)},t.prototype.sub=function(t){var
e,i,r=this.buf,n=r.length;for(e=0;e<n;++e)(i=r[e]-
t)<0?(i+=d,t=1):t=0,r[e]=i;for(;0===r[r.length-
1];)r.pop()},t.prototype.toString=function(t){if(10!=(t||10))throw new Error("only base 10
is supported");for(var e=this.buf,i=e[e.length-1].toString(),r=e.length-2;r>=0;--
r)i+=(d+e[r]).toString().substring(1);return i},t.prototype.valueOf=function(){for(var
t=this.buf,e=0,i=t.length-1;i>=0;--i)e=e*d+t[i];return
e},t.prototype.simplify=function(){var t=this.buf;return
1==t.length?t[0]:this},t}(),m=/^(\d\d)(0[1-9]|1[0-2])(0[1-9]|[12]\d|3[01])([01]\d|2[0-
3])(?:([0-5]\d)(?:([0-5]\d)(?:[.,](\d{1,3}))?)?)?(Z|[-+](?:[0]\d|1[0-2])([0-
5]\d)?)?$/,y=/^(\d\d\d\d)(0[1-9]|1[0-2])(0[1-9]|[12]\d|3[01])([01]\d|2[0-3])(?:([0-
5]\d)(?:([0-5]\d)(?:[.,](\d{1,3}))?)?)?(Z|[-+](?:[0]\d|1[0-2])([0-5]\d)?)?$/;function
b(t,e){return t.length>e&&(t=t.substring(0,e)+"…"),t}var T,S=function(){function
t(e,i){this.hexDigits="0123456789ABCDEF",e instanceof
t?(this.enc=e.enc,this.pos=e.pos):(this.enc=e,this.pos=i)}return
t.prototype.get=function(t){if(void 0===t&&(t=this.pos++),t>=this.enc.length)throw new
Error("Requesting byte offset ".concat(t," on a stream of length
").concat(this.enc.length));return"string"==typeof
this.enc?this.enc.charCodeAt(t):this.enc[t]},t.prototype.hexByte=function(t){return
this.hexDigits.charAt(t>>4&15)+this.hexDigits.charAt(15&t)},t.prototype.hexDump=function(t
,e,i){for(var r="",n=t;n<e;++n)if(r+=this.hexByte(this.get(n)),!0!==i)switch(15&n){case
7:r+="  ";break;case 15:r+="\n";break;default:r+=" "}return
r},t.prototype.isASCII=function(t,e){for(var i=t;i<e;++i){var
r=this.get(i);if(r<32||r>176)return!1}return!0},t.prototype.parseStringISO=function(t,e){f
or(var i="",r=t;r<e;++r)i+=String.fromCharCode(this.get(r));return
i},t.prototype.parseStringUTF=function(t,e){for(var i="",r=t;r<e;){var
n=this.get(r++);i+=n<128?String.fromCharCode(n):n>191&&n<224?String.fromCharCode((31&n)<<6
|63&this.get(r++)):String.fromCharCode((15&n)<<12|(63&this.get(r++))<<6|63&this.get(r++))}
return i},t.prototype.parseStringBMP=function(t,e){for(var
i,r,n="",s=t;s<e;)i=this.get(s++),r=this.get(s++),n+=String.fromCharCode(i<<8|r);return
```

```
n},t.prototype.parseTime=function(t,e,i){var
r=this.parseStringISO(t,e),n=(i?m:y).exec(r);return
n?(i&&(n[1]=+n[1],n[1]+=+n[1]<70?2e3:1900),r=n[1]+"-"+n[2]+"-"+n[3]+"
"+n[4],n[5]&&(r+=":"+n[5],n[6]&&(r+=":"+n[6],n[7]&&(r+="."+n[7]))),n[8]&&(r+="
UTC","Z"!=n[8]&&(r+=n[8],n[9]&&(r+=":"+n[9]))),r):"Unrecognized time:
"+r},t.prototype.parseInteger=function(t,e){for(var
i,r=this.get(t),n=r>127,s=n?255:0,o="";r==s&&++t<e;)r=this.get(t);if(0==(i=e-t))return n?-
1:0;if(i>4){for(o=r,i<<=3;0==(128&(+o^s));)o=+o<<1,--i;o="("+i+" bit)\n"}n&&(r-
=256);for(var h=new v(r),a=t+1;a<e;++a)h.mulAdd(256,this.get(a));return
o+h.toString()},t.prototype.parseBitString=function(t,e,i){for(var
r=this.get(t),n="("+((e-t-1<<3)-r)+" bit)\n",s="",o=t+1;o<e;++o){for(var
h=this.get(o),a=o==e-1?r:0,u=7;u>=a;--u)s+=h>>u&1?"1":"0";if(s.length>i)return
n+b(s,i)}return
n+s},t.prototype.parseOctetString=function(t,e,i){if(this.isASCII(t,e))return
b(this.parseStringISO(t,e),i);var r=e-t,n="("+r+" byte)\n";r>(i/=2)&&(e=t+i);for(var
s=t;s<e;++s)n+=this.hexByte(this.get(s));return
r>i&&(n+="…"),n},t.prototype.parseOID=function(t,e,i){for(var r="",n=new
v,s=0,o=t;o<e;++o){var
h=this.get(o);if(n.mulAdd(128,127&h),s+=7,!(128&h)){if(""===r)if((n=n.simplify())instanceo
f v)n.sub(80),r="2."+n.toString();else{var a=n<80?n<40?0:1:2;r=a+"."+(n-40*a)}else
r+="."+n.toString();if(r.length>i)return b(r,i);n=new v,s=0}}return
s>0&&(r+=".incomplete"),r},t}(),E=function(){function t(t,e,i,r,n){if(!(r instanceof
w))throw new Error("Invalid tag
value.");this.stream=t,this.header=e,this.length=i,this.tag=r,this.sub=n}return
t.prototype.typeName=function(){switch(this.tag.tagClass){case
0:switch(this.tag.tagNumber){case 0:return"EOC";case 1:return"BOOLEAN";case
2:return"INTEGER";case 3:return"BIT_STRING";case 4:return"OCTET_STRING";case
5:return"NULL";case 6:return"OBJECT_IDENTIFIER";case 7:return"ObjectDescriptor";case
8:return"EXTERNAL";case 9:return"REAL";case 10:return"ENUMERATED";case
11:return"EMBEDDED_PDV";case 12:return"UTF8String";case 16:return"SEQUENCE";case
17:return"SET";case 18:return"NumericString";case 19:return"PrintableString";case
20:return"TeletexString";case 21:return"VideotexString";case 22:return"IA5String";case
23:return"UTCTime";case 24:return"GeneralizedTime";case 25:return"GraphicString";case
26:return"VisibleString";case 27:return"GeneralString";case
28:return"UniversalString";case
30:return"BMPString"}return"Universal_"+this.tag.tagNumber.toString();case
1:return"Application_"+this.tag.tagNumber.toString();case
2:return"["+this.tag.tagNumber.toString()+"]";case
3:return"Private_"+this.tag.tagNumber.toString()}},t.prototype.content=function(t){if(void
0===this.tag)return null;void 0===t&&(t=1/0);var
e=this.posContent(),i=Math.abs(this.length);if(!this.tag.isUniversal())return
null!==this.sub?"("+this.sub.length+"
elem)":this.stream.parseOctetString(e,e+i,t);switch(this.tag.tagNumber){case 1:return
0===this.stream.get(e)?"false":"true";case 2:return this.stream.parseInteger(e,e+i);case
3:return this.sub?"("+this.sub.length+" elem)":this.stream.parseBitString(e,e+i,t);case
4:return this.sub?"("+this.sub.length+" elem)":this.stream.parseOctetString(e,e+i,t);case
6:return this.stream.parseOID(e,e+i,t);case 16:case 17:return
null!==this.sub?"("+this.sub.length+" elem)":"(no elem)";case 12:return
b(this.stream.parseStringUTF(e,e+i),t);case 18:case 19:case 20:case 21:case 22:case
26:return b(this.stream.parseStringISO(e,e+i),t);case 30:return
b(this.stream.parseStringBMP(e,e+i),t);case 23:case 24:return
this.stream.parseTime(e,e+i,23==this.tag.tagNumber)}return
null},t.prototype.toString=function(){return
this.typeName()+"@"+this.stream.pos+"[header:"+this.header+",length:"+this.length+",sub:"+
(null===this.sub?"null":this.sub.length)+"]"},t.prototype.toPrettyString=function(t){void
0===t&&(t="");var e=t+this.typeName()+"
@"+this.stream.pos;if(this.length>=0&&(e+="+"),e+=this.length,this.tag.tagConstructed?e+="
(constructed)":!this.tag.isUniversal()||3!=this.tag.tagNumber&&4!=this.tag.tagNumber||null
===this.sub||(e+=" (encapsulates)"),e+="\n",null!==this.sub){t+="  ";for(var
i=0,r=this.sub.length;i<r;++i)e+=this.sub[i].toPrettyString(t)}return
```

e},t.prototype.posStart=function(){return
this.stream.pos},t.prototype.posContent=function(){return
this.stream.pos+this.header},t.prototype.posEnd=function(){return
this.stream.pos+this.header+Math.abs(this.length)},t.prototype.toHexString=function(){retu
rn this.stream.hexDump(this.posStart(),this.posEnd(),!0)},t.decodeLength=function(t){var
e=t.get(),i=127&e;if(i==e)return i;if(i>6)throw new Error("Length over 48 bits not
supported at position "+(t.pos-1));if(0===i)return null;e=0;for(var
r=0;r<i;++r)e=256*e+t.get();return e},t.prototype.getHexStringValue=function(){var
t=this.toHexString(),e=2*this.header,i=2*this.length;return
t.substr(e,i)},t.decode=function(e){var i;i=e instanceof S?e:new S(e,0);var r=new
S(i),n=new w(i),s=t.decodeLength(i),o=i.pos,h=o-r.pos,a=null,u=function(){var
e=[];if(null!==s){for(var r=o+s;i.pos<r;)e[e.length]=t.decode(i);if(i.pos!=r)throw new
Error("Content size is not correct for container starting at offset "+o)}else
try{for(;;){var n=t.decode(i);if(n.tag.isEOC())break;e[e.length]=n}s=o-
i.pos}catch(t){throw new Error("Exception while decoding undefined length content:
"+t)}return e};if(n.tagConstructed)a=u();else
if(n.isUniversal()&&(3==n.tagNumber||4==n.tagNumber))try{if(3==n.tagNumber&&0!=i.get())thr
ow new Error("BIT STRINGs with unused bits cannot encapsulate.");a=u();for(var
c=0;c<a.length;++c)if(a[c].tag.isEOC())throw new Error("EOC is not supposed to be actual
content.")}catch(t){a=null}if(null===a){if(null===s)throw new Error("We can't skip over an
invalid tag with undefined length at offset "+o);i.pos=o+Math.abs(s)}return new
t(r,h,s,n,a)},t}(),w=function(){function t(t){var
e=t.get();if(this.tagClass=e>>6,this.tagConstructed=0!=(32&e),this.tagNumber=31&e,31==this
.tagNumber){var i=new
v;do{e=t.get(),i.mulAdd(128,127&e)}while(128&e);this.tagNumber=i.simplify()}}return
t.prototype.isUniversal=function(){return
0===this.tagClass},t.prototype.isEOC=function(){return
0===this.tagClass&&0===this.tagNumber},t}(),D=[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53
,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,17
9,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,
311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,44
3,449,457,461,463,467,479,487,491,499,503,509,521,523,541,547,557,563,569,571,577,587,593,
599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,727,733,73
9,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,
887,907,911,919,929,937,941,947,953,967,971,977,983,991,997],x=(1<<26)/D[D.length-
1],R=function(){function i(t,e,i){null!=t&&("number"==typeof
t?this.fromNumber(t,e,i):null==e&&"string"!=typeof
t?this.fromString(t,256):this.fromString(t,e))}return
i.prototype.toString=function(e){if(this.s<0)return"-"+this.negate().toString(e);var
i;if(16==e)i=4;else if(8==e)i=3;else if(2==e)i=1;else if(32==e)i=5;else{if(4!=e)return
this.toRadix(e);i=2}var r,n=(1<<i)-1,s=!1,o="",h=this.t,a=this.DB-h*this.DB%i;if(h--
>0)for(a<this.DB&&(r=this[h]>>a)>0&&(s=!0,o=t(r));h>=0;)a<i?(r=(this[h]&(1<<a)-1)<<i-
a,r|=this[--h]>>(a+=this.DB-i)):(r=this[h]>>(a-=i)&n,a<=0&&(a+=this.DB,--
h)),r>0&&(s=!0),s&&(o+=t(r));return s?o:"0"},i.prototype.negate=function(){var
t=I();return i.ZERO.subTo(this,t),t},i.prototype.abs=function(){return
this.s<0?this.negate():this},i.prototype.compareTo=function(t){var e=this.s-
t.s;if(0!=e)return e;var i=this.t;if(0!=(e=i-t.t))return this.s<0?-e:e;for(;--
i>=0;)if(0!=(e=this[i]-t[i]))return e;return 0},i.prototype.bitLength=function(){return
this.t<=0?0:this.DB*(this.t-1)+C(this[this.t-
1]^this.s&this.DM)},i.prototype.mod=function(t){var e=I();return
this.abs().divRemTo(t,null,e),this.s<0&&e.compareTo(i.ZERO)>0&&t.subTo(e,e),e},i.prototype
.modPowInt=function(t,e){var i;return i=t<256||e.isEven()?new O(e):new
A(e),this.exp(t,i)},i.prototype.clone=function(){var t=I();return
this.copyTo(t),t},i.prototype.intValue=function(){if(this.s<0){if(1==this.t)return
this[0]-this.DV;if(0==this.t)return-1}else{if(1==this.t)return this[0];if(0==this.t)return
0}return(this[1]&(1<<32-this.DB)-
1)<<this.DB|this[0]},i.prototype.byteValue=function(){return
0==this.t?this.s:this[0]<<24>>24},i.prototype.shortValue=function(){return
0==this.t?this.s:this[0]<<16>>16},i.prototype.signum=function(){return this.s<0?-
1:this.t<=0||1==this.t&&this[0]<=0?0:1},i.prototype.toByteArray=function(){var

```
t=this.t,e=[];e[0]=this.s;var i,r=this.DB-t*this.DB%8,n=0;if(t--
>0)for(r<this.DB&&(i=this[t]>>r)!=(this.s&this.DM)>>r&&(e[n++]=i|this.s<<this.DB-
r);t>=0;)r<8?(i=(this[t]&(1<<r)-1)<<8-r,i|=this[--t]>>(r+=this.DB-8)):(i=this[t]>>(r-
=8)&255,r<=0&&(r+=this.DB,--t)),0!=(128&i)&&(i|=-
256),0==n&&(128&this.s)!=(128&i)&&++n,(n>0||i!=this.s)&&(e[n++]=i);return
e},i.prototype.equals=function(t){return
0==this.compareTo(t)},i.prototype.min=function(t){return
this.compareTo(t)<0?this:t},i.prototype.max=function(t){return
this.compareTo(t)>0?this:t},i.prototype.and=function(t){var i=I();return
this.bitwiseTo(t,e,i),i},i.prototype.or=function(t){var e=I();return
this.bitwiseTo(t,n,e),e},i.prototype.xor=function(t){var e=I();return
this.bitwiseTo(t,s,e),e},i.prototype.andNot=function(t){var e=I();return
this.bitwiseTo(t,o,e),e},i.prototype.not=function(){for(var
t=I(),e=0;e<this.t;++e)t[e]=this.DM&~this[e];return
t.t=this.t,t.s=~this.s,t},i.prototype.shiftLeft=function(t){var e=I();return
t<0?this.rShiftTo(-t,e):this.lShiftTo(t,e),e},i.prototype.shiftRight=function(t){var
e=I();return t<0?this.lShiftTo(-
t,e):this.rShiftTo(t,e),e},i.prototype.getLowestSetBit=function(){for(var
t=0;t<this.t;++t)if(0!=this[t])return t*this.DB+h(this[t]);return
this.s<0?this.t*this.DB:-1},i.prototype.bitCount=function(){for(var
t=0,e=this.s&this.DM,i=0;i<this.t;++i)t+=a(this[i]^e);return
t},i.prototype.testBit=function(t){var e=Math.floor(t/this.DB);return
e>=this.t?0!=this.s:0!=(this[e]&1<<t%this.DB)},i.prototype.setBit=function(t){return
this.changeBit(t,n)},i.prototype.clearBit=function(t){return
this.changeBit(t,o)},i.prototype.flipBit=function(t){return
this.changeBit(t,s)},i.prototype.add=function(t){var e=I();return
this.addTo(t,e),e},i.prototype.subtract=function(t){var e=I();return
this.subTo(t,e),e},i.prototype.multiply=function(t){var e=I();return
this.multiplyTo(t,e),e},i.prototype.divide=function(t){var e=I();return
this.divRemTo(t,e,null),e},i.prototype.remainder=function(t){var e=I();return
this.divRemTo(t,null,e),e},i.prototype.divideAndRemainder=function(t){var
e=I(),i=I();return this.divRemTo(t,e,i),[e,i]},i.prototype.modPow=function(t,e){var
i,r,n=t.bitLength(),s=H(1);if(n<=0)return s;i=n<18?1:n<48?3:n<144?4:n<768?5:6,r=n<8?new
O(e):e.isEven()?new V(e):new A(e);var o=[],h=3,a=i-1,u=(1<<i)-
1;if(o[1]=r.convert(this),i>1){var c=I();for(r.sqrTo(o[1],c);h<=u;)o[h]=I(),r.mulTo(c,o[h-
2],o[h]),h+=2}var f,l,p=t.t-1,g=!0,d=I();for(n=C(t[p])-1;p>=0;){for(n>=a?f=t[p]>>n-
a&u:(f=(t[p]&(1<<n+1)-1)<<a-n,p>0&&(f|=t[p-1]>>this.DB+n-a)),h=i;0==(1&f);)f>>=1,--
h;if((n-=h)<0&&(n+=this.DB,--
p),g)o[f].copyTo(s),g=!1;else{for(;h>1;)r.sqrTo(s,d),r.sqrTo(d,s),h-
=2;h>0?r.sqrTo(s,d):(l=s,s=d,d=l),r.mulTo(d,o[f],s)}for(;p>=0&&0==(t[p]&1<<n);)r.sqrTo(s,d
),l=s,s=d,d=l,--n<0&&(n=this.DB-1,--p)}return
r.revert(s)},i.prototype.modInverse=function(t){var
e=t.isEven();if(this.isEven()&&e||0==t.signum())return i.ZERO;for(var
r=t.clone(),n=this.clone(),s=H(1),o=H(0),h=H(0),a=H(1);0!=r.signum();){for(;r.isEven();)r.
rShiftTo(1,r),e?(s.isEven()&&o.isEven()||(s.addTo(this,s),o.subTo(t,o)),s.rShiftTo(1,s)):o
.isEven()||o.subTo(t,o),o.rShiftTo(1,o);for(;n.isEven();)n.rShiftTo(1,n),e?(h.isEven()&&a.
isEven()||(h.addTo(this,h),a.subTo(t,a)),h.rShiftTo(1,h)):a.isEven()||a.subTo(t,a),a.rShif
tTo(1,a);r.compareTo(n)>=0?(r.subTo(n,r),e&&s.subTo(h,s),o.subTo(a,o)):(n.subTo(r,n),e&&h.
subTo(s,h),a.subTo(o,a))}return
0!=n.compareTo(i.ONE)?i.ZERO:a.compareTo(t)>=0?a.subtract(t):a.signum()<0?(a.addTo(t,a),a.
signum()<0?a.add(t):a):a},i.prototype.pow=function(t){return this.exp(t,new
B)},i.prototype.gcd=function(t){var
e=this.s<0?this.negate():this.clone(),i=t.s<0?t.negate():t.clone();if(e.compareTo(i)<0){va
r r=e;e=i,i=r}var n=e.getLowestSetBit(),s=i.getLowestSetBit();if(s<0)return
e;for(n<s&&(s=n),s>0&&(e.rShiftTo(s,e),i.rShiftTo(s,i));e.signum()>0;)(n=e.getLowestSetBit
())>0&&e.rShiftTo(n,e),(n=i.getLowestSetBit())>0&&i.rShiftTo(n,i),e.compareTo(i)>=0?(e.sub
To(i,e),e.rShiftTo(1,e)):(i.subTo(e,i),i.rShiftTo(1,i));return
s>0&&i.lShiftTo(s,i),i},i.prototype.isProbablePrime=function(t){var
e,i=this.abs();if(1==i.t&&i[0]<=D[D.length-
1]){for(e=0;e<D.length;++e)if(i[0]==D[e])return!0;return!1}if(i.isEven())return!1;for(e=1;
```

```
e<D.length;){for(var
r=D[e],n=e+1;n<D.length&&r<x;)r*=D[n++];for(r=i.modInt(r);e<n;)if(r%D[e++]==0)return!1}ret
urn i.millerRabin(t)},i.prototype.copyTo=function(t){for(var e=this.t-1;e>=0;--
e)t[e]=this[e];t.t=this.t,t.s=this.s},i.prototype.fromInt=function(t){this.t=1,this.s=t<0?
-1:0,t>0?this[0]=t:t<-
1?this[0]=t+this.DV:this.t=0},i.prototype.fromString=function(t,e){var r;if(16==e)r=4;else
if(8==e)r=3;else if(256==e)r=8;else if(2==e)r=1;else if(32==e)r=5;else{if(4!=e)return void
this.fromRadix(t,e);r=2}this.t=0,this.s=0;for(var n=t.length,s=!1,o=0;--n>=0;){var
h=8==r?255&+t[n]:q(t,n);h<0?"-
"==t.charAt(n)&&(s=!0):(s=!1,0==o?this[this.t++]=h:o+r>this.DB?(this[this.t-
1]|=(h&(1<<this.DB-o)-1)<<o,this[this.t++]=h>>this.DB-o):this[this.t-
1]|=h<<o,(o+=r)>=this.DB&&(o-=this.DB))}8==r&&0!=(128&+t[0])&&(this.s=-
1,o>0&&(this[this.t-1]|=(1<<this.DB-o)-
1<<o)),this.clamp(),s&&i.ZERO.subTo(this,this)},i.prototype.clamp=function(){for(var
t=this.s&this.DM;this.t>0&&this[this.t-1]==t;)--
this.t},i.prototype.dlShiftTo=function(t,e){var i;for(i=this.t-1;i>=0;--
i)e[i+t]=this[i];for(i=t-1;i>=0;--
i)e[i]=0;e.t=this.t+t,e.s=this.s},i.prototype.drShiftTo=function(t,e){for(var
i=t;i<this.t;++i)e[i-t]=this[i];e.t=Math.max(this.t-
t,0),e.s=this.s},i.prototype.lShiftTo=function(t,e){for(var i=t%this.DB,r=this.DB-
i,n=(1<<r)-1,s=Math.floor(t/this.DB),o=this.s<<i&this.DM,h=this.t-1;h>=0;--
h)e[h+s+1]=this[h]>>r|o,o=(this[h]&n)<<i;for(h=s-1;h>=0;--
h)e[h]=0;e[s]=o,e.t=this.t+s+1,e.s=this.s,e.clamp()},i.prototype.rShiftTo=function(t,e){e.
s=this.s;var i=Math.floor(t/this.DB);if(i>=this.t)e.t=0;else{var r=t%this.DB,n=this.DB-
r,s=(1<<r)-1;e[0]=this[i]>>r;for(var o=i+1;o<this.t;++o)e[o-i-1]|=(this[o]&s)<<n,e[o-
i]=this[o]>>r;r>0&&(e[this.t-i-1]|=(this.s&s)<<n),e.t=this.t-
i,e.clamp()}},i.prototype.subTo=function(t,e){for(var
i=0,r=0,n=Math.min(t.t,this.t);i<n;)r+=this[i]-
t[i],e[i++]=r&this.DM,r>>=this.DB;if(t.t<this.t){for(r-
=t.s;i<this.t;)r+=this[i],e[i++]=r&this.DM,r>>=this.DB;r+=this.s}else{for(r+=this.s;i<t.t;
)r-=t[i],e[i++]=r&this.DM,r>>=this.DB;r-=t.s}e.s=r<0?-1:0,r<-
1?e[i++]=this.DV+r:r>0&&(e[i++]=r),e.t=i,e.clamp()},i.prototype.multiplyTo=function(t,e){v
ar r=this.abs(),n=t.abs(),s=r.t;for(e.t=s+n.t;--
s>=0;)e[s]=0;for(s=0;s<n.t;++s)e[s+r.t]=r.am(0,n[s],e,s,0,r.t);e.s=0,e.clamp(),this.s!=t.s
&&i.ZERO.subTo(e,e)},i.prototype.squareTo=function(t){for(var e=this.abs(),i=t.t=2*e.t;--
i>=0;)t[i]=0;for(i=0;i<e.t-1;++i){var
r=e.am(i,e[i],t,2*i,0,1);(t[i+e.t]+=e.am(i+1,2*e[i],t,2*i+1,r,e.t-i-1))>=e.DV&&(t[i+e.t]-
=e.DV,t[i+e.t+1]=1)}t.t>0&&(t[t.t-
1]+=e.am(i,e[i],t,2*i,0,1)),t.s=0,t.clamp()},i.prototype.divRemTo=function(t,e,r){var
n=t.abs();if(!(n.t<=0)){var s=this.abs();if(s.t<n.t)return
null!=e&&e.fromInt(0),void(null!=r&&this.copyTo(r));null==r&&(r=I());var
o=I(),h=this.s,a=t.s,u=this.DB-C(n[n.t-
1]);u>0?(n.lShiftTo(u,o),s.lShiftTo(u,r)):(n.copyTo(o),s.copyTo(r));var c=o.t,f=o[c-
1];if(0!=f){var l=f*(1<<this.F1)+(c>1?o[c-
2]>>this.F2:0),p=this.FV/l,g=(1<<this.F1)/l,d=1<<this.F2,v=r.t,m=v-
c,y=null==e?I():e;for(o.dlShiftTo(m,y),r.compareTo(y)>=0&&(r[r.t++]=1,r.subTo(y,r)),i.ONE.
dlShiftTo(c,y),y.subTo(o,o);o.t<c;)o[o.t++]=0;for(;--m>=0;){var b=r[--
v]==f?this.DM:Math.floor(r[v]*p+(r[v-
1]+d)*g);if((r[v]+=o.am(0,b,r,m,0,c))<b)for(o.dlShiftTo(m,y),r.subTo(y,r);r[v]<--
b;)r.subTo(y,r)}null!=e&&(r.drShiftTo(c,e),h!=a&&i.ZERO.subTo(e,e)),r.t=c,r.clamp(),u>0&&r
.rShiftTo(u,r),h<0&&i.ZERO.subTo(r,r)}}},i.prototype.invDigit=function(){if(this.t<1)retur
n 0;var t=this[0];if(0==(1&t))return 0;var e=3&t;return(e=(e=(e=(e=e*(2-(15&t)*e)&15)*(2-
(255&t)*e)&255)*(2-((65535&t)*e&65535))&65535)*(2-t*e%this.DV)%this.DV)>0?this.DV-e:-
e},i.prototype.isEven=function(){return
0==(this.t>0?1&this[0]:this.s)},i.prototype.exp=function(t,e){if(t>4294967295||t<1)return
i.ONE;var r=I(),n=I(),s=e.convert(this),o=C(t)-1;for(s.copyTo(r);--
o>=0;)if(e.sqrTo(r,n),(t&1<<o)>0)e.mulTo(n,s,r);else{var h=r;r=n,n=h}return
e.revert(r)},i.prototype.chunkSize=function(t){return
Math.floor(Math.LN2*this.DB/Math.log(t))},i.prototype.toRadix=function(t){if(null==t&&(t=1
0),0==this.signum()||t<2||t>36)return"0";var
```

```
e=this.chunkSize(t),i=Math.pow(t,e),r=H(i),n=I(),s=I(),o="";for(this.divRemTo(r,n,s);n.sig
num()>0;)o=(i+s.intValue()).toString(t).substr(1)+o,n.divRemTo(r,n,s);return
s.intValue().toString(t)+o},i.prototype.fromRadix=function(t,e){this.fromInt(0),null==e&&(
e=10);for(var r=this.chunkSize(e),n=Math.pow(e,r),s=!1,o=0,h=0,a=0;a<t.length;++a){var
u=q(t,a);u<0?"-
"==t.charAt(a)&&0==this.signum()&&(s=!0):(h=e*h+u,++o>=r&&(this.dMultiply(n),this.dAddOffs
et(h,0),o=0,h=0))}o>0&&(this.dMultiply(Math.pow(e,o)),this.dAddOffset(h,0)),s&&i.ZERO.subT
o(this,this)},i.prototype.fromNumber=function(t,e,r){if("number"==typeof
e)if(t<2)this.fromInt(1);else for(this.fromNumber(t,r),this.testBit(t-
1)||this.bitwiseTo(i.ONE.shiftLeft(t-
1),n,this),this.isEven()&&this.dAddOffset(1,0);!this.isProbablePrime(e);)this.dAddOffset(2
,0),this.bitLength()>t&&this.subTo(i.ONE.shiftLeft(t-1),this);else{var
s=[],o=7&t;s.length=1+(t>>3),e.nextBytes(s),o>0?s[0]&=(1<<o)-
1:s[0]=0,this.fromString(s,256)}},i.prototype.bitwiseTo=function(t,e,i){var
r,n,s=Math.min(t.t,this.t);for(r=0;r<s;++r)i[r]=e(this[r],t[r]);if(t.t<this.t){for(n=t.s&t
his.DM,r=s;r<this.t;++r)i[r]=e(this[r],n);i.t=this.t}else{for(n=this.s&this.DM,r=s;r<t.t;+
+r)i[r]=e(n,t[r]);i.t=t.t}i.s=e(this.s,t.s),i.clamp()},i.prototype.changeBit=function(t,e)
{var r=i.ONE.shiftLeft(t);return
this.bitwiseTo(r,e,r),r},i.prototype.addTo=function(t,e){for(var
i=0,r=0,n=Math.min(t.t,this.t);i<n;)r+=this[i]+t[i],e[i++]=r&this.DM,r>>=this.DB;if(t.t<th
is.t){for(r+=t.s;i<this.t;)r+=this[i],e[i++]=r&this.DM,r>>=this.DB;r+=this.s}else{for(r+=t
his.s;i<t.t;)r+=t[i],e[i++]=r&this.DM,r>>=this.DB;r+=t.s}e.s=r<0?-1:0,r>0?e[i++]=r:r<-
1&&(e[i++]=this.DV+r),e.t=i,e.clamp()},i.prototype.dMultiply=function(t){this[this.t]=this
.am(0,t-
1,this,0,0,this.t),++this.t,this.clamp()},i.prototype.dAddOffset=function(t,e){if(0!=t){fo
r(;this.t<=e;)this[this.t++]=0;for(this[e]+=t;this[e]>=this.DV;)this[e]-
=this.DV,++e>=this.t&&(this[this.t++]=0),++this[e]}},i.prototype.multiplyLowerTo=function(
t,e,i){var r=Math.min(this.t+t.t,e);for(i.s=0,i.t=r;r>0;)i[--r]=0;for(var n=i.t-
this.t;r<n;++r)i[r+this.t]=this.am(0,t[r],i,r,0,this.t);for(n=Math.min(t.t,e);r<n;++r)this
.am(0,t[r],i,r,0,e-r);i.clamp()},i.prototype.multiplyUpperTo=function(t,e,i){--e;var
r=i.t=this.t+t.t-e;for(i.s=0;--r>=0;)i[r]=0;for(r=Math.max(e-
this.t,0);r<t.t;++r)i[this.t+r-e]=this.am(e-r,t[r],i,0,0,this.t+r-
e);i.clamp(),i.drShiftTo(1,i)},i.prototype.modInt=function(t){if(t<=0)return 0;var
e=this.DV%t,i=this.s<0?t-1:0;if(this.t>0)if(0==e)i=this[0]%t;else for(var r=this.t-
1;r>=0;--r)i=(e*i+this[r])%t;return i},i.prototype.millerRabin=function(t){var
e=this.subtract(i.ONE),r=e.getLowestSetBit();if(r<=0)return!1;var
n=e.shiftRight(r);(t=t+1>>1)>D.length&&(t=D.length);for(var
s=I(),o=0;o<t;++o){s.fromInt(D[Math.floor(Math.random()*D.length)]);var
h=s.modPow(n,this);if(0!=h.compareTo(i.ONE)&&0!=h.compareTo(e)){for(var
a=1;a++<r&&0!=h.compareTo(e);)if(0==(h=h.modPowInt(2,this)).compareTo(i.ONE))return!1;if(0
!=h.compareTo(e))return!1}}return!0},i.prototype.square=function(){var t=I();return
this.squareTo(t),t},i.prototype.gcda=function(t,e){var
i=this.s<0?this.negate():this.clone(),r=t.s<0?t.negate():t.clone();if(i.compareTo(r)<0){va
r n=i;i=r,r=n}var
s=i.getLowestSetBit(),o=r.getLowestSetBit();if(o<0)e(i);else{s<o&&(o=s),o>0&&(i.rShiftTo(o
,i),r.rShiftTo(o,r));var
h=function(){(s=i.getLowestSetBit())>0&&i.rShiftTo(s,i),(s=r.getLowestSetBit())>0&&r.rShif
tTo(s,r),i.compareTo(r)>=0?(i.subTo(r,i),i.rShiftTo(1,i)):(r.subTo(i,r),r.rShiftTo(1,r)),i
.signum()>0?setTimeout(h,0):(o>0&&r.lShiftTo(o,r),setTimeout((function(){e(r)}),0))};setTi
meout(h,10)}},i.prototype.fromNumberAsync=function(t,e,r,s){if("number"==typeof
e)if(t<2)this.fromInt(1);else{this.fromNumber(t,r),this.testBit(t-
1)||this.bitwiseTo(i.ONE.shiftLeft(t-1),n,this),this.isEven()&&this.dAddOffset(1,0);var
o=this,h=function(){o.dAddOffset(2,0),o.bitLength()>t&&o.subTo(i.ONE.shiftLeft(t-
1),o),o.isProbablePrime(e)?setTimeout((function(){s()}),0):setTimeout(h,0)};setTimeout(h,0
)}else{var a=[],u=7&t;a.length=1+(t>>3),e.nextBytes(a),u>0?a[0]&=(1<<u)-
1:a[0]=0,this.fromString(a,256)}},i}(),B=function(){function t(){}return
t.prototype.convert=function(t){return t},t.prototype.revert=function(t){return
t},t.prototype.mulTo=function(t,e,i){t.multiplyTo(e,i)},t.prototype.sqrTo=function(t,e){t.
squareTo(e)},t}(),O=function(){function t(t){this.m=t}return
t.prototype.convert=function(t){return
```

```
t.s<0||t.compareTo(this.m)>=0?t.mod(this.m):t},t.prototype.revert=function(t){return
t},t.prototype.reduce=function(t){t.divRemTo(this.m,null,t)},t.prototype.mulTo=function(t,
e,i){t.multiplyTo(e,i),this.reduce(i)},t.prototype.sqrTo=function(t,e){t.squareTo(e),this.
reduce(e)},t}(),A=function(){function
t(t){this.m=t,this.mp=t.invDigit(),this.mpl=32767&this.mp,this.mph=this.mp>>15,this.um=(1<
<t.DB-15)-1,this.mt2=2*t.t}return t.prototype.convert=function(t){var e=I();return
t.abs().dlShiftTo(this.m.t,e),e.divRemTo(this.m,null,e),t.s<0&&e.compareTo(R.ZERO)>0&&this
.m.subTo(e,e),e},t.prototype.revert=function(t){var e=I();return
t.copyTo(e),this.reduce(e),e},t.prototype.reduce=function(t){for(;t.t<=this.mt2;)t[t.t++]=
0;for(var e=0;e<this.m.t;++e){var
i=32767&t[e],r=i*this.mpl+((i*this.mph+(t[e]>>15)*this.mpl&this.um)<<15)&t.DM;for(t[i=e+th
is.m.t]+=this.m.am(0,r,t,e,0,this.m.t);t[i]>=t.DV;)t[i]-
=t.DV,t[++i]++}t.clamp(),t.drShiftTo(this.m.t,t),t.compareTo(this.m)>=0&&t.subTo(this.m,t)
},t.prototype.mulTo=function(t,e,i){t.multiplyTo(e,i),this.reduce(i)},t.prototype.sqrTo=fu
nction(t,e){t.squareTo(e),this.reduce(e)},t}(),V=function(){function
t(t){this.m=t,this.r2=I(),this.q3=I(),R.ONE.dlShiftTo(2*t.t,this.r2),this.mu=this.r2.divid
e(t)}return t.prototype.convert=function(t){if(t.s<0||t.t>2*this.m.t)return
t.mod(this.m);if(t.compareTo(this.m)<0)return t;var e=I();return
t.copyTo(e),this.reduce(e),e},t.prototype.revert=function(t){return
t},t.prototype.reduce=function(t){for(t.drShiftTo(this.m.t-
1,this.r2),t.t>this.m.t+1&&(t.t=this.m.t+1,t.clamp()),this.mu.multiplyUpperTo(this.r2,this
.m.t+1,this.q3),this.m.multiplyLowerTo(this.q3,this.m.t+1,this.r2);t.compareTo(this.r2)<0;
)t.dAddOffset(1,this.m.t+1);for(t.subTo(this.r2,t);t.compareTo(this.m)>=0;)t.subTo(this.m,
t)},t.prototype.mulTo=function(t,e,i){t.multiplyTo(e,i),this.reduce(i)},t.prototype.sqrTo=
function(t,e){t.squareTo(e),this.reduce(e)},t}();function I(){return new R(null)}function
N(t,e){return new R(t,e)}var P="undefined"!=typeof navigator;P&&"Microsoft Internet
Explorer"==navigator.appName?(R.prototype.am=function(t,e,i,r,n,s){for(var
o=32767&e,h=e>>15;--s>=0;){var
a=32767&this[t],u=this[t++]>>15,c=h*a+u*o;n=((a=o*a+((32767&c)<<15)+i[r]+(1073741823&n))>>
>30)+(c>>>15)+h*u+(n>>>30),i[r++]=1073741823&a}return
n},T=30):P&&"Netscape"!=navigator.appName?(R.prototype.am=function(t,e,i,r,n,s){for(;--
s>=0;){var o=e*this[t++]+i[r]+n;n=Math.floor(o/67108864),i[r++]=67108863&o}return
n},T=26):(R.prototype.am=function(t,e,i,r,n,s){for(var o=16383&e,h=e>>14;--s>=0;){var
a=16383&this[t],u=this[t++]>>14,c=h*a+u*o;n=((a=o*a+((16383&c)<<14)+i[r]+n)>>28)+(c>>14)+h
*u,i[r++]=268435455&a}return n},T=28),R.prototype.DB=T,R.prototype.DM=(1<<T)-
1,R.prototype.DV=1<<T,R.prototype.FV=Math.pow(2,52),R.prototype.F1=52-
T,R.prototype.F2=2*T-52;var
M,L,j=[];for(M="0".charCodeAt(0),L=0;L<=9;++L)j[M++]=L;for(M="a".charCodeAt(0),L=10;L<36;+
+L)j[M++]=L;for(M="A".charCodeAt(0),L=10;L<36;++L)j[M++]=L;function q(t,e){var
i=j[t.charCodeAt(e)];return null==i?-1:i}function H(t){var e=I();return
e.fromInt(t),e}function C(t){var e,i=1;return
0!=(e=t>>>16)&&(t=e,i+=16),0!=(e=t>>8)&&(t=e,i+=8),0!=(e=t>>4)&&(t=e,i+=4),0!=(e=t>>2)&&(t
=e,i+=2),0!=(e=t>>1)&&(t=e,i+=1),i}R.ZERO=H(0),R.ONE=H(1);var F,U,K=function(){function
t(){this.i=0,this.j=0,this.S=[]}return t.prototype.init=function(t){var
e,i,r;for(e=0;e<256;++e)this.S[e]=e;for(i=0,e=0;e<256;++e)i=i+this.S[e]+t[e%t.length]&255,
r=this.S[e],this.S[e]=this.S[i],this.S[i]=r;this.i=0,this.j=0},t.prototype.next=function()
{var t;return
this.i=this.i+1&255,this.j=this.j+this.S[this.i]&255,t=this.S[this.i],this.S[this.i]=this.
S[this.j],this.S[this.j]=t,this.S[t+this.S[this.i]&255]},t}(),k=null;if(null==k){k=[],U=0;
var _=void 0;if("undefined"!=typeof
window&&window.crypto&&window.crypto.getRandomValues){var z=new
Uint32Array(256);for(window.crypto.getRandomValues(z),_=0;_<z.length;++_)k[U++]=255&z[_]}v
ar
Z=0,G=function(t){if((Z=Z||0)>=256||U>=256)window.removeEventListener?window.removeEventLi
stener("mousemove",G,!1):window.detachEvent&&window.detachEvent("onmousemove",G);else
try{var e=t.x+t.y;k[U++]=255&e,Z+=1}catch(t){}};"undefined"!=typeof
window&&(window.addEventListener?window.addEventListener("mousemove",G,!1):window.attachEv
ent&&window.attachEvent("onmousemove",G))}function $(){if(null==F){for(F=new K;U<256;){var
t=Math.floor(65536*Math.random());k[U++]=255&t}for(F.init(k),U=0;U<k.length;++U)k[U]=0;U=0
}return F.next()}var Y=function(){function t(){}return
```

```
t.prototype.nextBytes=function(t){for(var
e=0;e<t.length;++e)t[e]=$()},t}(),J=function(){function
t(){this.n=null,this.e=0,this.d=null,this.p=null,this.q=null,this.dmp1=null,this.dmq1=null
,this.coeff=null}return t.prototype.doPublic=function(t){return
t.modPowInt(this.e,this.n)},t.prototype.doPrivate=function(t){if(null==this.p||null==this.
q)return t.modPow(this.d,this.n);for(var
e=t.mod(this.p).modPow(this.dmp1,this.p),i=t.mod(this.q).modPow(this.dmq1,this.q);e.compar
eTo(i)<0;)e=e.add(this.p);return
e.subtract(i).multiply(this.coeff).mod(this.p).multiply(this.q).add(i)},t.prototype.setPub
lic=function(t,e){null!=t&&null!=e&&t.length>0&&e.length>0?(this.n=N(t,16),this.e=parseInt
(e,16)):console.error("Invalid RSA public key")},t.prototype.encrypt=function(t){var
e=this.n.bitLength()+7>>3,i=function(t,e){if(e<t.length+11)return console.error("Message
too long for RSA"),null;for(var i=[],r=t.length-1;r>=0&&e>0;){var n=t.charCodeAt(r--
);n<128?i[--e]=n:n>127&&n<2048?(i[--e]=63&n|128,i[--e]=n>>6|192):(i[--e]=63&n|128,i[--
e]=n>>6&63|128,i[--e]=n>>12|224)}i[--e]=0;for(var s=new
Y,o=[];e>2;){for(o[0]=0;0==o[0];)s.nextBytes(o);i[--e]=o[0]}return i[--e]=2,i[--e]=0,new
R(i)}(t,e);if(null==i)return null;var r=this.doPublic(i);if(null==r)return null;for(var
n=r.toString(16),s=n.length,o=0;o<2*e-s;o++)n="0"+n;return
n},t.prototype.setPrivate=function(t,e,i){null!=t&&null!=e&&t.length>0&&e.length>0?(this.n
=N(t,16),this.e=parseInt(e,16),this.d=N(i,16)):console.error("Invalid RSA private
key")},t.prototype.setPrivateEx=function(t,e,i,r,n,s,o,h){null!=t&&null!=e&&t.length>0&&e.
length>0?(this.n=N(t,16),this.e=parseInt(e,16),this.d=N(i,16),this.p=N(r,16),this.q=N(n,16
),this.dmp1=N(s,16),this.dmq1=N(o,16),this.coeff=N(h,16)):console.error("Invalid RSA
private key")},t.prototype.generate=function(t,e){var i=new
Y,r=t>>1;this.e=parseInt(e,16);for(var n=new R(e,16);;){for(;this.p=new R(t-
r,1,i),0!=this.p.subtract(R.ONE).gcd(n).compareTo(R.ONE)||!this.p.isProbablePrime(10););fo
r(;this.q=new
R(r,1,i),0!=this.q.subtract(R.ONE).gcd(n).compareTo(R.ONE)||!this.q.isProbablePrime(10););
if(this.p.compareTo(this.q)<=0){var s=this.p;this.p=this.q,this.q=s}var
o=this.p.subtract(R.ONE),h=this.q.subtract(R.ONE),a=o.multiply(h);if(0==a.gcd(n).compareTo
(R.ONE)){this.n=this.p.multiply(this.q),this.d=n.modInverse(a),this.dmp1=this.d.mod(o),thi
s.dmq1=this.d.mod(h),this.coeff=this.q.modInverse(this.p);break}}},t.prototype.decrypt=fun
ction(t){var e=N(t,16),i=this.doPrivate(e);return null==i?null:function(t,e){for(var
i=t.toByteArray(),r=0;r<i.length&&0==i[r];)++r;if(i.length-r!=e-1||2!=i[r])return
null;for(++r;0!=i[r];)if(++r>=i.length)return null;for(var n="";++r<i.length;){var
s=255&i[r];s<128?n+=String.fromCharCode(s):s>191&&s<224?(n+=String.fromCharCode((31&s)<<6|
63&i[r+1]),++r):(n+=String.fromCharCode((15&s)<<12|(63&i[r+1])<<6|63&i[r+2]),r+=2)}return
n}(i,this.n.bitLength()+7>>3)},t.prototype.generateAsync=function(t,e,i){var r=new
Y,n=t>>1;this.e=parseInt(e,16);var s=new R(e,16),o=this,h=function(){var
e=function(){if(o.p.compareTo(o.q)<=0){var t=o.p;o.p=o.q,o.q=t}var
e=o.p.subtract(R.ONE),r=o.q.subtract(R.ONE),n=e.multiply(r);0==n.gcd(s).compareTo(R.ONE)?(
o.n=o.p.multiply(o.q),o.d=s.modInverse(n),o.dmp1=o.d.mod(e),o.dmq1=o.d.mod(r),o.coeff=o.q.
modInverse(o.p),setTimeout((function(){i()}),0)):setTimeout(h,0)},a=function(){o.q=I(),o.q
.fromNumberAsync(n,1,r,(function(){o.q.subtract(R.ONE).gcda(s,(function(t){0==t.compareTo(
R.ONE)&&o.q.isProbablePrime(10)?setTimeout(e,0):setTimeout(a,0)}))}))},u=function(){o.p=I(
),o.p.fromNumberAsync(t-
n,1,r,(function(){o.p.subtract(R.ONE).gcda(s,(function(t){0==t.compareTo(R.ONE)&&o.p.isPro
bablePrime(10)?setTimeout(a,0):setTimeout(u,0)}))}))};setTimeout(u,0)};setTimeout(h,0)},t.
prototype.sign=function(t,e,i){var r=function(t,e){if(e<t.length+22)return
console.error("Message too long for RSA"),null;for(var i=e-t.length-
6,r="",n=0;n<i;n+=2)r+="ff";return
N("0001"+r+"00"+t,16)}((X[i]||"")+e(t).toString(),this.n.bitLength()/4);if(null==r)return
null;var n=this.doPrivate(r);if(null==n)return null;var s=n.toString(16);return
0==(1&s.length)?s:"0"+s},t.prototype.verify=function(t,e,i){var
r=N(e,16),n=this.doPublic(r);return null==n?null:function(t){for(var e in
X)if(X.hasOwnProperty(e)){var i=X[e],r=i.length;if(t.substr(0,r)==i)return
t.substr(r)}return
t}(n.toString(16).replace(/^1f+00/,""))==i(t).toString()},t}(),X={md2:"3020300c06082a86488
6f70d020205000410",md5:"3020300c06082a864886f70d020505000410",sha1:"3021300906052b0e03021a
05000414",sha224:"302d300d0609608648016503040204050004 1c",sha256:"3031300d0609608648016503
```

04020105000420",sha384:"3041300d0609608648016503040202050004 30",sha512:"3051300d0609608648016503040203050004 40",ripemd160:"3021300906052b2403020105000414"},Q={};Q.lang={extend:function(t,e,i){if(!e||!t)throw new Error("YAHOO.lang.extend failed, please check that all dependencies are included.");var r=function(){};if(r.prototype=e.prototype,t.prototype=new r,t.prototype.constructor=t,t.superclass=e.prototype,e.prototype.constructor==Object.prototype.constructor&&(e.prototype.constructor=e),i){var n;for(n in i)t.prototype[n]=i[n];var s=function(){},o=["toString","valueOf"];try{/MSIE/.test(navigator.userAgent)&&(s=function(t,e){for(n=0;n<o.length;n+=1){var i=o[n],r=e[i];"function"==typeof r&&r!=Object.prototype[i]&&(t[i]=r)}})}catch(t){}s(t.prototype,i)}}};var W={};void 0!==W.asn1&&W.asn1||(W.asn1={}),W.asn1.ASN1Util=new function(){this.integerToByteHex=function(t){var e=t.toString(16);return e.length%2==1&&(e="0"+e),e},this.bigIntToMinTwosComplementsHex=function(t){var e=t.toString(16);if("-"!=e.substr(0,1))e.length%2==1?e="0"+e:e.match(/^[0-7]/)||(e="00"+e);else{var i=e.substr(1).length;i%2==1?i+=1:e.match(/^[0-7]/)||(i+=2);for(var r="",n=0;n<i;n++)r+="f";e=new R(r,16).xor(t).add(R.ONE).toString(16).replace(/^-/,"")}return e},this.getPEMStringFromHex=function(t,e){return hextopem(t,e)},this.newObject=function(t){var e=W.asn1,i=e.DERBoolean,r=e.DERInteger,n=e.DERBitString,s=e.DEROctetString,o=e.DERNull,h=e.DERObjectIdentifier,a=e.DEREnumerated,u=e.DERUTF8String,c=e.DERNumericString,f=e.DERPrintableString,l=e.DERTeletexString,p=e.DERIA5String,g=e.DERUTCTime,d=e.DERGeneralizedTime,v=e.DERSequence,m=e.DERSet,y=e.DERTaggedObject,b=e.ASN1Util.newObject,T=Object.keys(t);if(1!=T.length)throw"key of param shall be only one.";var S=T[0];if(-1==":bool:int:bitstr:octstr:null:oid:enum:utf8str:numstr:prnstr:telstr:ia5str:utctime:gentime:seq:set:tag:".indexOf(":"+S+":"))throw"undefined key: "+S;if("bool"==S)return new i(t[S]);if("int"==S)return new r(t[S]);if("bitstr"==S)return new n(t[S]);if("octstr"==S)return new s(t[S]);if("null"==S)return new o(t[S]);if("oid"==S)return new h(t[S]);if("enum"==S)return new a(t[S]);if("utf8str"==S)return new u(t[S]);if("numstr"==S)return new c(t[S]);if("prnstr"==S)return new f(t[S]);if("telstr"==S)return new l(t[S]);if("ia5str"==S)return new p(t[S]);if("utctime"==S)return new g(t[S]);if("gentime"==S)return new d(t[S]);if("seq"==S){for(var E=t[S],w=[],D=0;D<E.length;D++){var x=b(E[D]);w.push(x)}return new v({array:w})}if("set"==S){for(E=t[S],w=[],D=0;D<E.length;D++)x=b(E[D]),w.push(x);return new m({array:w})}if("tag"==S){var R=t[S];if("[object Array]"===Object.prototype.toString.call(R)&&3==R.length){var B=b(R[2]);return new y({tag:R[0],explicit:R[1],obj:B})}var O={};if(void 0!==R.explicit&&(O.explicit=R.explicit),void 0!==R.tag&&(O.tag=R.tag),void 0===R.obj)throw"obj shall be specified for 'tag'.";return O.obj=b(R.obj),new y(O)}},this.jsonToASN1HEX=function(t){return this.newObject(t).getEncodedHex()}},W.asn1.ASN1Util.oidHexToInt=function(t){for(var e="",i=parseInt(t.substr(0,2),16),r=(e=Math.floor(i/40)+"."+i%40,""),n=2;n<t.length;n+=2){var s=("00000000"+parseInt(t.substr(n,2),16).toString(2)).slice(-8);r+=s.substr(1,7),"0"==s.substr(0,1)&&(e=e+"."+new R(r,2).toString(10),r="")}return e},W.asn1.ASN1Util.oidIntToHex=function(t){var e=function(t){var e=t.toString(16);return 1==e.length&&(e="0"+e),e},i=function(t){var i="",r=new R(t,10).toString(2),n=7-r.length%7;7==n&&(n=0);for(var s="",o=0;o<n;o++)s+="0";for(r=s+r,o=0;o<r.length-1;o+=7){var h=r.substr(o,7);o!=r.length-7&&(h="1"+h),i+=e(parseInt(h,2))}return i};if(!t.match(/^[0-9.]+$/))throw"malformed oid string: "+t;var r="",n=t.split("."),s=40*parseInt(n[0])+parseInt(n[1]);r+=e(s),n.splice(0,2);for(var o=0;o<n.length;o++)r+=i(n[o]);return r},W.asn1.ASN1Object=function(){this.getLengthHexFromValue=function(){if(void 0===this.hV||null==this.hV)throw"this.hV is null or undefined.";if(this.hV.length%2==1)throw"value hex must be even length: n="+"".length+",v="+this.hV;var t=this.hV.length/2,e=t.toString(16);if(e.length%2==1&&(e="0"+e),t<128)return e;var i=e.length/2;if(i>15)throw"ASN.1 length too long to represent by 8x: n = "+t.toString(16);return(128+i).toString(16)+e},this.getEncodedHex=function(){return(null==this.hTLV||this.isModified)&&(this.hV=this.getFreshValueHex(),this.hL=this.getLengthHexFromValue(),this.hTLV=this.hT+this.hL+this.hV,this.isModified=!1),this.hTLV},this.getValueHex

=function(){return
this.getEncodedHex(),this.hV},this.getFreshValueHex=function(){return""}},W.asn1.DERAbstra
ctString=function(t){W.asn1.DERAbstractString.superclass.constructor.call(this),this.getSt
ring=function(){return
this.s},this.setString=function(t){this.hTLV=null,this.isModified=!0,this.s=t,this.hV=stoh
ex(this.s)},this.setStringHex=function(t){this.hTLV=null,this.isModified=!0,this.s=null,th
is.hV=t},this.getFreshValueHex=function(){return this.hV},void 0!==t&&("string"==typeof
t?this.setString(t):void 0!==t.str?this.setString(t.str):void
0!==t.hex&&this.setStringHex(t.hex))},Q.lang.extend(W.asn1.DERAbstractString,W.asn1.ASN1Ob
ject),W.asn1.DERAbstractTime=function(t){W.asn1.DERAbstractTime.superclass.constructor.cal
l(this),this.localDateToUTC=function(t){return
utc=t.getTime()+6e4*t.getTimezoneOffset(),new
Date(utc)},this.formatDate=function(t,e,i){var
r=this.zeroPadding,n=this.localDateToUTC(t),s=String(n.getFullYear());"utc"==e&&(s=s.subst
r(2,2));var
o=s+r(String(n.getMonth()+1),2)+r(String(n.getDate()),2)+r(String(n.getHours()),2)+r(Strin
g(n.getMinutes()),2)+r(String(n.getSeconds()),2);if(!0===i){var
h=n.getMilliseconds();if(0!=h){var
a=r(String(h),3);o=o+"."+(a=a.replace(/[0]+$/,""))}}return
o+"Z"},this.zeroPadding=function(t,e){return t.length>=e?t:new Array(e-
t.length+1).join("0")+t},this.getString=function(){return
this.s},this.setString=function(t){this.hTLV=null,this.isModified=!0,this.s=t,this.hV=stoh
ex(t)},this.setByDateValue=function(t,e,i,r,n,s){var o=new Date(Date.UTC(t,e-
1,i,r,n,s,0));this.setByDate(o)},this.getFreshValueHex=function(){return
this.hV}},Q.lang.extend(W.asn1.DERAbstractTime,W.asn1.ASN1Object),W.asn1.DERAbstractStruct
ured=function(t){W.asn1.DERAbstractString.superclass.constructor.call(this),this.setByASN1
ObjectArray=function(t){this.hTLV=null,this.isModified=!0,this.asn1Array=t},this.appendASN
1Object=function(t){this.hTLV=null,this.isModified=!0,this.asn1Array.push(t)},this.asn1Arr
ay=new Array,void 0!==t&&void
0!==t.array&&(this.asn1Array=t.array)},Q.lang.extend(W.asn1.DERAbstractStructured,W.asn1.A
SN1Object),W.asn1.DERBoolean=function(){W.asn1.DERBoolean.superclass.constructor.call(this
),this.hT="01",this.hTLV="0101ff"},Q.lang.extend(W.asn1.DERBoolean,W.asn1.ASN1Object),W.as
n1.DERInteger=function(t){W.asn1.DERInteger.superclass.constructor.call(this),this.hT="02"
,this.setByBigInteger=function(t){this.hTLV=null,this.isModified=!0,this.hV=W.asn1.ASN1Uti
l.bigIntToMinTwosComplementsHex(t)},this.setByInteger=function(t){var e=new
R(String(t),10);this.setByBigInteger(e)},this.setValueHex=function(t){this.hV=t},this.getF
reshValueHex=function(){return this.hV},void 0!==t&&(void
0!==t.bigint?this.setByBigInteger(t.bigint):void
0!==t.int?this.setByInteger(t.int):"number"==typeof t?this.setByInteger(t):void
0!==t.hex&&this.setValueHex(t.hex))},Q.lang.extend(W.asn1.DERInteger,W.asn1.ASN1Object),W.
asn1.DERBitString=function(t){if(void 0!==t&&void 0!==t.obj){var
e=W.asn1.ASN1Util.newObject(t.obj);t.hex="00"+e.getEncodedHex()}W.asn1.DERBitString.superc
lass.constructor.call(this),this.hT="03",this.setHexValueIncludingUnusedBits=function(t){t
his.hTLV=null,this.isModified=!0,this.hV=t},this.setUnusedBitsAndHexValue=function(t,e){if
(t<0||7<t)throw"unused bits shall be from 0 to 7: u = "+t;var
i="0"+t;this.hTLV=null,this.isModified=!0,this.hV=i+e},this.setByBinaryString=function(t){
var e=8-(t=t.replace(/0+$/,"")).length%8;8==e&&(e=0);for(var i=0;i<=e;i++)t+="0";var
r="";for(i=0;i<t.length-1;i+=8){var
n=t.substr(i,8),s=parseInt(n,2).toString(16);1==s.length&&(s="0"+s),r+=s}this.hTLV=null,th
is.isModified=!0,this.hV="0"+e+r},this.setByBooleanArray=function(t){for(var
e="",i=0;i<t.length;i++)1==t[i]?e+="1":e+="0";this.setByBinaryString(e)},this.newFalseArra
y=function(t){for(var e=new Array(t),i=0;i<t;i++)e[i]=!1;return
e},this.getFreshValueHex=function(){return this.hV},void 0!==t&&("string"==typeof
t&&t.toLowerCase().match(/^[0-9a-f]+$/)?this.setHexValueIncludingUnusedBits(t):void
0!==t.hex?this.setHexValueIncludingUnusedBits(t.hex):void
0!==t.bin?this.setByBinaryString(t.bin):void
0!==t.array&&this.setByBooleanArray(t.array))},Q.lang.extend(W.asn1.DERBitString,W.asn1.AS
N1Object),W.asn1.DEROctetString=function(t){if(void 0!==t&&void 0!==t.obj){var
e=W.asn1.ASN1Util.newObject(t.obj);t.hex=e.getEncodedHex()}W.asn1.DEROctetString.superclas
s.constructor.call(this,t),this.hT="04"},Q.lang.extend(W.asn1.DEROctetString,W.asn1.DERAbs

tractString),W.asn1.DERNull=function(){W.asn1.DERNull.superclass.constructor.call(this),th
is.hT="05",this.hTLV="0500"},Q.lang.extend(W.asn1.DERNull,W.asn1.ASN1Object),W.asn1.DERObj
ectIdentifier=function(t){var e=function(t){var e=t.toString(16);return
1==e.length&&(e="0"+e),e},i=function(t){var i="",r=new R(t,10).toString(2),n=7-
r.length%7;7==n&&(n=0);for(var s="",o=0;o<n;o++)s+="0";for(r=s+r,o=0;o<r.length-
1;o+=7){var h=r.substr(o,7);o!=r.length-7&&(h="1"+h),i+=e(parseInt(h,2))}return
i};W.asn1.DERObjectIdentifier.superclass.constructor.call(this),this.hT="06",this.setValue
Hex=function(t){this.hTLV=null,this.isModified=!0,this.s=null,this.hV=t},this.setValueOidS
tring=function(t){if(!t.match(/^[0-9.]+$/))throw"malformed oid string: "+t;var
r="",n=t.split("."),s=40*parseInt(n[0])+parseInt(n[1]);r+=e(s),n.splice(0,2);for(var
o=0;o<n.length;o++)r+=i(n[o]);this.hTLV=null,this.isModified=!0,this.s=null,this.hV=r},thi
s.setValueName=function(t){var
e=W.asn1.x509.OID.name2oid(t);if(""===e)throw"DERObjectIdentifier oidName undefined:
"+t;this.setValueOidString(e)},this.getFreshValueHex=function(){return this.hV},void
0!==t&&("string"==typeof t?t.match(/^[0-2].[0-
9.]+$/)?this.setValueOidString(t):this.setValueName(t):void
0!==t.oid?this.setValueOidString(t.oid):void 0!==t.hex?this.setValueHex(t.hex):void
0!==t.name&&this.setValueName(t.name))},Q.lang.extend(W.asn1.DERObjectIdentifier,W.asn1.AS
N1Object),W.asn1.DEREnumerated=function(t){W.asn1.DEREnumerated.superclass.constructor.cal
l(this),this.hT="0a",this.setByBigInteger=function(t){this.hTLV=null,this.isModified=!0,th
is.hV=W.asn1.ASN1Util.bigIntToMinTwosComplementsHex(t)},this.setByInteger=function(t){var
e=new
R(String(t),10);this.setByBigInteger(e)},this.setValueHex=function(t){this.hV=t},this.getF
reshValueHex=function(){return this.hV},void 0!==t&&(void
0!==t.int?this.setByInteger(t.int):"number"==typeof t?this.setByInteger(t):void
0!==t.hex&&this.setValueHex(t.hex))},Q.lang.extend(W.asn1.DEREnumerated,W.asn1.ASN1Object)
,W.asn1.DERUTF8String=function(t){W.asn1.DERUTF8String.superclass.constructor.call(this,t)
,this.hT="0c"},Q.lang.extend(W.asn1.DERUTF8String,W.asn1.DERAbstractString),W.asn1.DERNume
ricString=function(t){W.asn1.DERNumericString.superclass.constructor.call(this,t),this.hT=
"12"},Q.lang.extend(W.asn1.DERNumericString,W.asn1.DERAbstractString),W.asn1.DERPrintableS
tring=function(t){W.asn1.DERPrintableString.superclass.constructor.call(this,t),this.hT="1
3"},Q.lang.extend(W.asn1.DERPrintableString,W.asn1.DERAbstractString),W.asn1.DERTeletexStr
ing=function(t){W.asn1.DERTeletexString.superclass.constructor.call(this,t),this.hT="14"},
Q.lang.extend(W.asn1.DERTeletexString,W.asn1.DERAbstractString),W.asn1.DERIA5String=functi
on(t){W.asn1.DERIA5String.superclass.constructor.call(this,t),this.hT="16"},Q.lang.extend(
W.asn1.DERIA5String,W.asn1.DERAbstractString),W.asn1.DERUTCTime=function(t){W.asn1.DERUTCT
ime.superclass.constructor.call(this,t),this.hT="17",this.setByDate=function(t){this.hTLV=
null,this.isModified=!0,this.date=t,this.s=this.formatDate(this.date,"utc"),this.hV=stohex
(this.s)},this.getFreshValueHex=function(){return void 0===this.date&&void
0===this.s&&(this.date=new
Date,this.s=this.formatDate(this.date,"utc"),this.hV=stohex(this.s)),this.hV},void
0!==t&&(void 0!==t.str?this.setString(t.str):"string"==typeof t&&t.match(/^[0-
9]{12}Z$/)?this.setString(t):void 0!==t.hex?this.setStringHex(t.hex):void
0!==t.date&&this.setByDate(t.date))},Q.lang.extend(W.asn1.DERUTCTime,W.asn1.DERAbstractTim
e),W.asn1.DERGeneralizedTime=function(t){W.asn1.DERGeneralizedTime.superclass.constructor.
call(this,t),this.hT="18",this.withMillis=!1,this.setByDate=function(t){this.hTLV=null,thi
s.isModified=!0,this.date=t,this.s=this.formatDate(this.date,"gen",this.withMillis),this.h
V=stohex(this.s)},this.getFreshValueHex=function(){return void 0===this.date&&void
0===this.s&&(this.date=new
Date,this.s=this.formatDate(this.date,"gen",this.withMillis),this.hV=stohex(this.s)),this.
hV},void 0!==t&&(void 0!==t.str?this.setString(t.str):"string"==typeof t&&t.match(/^[0-
9]{14}Z$/)?this.setString(t):void 0!==t.hex?this.setStringHex(t.hex):void
0!==t.date&&this.setByDate(t.date),!0===t.millis&&(this.withMillis=!0))},Q.lang.extend(W.a
sn1.DERGeneralizedTime,W.asn1.DERAbstractTime),W.asn1.DERSequence=function(t){W.asn1.DERSe
quence.superclass.constructor.call(this,t),this.hT="30",this.getFreshValueHex=function(){f
or(var t="",e=0;e<this.asn1Array.length;e++)t+=this.asn1Array[e].getEncodedHex();return
this.hV=t,this.hV}},Q.lang.extend(W.asn1.DERSequence,W.asn1.DERAbstractStructured),W.asn1.
DERSet=function(t){W.asn1.DERSet.superclass.constructor.call(this,t),this.hT="31",this.sor
tFlag=!0,this.getFreshValueHex=function(){for(var t=new
Array,e=0;e<this.asn1Array.length;e++){var

```
i=this.asn1Array[e];t.push(i.getEncodedHex())}return
1==this.sortFlag&&t.sort(),this.hV=t.join(""),this.hV},void 0!==t&&void
0!==t.sortflag&&0==t.sortflag&&(this.sortFlag=!1)},Q.lang.extend(W.asn1.DERSet,W.asn1.DERA
bstractStructured),W.asn1.DERTaggedObject=function(t){W.asn1.DERTaggedObject.superclass.co
nstructor.call(this),this.hT="a0",this.hV="",this.isExplicit=!0,this.asn1Object=null,this.
setASN1Object=function(t,e,i){this.hT=e,this.isExplicit=t,this.asn1Object=i,this.isExplici
t?(this.hV=this.asn1Object.getEncodedHex(),this.hTLV=null,this.isModified=!0):(this.hV=nul
l,this.hTLV=i.getEncodedHex(),this.hTLV=this.hTLV.replace(/^../,e),this.isModified=!1)},th
is.getFreshValueHex=function(){return this.hV},void 0!==t&&(void
0!==t.tag&&(this.hT=t.tag),void 0!==t.explicit&&(this.isExplicit=t.explicit),void
0!==t.obj&&(this.asn1Object=t.obj,this.setASN1Object(this.isExplicit,this.hT,this.asn1Obje
ct)))},Q.lang.extend(W.asn1.DERTaggedObject,W.asn1.ASN1Object);var
tt,et,it=(tt=function(t,e){return tt=Object.setPrototypeOf||{__proto__:[]}instanceof
Array&&function(t,e){t.__proto__=e}||function(t,e){for(var i in
e)Object.prototype.hasOwnProperty.call(e,i)&&(t[i]=e[i])},tt(t,e)},function(t,e){if("funct
ion"!=typeof e&&null!==e)throw new TypeError("Class extends value "+String(e)+" is not a
constructor or null");function
i(){this.constructor=t}tt(t,e),t.prototype=null===e?Object.create(e):(i.prototype=e.protot
ype,new i)}),rt=function(t){function e(i){var r=t.call(this)||this;return
i&&("string"==typeof
i?r.parseKey(i):(e.hasPrivateKeyProperty(i)||e.hasPublicKeyProperty(i))&&r.parseProperties
From(i)),r}return it(e,t),e.prototype.parseKey=function(t){try{var e=0,i=0,r=/^\s*(?:[0-
9A-Fa-f][0-9A-Fa-f]\s*)+$/.test(t)?function(t){var e;if(void 0===u){var
i="0123456789ABCDEF",r="
\f\n\r\t \u2028\u2029";for(u={},e=0;e<16;++e)u[i.charAt(e)]=e;for(i=i.toLowerCase(),e=10;e
<16;++e)u[i.charAt(e)]=e;for(e=0;e<r.length;++e)u[r.charAt(e)]=-1}var
n=[],s=0,o=0;for(e=0;e<t.length;++e){var h=t.charAt(e);if("="==h)break;if(-
1!=(h=u[h])){if(void 0===h)throw new Error("Illegal character at offset
"+e);s|=h,++o>=2?(n[n.length]=s,s=0,o=0):s<<=4}}if(o)throw new Error("Hex encoding
incomplete: 4 bits missing");return
n}(t):g.unarmor(t),n=E.decode(r);if(3===n.sub.length&&(n=n.sub[2].sub[0]),9===n.sub.length
){e=n.sub[1].getHexStringValue(),this.n=N(e,16),i=n.sub[2].getHexStringValue(),this.e=pars
eInt(i,16);var s=n.sub[3].getHexStringValue();this.d=N(s,16);var
o=n.sub[4].getHexStringValue();this.p=N(o,16);var
h=n.sub[5].getHexStringValue();this.q=N(h,16);var
a=n.sub[6].getHexStringValue();this.dmp1=N(a,16);var
c=n.sub[7].getHexStringValue();this.dmq1=N(c,16);var
f=n.sub[8].getHexStringValue();this.coeff=N(f,16)}else{if(2!==n.sub.length)return!1;if(n.s
ub[0].sub){var
l=n.sub[1].sub[0];e=l.sub[0].getHexStringValue(),this.n=N(e,16),i=l.sub[1].getHexStringVal
ue(),this.e=parseInt(i,16)}else
e=n.sub[0].getHexStringValue(),this.n=N(e,16),i=n.sub[1].getHexStringValue(),this.e=parseI
nt(i,16)}return!0}catch(t){return!1}},e.prototype.getPrivateBaseKey=function(){var
t={array:[new W.asn1.DERInteger({int:0}),new W.asn1.DERInteger({bigint:this.n}),new
W.asn1.DERInteger({int:this.e}),new W.asn1.DERInteger({bigint:this.d}),new
W.asn1.DERInteger({bigint:this.p}),new W.asn1.DERInteger({bigint:this.q}),new
W.asn1.DERInteger({bigint:this.dmp1}),new W.asn1.DERInteger({bigint:this.dmq1}),new
W.asn1.DERInteger({bigint:this.coeff})]};return new
W.asn1.DERSequence(t).getEncodedHex()},e.prototype.getPrivateBaseKeyB64=function(){return
f(this.getPrivateBaseKey())},e.prototype.getPublicBaseKey=function(){var t=new
W.asn1.DERSequence({array:[new
W.asn1.DERObjectIdentifier({oid:"1.2.840.113549.1.1.1"}),new W.asn1.DERNull]}),e=new
W.asn1.DERSequence({array:[new W.asn1.DERInteger({bigint:this.n}),new
W.asn1.DERInteger({int:this.e})]}),i=new
W.asn1.DERBitString({hex:"00"+e.getEncodedHex()});return new
W.asn1.DERSequence({array:[t,i]}).getEncodedHex()},e.prototype.getPublicBaseKeyB64=functio
n(){return f(this.getPublicBaseKey())},e.wordwrap=function(t,e){if(!t)return t;var
i="(.{1,"+(e=e||64)+"})( +|$\n?)|(.{1,"+e+"})";return
t.match(RegExp(i,"g")).join("\n")},e.prototype.getPrivateKey=function(){var t="-----BEGIN
RSA PRIVATE KEY-----\n";return(t+=e.wordwrap(this.getPrivateBaseKeyB64())+"\n")+"-----END
```

```
RSA PRIVATE KEY-----"},e.prototype.getPublicKey=function(){var t="-----BEGIN PUBLIC KEY---
---\n";return(t+=e.wordwrap(this.getPublicBaseKeyB64())+"\n")+"-----END PUBLIC KEY-----
"},e.hasPublicKeyProperty=function(t){return(t=t||{}).hasOwnProperty("n")&&t.hasOwnPropert
y("e")},e.hasPrivateKeyProperty=function(t){return(t=t||{}).hasOwnProperty("n")&&t.hasOwnP
roperty("e")&&t.hasOwnProperty("d")&&t.hasOwnProperty("p")&&t.hasOwnProperty("q")&&t.hasOw
nProperty("dmp1")&&t.hasOwnProperty("dmq1")&&t.hasOwnProperty("coeff")},e.prototype.parseP
ropertiesFrom=function(t){this.n=t.n,this.e=t.e,t.hasOwnProperty("d")&&(this.d=t.d,this.p=
t.p,this.q=t.q,this.dmp1=t.dmp1,this.dmq1=t.dmq1,this.coeff=t.coeff)},e}(J),nt=i(155),st=v
oid 0!==nt?null===(et=nt.env)||void 0===et?void 0:"3.3.1":void 0;const
ot=function(){function t(t){void
0===t&&(t={}),t=t||{},this.default_key_size=t.default_key_size?parseInt(t.default_key_size
,10):1024,this.default_public_exponent=t.default_public_exponent||"010001",this.log=t.log|
|!1,this.key=null}return
t.prototype.setKey=function(t){this.log&&this.key&&console.warn("A key was already set,
overriding existing."),this.key=new
rt(t)},t.prototype.setPrivateKey=function(t){this.setKey(t)},t.prototype.setPublicKey=func
tion(t){this.setKey(t)},t.prototype.decrypt=function(t){try{return
this.getKey().decrypt(l(t))}catch(t){return!1}},t.prototype.encrypt=function(t){try{return
f(this.getKey().encrypt(t))}catch(t){return!1}},t.prototype.sign=function(t,e,i){try{retur
n
f(this.getKey().sign(t,e,i))}catch(t){return!1}},t.prototype.verify=function(t,e,i){try{re
turn
this.getKey().verify(t,l(e),i)}catch(t){return!1}},t.prototype.getKey=function(t){if(!this
.key){if(this.key=new rt,t&&"[object Function]"==={}.toString.call(t))return void
this.key.generateAsync(this.default_key_size,this.default_public_exponent,t);this.key.gene
rate(this.default_key_size,this.default_public_exponent)}return
this.key},t.prototype.getPrivateKey=function(){return
this.getKey().getPrivateKey()},t.prototype.getPrivateKeyB64=function(){return
this.getKey().getPrivateBaseKeyB64()},t.prototype.getPublicKey=function(){return
this.getKey().getPublicKey()},t.prototype.getPublicKeyB64=function(){return
this.getKey().getPublicBaseKeyB64()},t.version=st,t}()})(),r.default})()));
```