Міністерство освіти і науки України
ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»
Кафедра комп'ютерної інженерії та електроніки
(повна назва кафедри)

Тарлєв Сергій Віталійович
Serhii Tarlev

УДК 004:681.5

Спеціальність 123 «комп'ютерна інженерія»
(шифр та назва спеціальності)

Кваліфікаційна робота
на здобуття освітньо-кваліфікаційного рівня бакалавр
(бакалавр, спеціаліст, магістр)

Мікроконтролерна система передачі криптографічних даних
Microcontroller system for cryptographic data transmission

Науковий керівник:
кандидат технічних наук,
доцент Голота В.І.

Рецензент:
Доктор фіз.-мат. наук, професор
кафедри фізики і хімії твердого
тіла
Салій Я.П.

Івано-Франківськ
2021

Формат	Зона	Поз.	Позначення	Найменування	К-ть	Прим.
			123.КІ-41.05	Пояснювальна записка	1	

					123.КІ-41.05
Змн.	Арк.	№ докум.	Підпис	Дата	
Розробив		Тарлєв С. В.			Специфікація
Перевірив		Голота В.І.			
Н. Контр.					
Затвердив					

Літ.	Арк.	Аркушів
	2	103

АНОТАЦІЯ

В бакалаверській роботі розроблено універсальний псевдокод для алгоритму шифрування AES. Даний алгоритм відноситься до симметричних алгоритмів блочного шифрования. Для перевірки роботи даного алгоритму було розроблено два варіанти на Java і Python.

Для демонстрації використано мікроконтролер ESP32 до якого можна підключитися по http який верне зашифроване повідомлення. Яке можна буде в подальшому дешифрувати.

Розроблено блок-схему роботи алгоритму.

Обґрунтувано економічну доцільність виготовлення даної системи.

					123.КІ-41.05			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розробив		Тарлєв С. В.			Мікроконтролерна система передачі криптографічних даних	Літ.	Арк.	Аркушів
Перевірив		Голота В.І.					3	103
Н. Контр.								
Затвердив								

SUMMARY

In the bachelor's thesis a universal pseudocode for the AES encryption algorithm was developed. This algorithm refers to symmetric block encryption algorithms. To test the operation of this algorithm, two options were developed in Java and Python.

For demonstration I used the ESP32 microcontroller to which it is possible to be connected on http which will return the encrypted message. Which can be further deciphered.

Developed a block diagram of the algorithm.

He substantiated the economic feasibility of manufacturing this system.

					123.КІ-41.05			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розробив		Тарлєв С. В.			Мікроконтролерна система передачі криптографічних даних	Літ.	Арк.	Аркушів
Перевірив		Голота В.І.					4	103
Н. Контр.								
Затвердив								

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра «Комп'ютерної інженерії та електроніки»

Пояснювальна записка

до кваліфікаційної роботи на тему:

Мікроконтролерна система передачі криптографічних даних

					123.КІ-41.05			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розробив		Тарлєв С. В.			Пояснювальна записка	Літ.	Арк.	Аркушів
Перевірив		Голота В.І.					5	103
Н. Контр.								
Затвердив								

Зміст

ВСТУП

В усі часи широко цінувалася інформація, оскільки вона завжди була найціннішим ресурсом. Завдяки ній можна багато чого отримати, змінити. Тому інформацію про місце знаходження рудників з цінними ресурсами, розташування ворожих військ, план атаки, внутрішня ситуація в країні і так далі завжди була таємною і всі її хотіли отримати.

Але від інформації не буде користі, якщо нею не обмінюватися. Саме це є саме слабким місцем в секретності інформації. Одним із перших це питання вирішив Юрій Цезар, почавши використовувати шифр підстановки який з часом названий в його честь для приватного листування.

На даний момент безпека інформації стосується не тільки державної, військової справи але і цивільної, оскільки ми живем в світі який живе в інтернеті, і вся інформація є широко доступною. В першу чергу це стосується особистої інформації, банківських рахунків фізичних і юридичних осіб і так далі. І доступ до частини інформації повинен бути обмежений, оскільки знання її третями особами може принести шкоду її власникові. І щоб забезпечити дану безпеку інформації, ми її шифруємо, а отримати доступ до неї зможе лише той хто має права доступу до неї.

Сьогодні існує доволі багато алгоритмів шифрування, предметом мого дослідження були різні алгоритми шифрування, різної категорії. І серед усіх них предметом мого дослідження став алгоритм AES, який на даний момент є одним із найбезпечніших алгоритмів шифрування і він використовується як основний для роботи спецслужб США. Метою дослідження було виведення універсального коду який можна використати для більшості мов як програмування так і скріптових.

1. ОСНОВНІ ПОНЯТТЯ І ЗАСТОСУВАННЯ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

1.1. Термінологія

Шифр - це об’єднання декількох алгоритмів шифрування, що в свою чергу дають криптографічні перетворення, що повертають множину зашифрованих даних на множину відкритих даних і навпаки.

Важливим елементом для будь якого шифру є ключ. Ключ - параметр криптографічного алгоритму, за допомогою якого можна задати вибір переторення з декількох перетворень відкритих даних у зашифровані дані. Сьогодні секретність алгоритму шифрування базується на ключі, а не як сам алгоритм шифрування.

Шифрування - це перетворення даних, що відбувається завдяки певному алгортму, яке виконується у строгій посимвольній послідовності, для отримання шифрованого тексту. Тобто перетворення інформації, з подальшим приховванням даних.

Розшифрування - це процес санкціонованого перетворення зашифрованих даних у придатний для читання. Вивчається криптографією.

Дешифрування - це процес несанкціонованого перетворення інформації з зашифрованиз даних. При умові що ключ який потрібен для розшифрування є невідомим. Вивчається криптоаналізом.

Шифротекст - це інформація яка була зашифрована в результаті використання шифру.

Біграмічний шифр - це шифр для роботи якого використовується групу з двох букв.

За типом ключа, що використовує алгоритми шифрування поділяються, на:

1. Шифрування з симетричним ключем - це алгоритми шифрування у яких використовуються один ключ для шифрування і дешифрування інформації
2. Шифрування з асиметричними ключами - це алгоритми шифрування у яких використовуються два ключі, один ключ для шифрування, другий для розшифрування.

Відкритий ключ - це ключ який використовується для шифрування даних у алгоритмах з асиметричним ключем. Даний ключ не обов’язково тримати в

таємниці і він може бути публічним.

Закритий ключ - це ключ який використовується для дешифрування даних у алгоритмах з асиметричним ключем. Даний ключ обов'язково тримати в таємниці і він повинен бути приватиним.

За типом вхідних даних позділяють на:

1. Блочне шифрування - це різновид симетричного шифрування при одній ітерації обробляється декілька байтів інформації.
2. Потокове шифрування - це різновид симетричного шифрування при одній ітерації обробляються один байт інформації незалежно від інших.

1.2. Криптографія

Доволі довгий час вважалося що криптографія це процес перетворення інформації в незрозумілий набір символів, тобто процес шифрування інформації. Але вона також має зворотній процес коли з незрозумілої послідовності символів отримаєм початкову інформацію під назвою розшифрування. Щоб утворити шифр потрібна пара алгоритмів для шифрування і розшифрування. В основу всіх шифрів положено алгоритм який виконує всю процедуру і секретний параметр, тобто ключ. Ключ є однією з найважливіших елементів шифру, оскільки при статичному ключі, шифр втрачає всю свою силу і тоді доволі легко отримати інформацію.

Тривалий час основною задачою криптографії була надати повідомленню конфеденційності. Тобто зашифрувати і розшифрувати повідомлення. Але протягом останього часу до задач криптографії почали входити: методи перевірки цілісності повідомлення, цифровий підпис, технології безпечного спілкування, інтерактивні пітверження, індефікування одержувача і відправника.

В часи коли криптографія тільки почала зароджуватися, доволі мала кількість людей могла читати і писати і потреба в складних шифрів не була. Тому більшість шифрів базувалися на методі перестановки символів або підстановичні шифри. Але такий захист є дуже слабким. Одим із таких шифрів і найбільш відомий був саме шифр Цезаря, який був названий в честь Юлія Цезаря який його найбільше використовував для спілкування з генералами під час військових дій. Суть алгоритму була в суві кожної літери відно алфавіту на 3 позиції. Якщо проводи

його з сучасними алгоритмами він є подібним на EXCESS-3 в булівській алгебрі.

Основна ціль шифрування є збереження спілкування в таємниці. Даний метод буде найбільш важливий для дипломатів, шпигунів, військових лідерів як це було наведе у прикладі з алгоритмом Цезаря. Також не менш важливим для даної цілі буде стенографія, тобто приховати факт того що повідомлення існує. Сучаними інтрументами стенографії є: цифрові водяні знаки, невидемі чорнила, мікрокрапки.

Більшість класичних шифрів видають статичну інформацію, це в свою чергу можна використати для взламу алгоритму шифрування і розшифрування. Це стало можливим завдяки частотного аналізу, що був відкритий в 9 столітті арабським вченим Аль-Кінді. Тому такі шифри зараз збереглися у вигляді головоломок. Але в 1467 був винайдений поліалфавітний шифр Альбертом Леоном-Батіста, в результаті чого шифрування знову отримало попит оскільки він мав захист від частотного аналізу, а всі алгоритми до цього ні. Але існує теорія що арабські вчені володіли ним до цього. Ідея шифру була доволі просто, щоб збільшити безпеку просто потрібно різні часнити повідомлення шифрувати різними шифрами.

У шифрі Відеженера що є поліалфавітним, для його роботи потрібно ключове слово, завдяки якому алгортм визначає яку літеру потрібно підставляти в залежності того яка літера використовується з ключового слова.

Але в середині 1800-х було доведено що дані алгоритми є все ж таки частично беззахисні перед частотним аналізом. Хоча більшість кроптоаналітиків не знали про цей метод, яким би не була дана техніка ефективна. Потреба в дешифрувані була і єдиним способ для цього протрібно знати шифр. Щоб отримати шифр, використовували крадіжки, шпигунство. З часом криптографи побачили що збереження шифру в таємниці є не доцільним, якщо сам алгоритм є доволі потужним. Але при умові що шифр використовує ключ, і збереження його в таємниці буде достатньо. Це твердження сформував Огюстом Керкгофсом у 1883 році.

Доволі багато було створенно інструментів та приладів для допомоги в шифруванні. Одним і з них була скітала, яка була створена в стародавній Греції. Це була палиця що слугувала як перестановочний шифр який використовувався спратанцями. В середньовіччя було винайдено дірковий шифр. Такі прилади також

були доволі популярні в 20 столітті, найвідомішим з них були Енігма. Він використовувався до кінця Другої Світової війни Німеччиною.

З часом появилася електроніка і цифрові комп'ютери, що в свою чергу дозволило ускладнити шифри. Нові алгоритми вже дозволяли працювати з бітами інформації, а не цілими літерами, як це було раніше. Але комп'ютери також дозволили підняти рівень криптоаналізу. Але сучасні алгоритми шифрування все ж таки випереджають криптоаналіз, що в свою чергу дає можливість безпечно передавати інформацію.

В 1970-х роках почалося широке академічне дослідження криптографії, що в свою чергу породило відкритий стандарт DES, публікація алгоритму Діффі-Хелмана та оприлюднення алгоритму RSA. Надійність безпеці таких алгоритмів надають складні матиматичні обчислення такі як проблеми з дискретними логарифмами та розклад цілих чисел.

Завдяки підвищенню обчислювальної потужності сучасних комп'ютерів дає можливість провести атаку грубої сили, тобто перебрати всі можливі ключі. Саме по цій причині постійно збільшують мінімальну довжину ключа необхідну для шифрування.

1.2.1 Класифікація криптографічних атак

Криптографічні атаки здійснюються завдяки криптоаналізу, на основі якого можна отримати певній рівень доступу до зашифрованої інформації, що піддається атаці. Спочатку користувач використовує ключ і шифр завдяки яким перетворює текст у шифротесті. Наступним кроком це передача по не захищеному каналі зашифрованого тесту. І в подальному інша сторона розшифровує інформацію завдяки ключу який у нього є для розшифрування. Задача криптоаналітика дешифрувати інформацію і спробувати вивести ключ на основі якого можна буде розшифрувати всю подальшу інформацію. Сьогодні усі алгоритми є відкритими і публічними. Це припущення має назву принципом Керкгоффза.

Ось приклади одних із самих відомих моделів атак:

Атака на основі шифротексту (COA) - дана атака базується, що зловмисник має шифротекст і не має відкритого тексту. Ця модель атаки є найбільш

популярною, але вона також є найбільш слабкою порівняно з іншими моделями злому. Але перед сучаними алгоритмами шифрування стоїть вимогати мати якомога сильніший захист перед даною атакою. Але також порібно зазначити щоб провести подібну атаку потрібно також зловмиснику знати на якій мові написаний відкритий тест, також мати певну інформацію про те, що знаходится в інформації до якої він хоче отримати доступ.

Атака грубої сили - це модель атаки, базується повному переборі усіх можливих ключів, тобто якщо довжина ключа становить N бітів то кількість переборів становитеме 2^N раз, середній час для даного способу становить $2^{(N-1)}$. Дану атаку використовують для порівняння з іншими атаками, з метою дослідити їхню ефективність. Перед даною атакою не устоїть жоден алгоритм шифрування. Але у нього є головний недолік. Потрібно мати достатью інформації про відкритий тест, щоб при переборі мати можливість порівняти чи правильний ключ був підібраний. Рекомендується мініму N бітів інформації.

Атака з відкритим текстом (KPA) - для даної атаки зловмиснику потрібно мати фрагмент з інформації, яка є зашифрована і сама шифрограма. Саме даний спосіб був використаний в успішній операції Enigma.

Атака на основі підібраного відкритого тексту (CPA) - для даної атаки криптоаналітику потрібно вибрати декілька текстів, які будуть шифруватися. В результаті він отримає можливітсь дослідити всі можливі стани. В результаті чого виведе закономірності даного алгоритму. У доволі популярних асиметричних алгоритмах шифрування використовується публічний ключ на основі якого зломисник зможе створити потрібні зашифровані тексти. Тому алгоритм публічного ключа повинен бути стійким до подібного роду атак.

Атака на основі підібраного шифротексту (CCA) - для даної атаки аналітику потрібно мати довільний зашифрований текст і отримати доступ до розшифрованого тексту. Щоб це отримати потрібно мати доступ до каналу зв'язку і повідомлень ожержувача.

Атака з моделю відкритого ключа - для даної атаки зловмиснику потрібно мати певні знання про ключ шифру.

1.2.2. Види розкриття алгоритмів шифрування

Усі алгоритми ділятися на симетричні і асиметричні.

Симетричний алгоритм як видно з назви використовує один ключ як для шифрування так і для дешифрування. Переваги даного алгоритму в швидкодії, потреба в менших обчислювальних потужностях. Найпоширенішими прикладими серед симетричних алгоритмів: AES, RC4, DES, 3DES, RC5, RC6.

DES (Data Encryption Standard) - стандарт що був прийнятий урядом США з 1976 року. Через деякий час став популярний у всьому світі. Але отримав неоднозначні відгуки протягом деякого часу. Оскільки перед тим як прийняти даний шифр уряд відправив фото на аналіз до Національного Агентства Безпеки США, який його трохи змінив. Таблиці які генерувалися стали статичними, і люди почали побоюватися що завдяки даним змінним вони спростили собі роботу для можливості розшифрування інформації яка була защифрована даним шифром. Також критика була в сторону того що для даного шифру був потрібен доволі малий за розміром ключ. Але даний шифр дав основу для блочних алгоритмів та криптоаналізу.

На даний момент DES уже не вважається найдійним і уряд США віднього відмовився ще у 2002 році, замінивши його більш сучасним алгоритмом AES. Причиною послужило те шо 1999 році ключ був дишефрований публічно протягом 22 голин і 15 хвилин. Зараз же даний алгоритм трохи вдосконалили до версії під назвою 3-DES, але і для нього вже є теоретичні стратегії для взому.

Робота над даним алгоритмом почалася після оголошення конкурсу НБС над розробкою нового алгоритму шифрування. Спочатку організували конкурс 15 травня 1973 року, але він він задавав строгі архітектурні вимоги, в результаті чого була відсутність заявок. Саме тому, організували інший конкурс 27 серпня 1974 року. Цього разу IBM подала заявку яка задовільнала всім вимогам. Вони його розробляли протягом 1973-1974 року і в його основу було закладений шифр Люцифер, розроблений Хостом Фейля.

Алгоритму DES відносять до алгоритмів блочного шифрування, де всі елементи розбиавються на 64 бітні блоки. Ці дані подаються на вхід і отримаються на виході роботи алгоитму. Щоб виконати операції шифрування і дешифрування ми

використовуємо один і той же самий алгорим шифруваннння.

Для роботи алгоритму потрібен ключ розміром 56 бітів. Хоча з джерела ми беремо 64 біти і 8 з них використовуємо для паритету. При підборі ключа можемо використувати довільну комбінацію, але не потрібно забувати що потужність алгоритму базується саме на його ключі, тому деякі комбінації можуть бути слабкими і їх краще не використовувати.

В самі основі алгоритму відбуваються дві основні дії підстановка і переміщування. І для роботи DES виконується поєднання цих двої технік.

Алгоритм симетричного 3DES

З 1990 року почати використовувати алгоритм 3DES замість DES і в його основу ліг саме алгоритм DES, як видно з назви якого запускають 3 рази для кожного блоку даних. В результатіь чого зашифровані дані буде взламати набагато важче. Його активно почали використовувати в фінансовій сфері а саме в платіжних системах. Саме він ліг в основу таких протоколів, як TLS, SSH, IPsec, OpenVPN.

Але з часом його взомали. Була знайдена вразливість Sweet32 що була знайдена Гаетаном Леурентом і Картікеяном Бхаварганом. В свою чергу його визнали устарілим і у 2019 році Національний інститут стандартів і технологій США оголосив про це офіційно.

Усі технології які використовують даний алгоритм шифрування повинні перестати його використовувати о 2023 року. Також варто наголосити що його вже перестали використовувати у SSL/TLS.

1.3. Законодавство України в галузі криптографії

На даний момент законовство в Україні знаходиться на стадії формування. Що в свою чергу є сукуністю законів, правил і нормативних актів. Вони регламентують загальну організацю робіт і створення систем захисту.

1.3.1. Закон "Про інформацію”

Правова система регламентує розмежування і відношення повноваження усіх учасників інформаційних відносин. Також основні державні органи, що забезпечують державну безпеку і контроль за надання доступу до інформації.

Правові норми і законодавчі акти охоплюють усі проблеми пов'язані з безпекою інформації, які відобрають усю специфіку пов'язану з забезпеченням безпеки інформації як для суспільства так і для держави.

Однією з важливих сторін забезпечення безпеки це розробка стандартів і серфітикатів, що будуть гарантами дотримання правил установлиними законодавством держави, у даному випадку це України. Законодавство сформоване для інформаційних систем, засобів телекомунікації і автоматизованих систем.

Законом України "Про інформацію" встановлено загальні правові основи отримання, поширення, збереження і використання інформації. З нього випливає, що інформацією вважається документи або відомомсті, що привселюдно оголошені про явища і події, які відбуваються у суспільстві, природному середовищі чи державі.

Законом також описані такі моменти як право особистості на інформацію у державі і суспільному житті, також джерела, статус учасників, регулює доступ до інформації, забезпечують її охорону і захищаючи суспільство від дезінформаії. Суб'єктами інформаційних відносин є громадяни України, юридичні особи, держави такі як Україна і інші, громадяни інших країн і ті хто не має громадянства, міжнародні організації. Для врегулювання потреби в інформації були створенні інформаційні служби, мережі, системи, бази і банки даних. Законом передбачено створення загальної системи охорони праці.

Закон поділяє інформацію на:

1. Інформація про фізичну особу
2. інформація довідково-енциклопедичного характеру
3. інформація про стан довкілля (екологічна інформація)
4. інформація про товар (роботу, послугу)
5. науково-технічна інформація
6. податкова інформація
7. правова інформація
8. статистична інформація
9. соціологічна інформація
10. інші види інформації.

Категорії інформації і режими доступу до неї визначаються такими статтями Закону: Стаття 20 Доступ до інформації. Стаття 21 Інформація з обмеженим доступом.

1.3.2. Закон "Про державну таємницю”

Державна таємниця - інформація у сфері економіки, оборони, техніки, науки, державної безпеки, зовнішніх відносин і охорони правопорядку, розголошення інформації може завдати шкоду національній безпеці і яка спеціально охороняється державою.

Інформацію яку потрібно віднести до державної таємниці визначають експертами з таємниць і затверджується СБУ у формі Зводу відомостей.

Носіям інформації надається один із грифів доступу до інформації: таємно, цілком таємно або особливої важливості. Надавати грифу обмеження доступу, носіями інформації, що не містять відомості, які підпадають під ЗВДТ, заборонено.

Доступу до інформації що відносять до державної таємниці мають посадові особи, по потребі роду їхньої діяльності.

1 форма допуску дає доступ до інформації з грифом: таємно, цілком, таємно, особливої вадливості.

2 форма допуску дає доступ до інформації з грифом: таємно, цілком таємно.

3 форма допуску дає доступ до інформації з грифом: таємно.

Посадова особа несе кримінальну відповідальність за розгошення інформації що відноситься до державної таємниці.

Стаття 328 Кримінального кодексу України передбачає покарання за розголошення державної таємниці: Розголошення державної таємниці. Покарання позбавлення волі на термін від 2 до 5 років, якщо наслідки важкі до від 5 до 8 років.

За часів АТО була виявленна прогалина в регулюванні доступу до інформації що відноситься до державної таємниці. Оскільки було погано врегульоване питання швидкого позбавлення рівня доступу до інформації. Таким чином депутати що були проросійко налаштованими відвідували окуповані території в результаті чого була розголошена інформація з статусом державної таємниці. Такій критиці піддався

депутат Богуслаєв що контактуква з російськими “партнерами” частіше ніж з своїми колегами і мав доспут до даної інформації.

Висновки

Ознайомленно з основною термінологією в криптографії.
Ознайомленно з основою криптографії
Ознайомленно з криптоаналізом
Ознайомленно з законодавством України

2. КЛАСИЧНІ І СУЧАСНІ ТЕХНІКИ ШИФРУВАННЯ

2.1 Шифри підстановок

Підстановичний шифр, також відомий під назвою моноалфавітний шифр - алгоритм шифрування, роль якого полягає в заміні знаків публічної інформації іншими знаками, що служить ключем.

Даний шифр схожий на алфавітне письмо, значення символів що не відоме особам, які намагаються прочитати інформацію.

Нажаль відсутня інформація про час коли саме почали використовувати даний шифр в перше. Але першими задокументованими згадками про застосування були за часів Давньої Греції і Римської імперії. Яскравим прикладом даного шифру служить шифр Цезаря.

Шифр Цезаря базується на заміні кожного елемента алфавіту на елемент що віддалений на певну кількість одиниць. Назва же зародилася в честь Юрія Цезаря, який використовував його у своїх приватних листуваннях. Для своїх листів він зсував на 3 елемента алфіту, тобто А ставала D, В - Е і так далі.

Але даний алгоритм вразливий до частотного аналізу і особливо до повного перебору. Для українського алфавіту він становитеме 32 комбінації. Сьогодні він не надає ніякого захисту. Але на основі нього створили шифр Віженера.

Принцип дії даного алгоритму полягає в суві алфавіту, а ось ключем буде саме на скільки елементів потрібно його зсувати.

Оскільки всі елементи алфавіту розташовані в строгому порядку і кожен має свою позицію, то на основі цього можна сформувати формулу по які працює даний алгоритм

$$y = (x + k) \bmod n. \quad (1.1)$$

$$x = (y - k) \bmod n. \quad (1.2)$$

x - порядковий номер символу інформації,

y - порядковий номер символу шифротексту

k - ключ

n - кількість символів у алфавіті

2.1.1. Шифр Play-fair

Шифр Плейфера - це шифр в якому вперше використовувалася заміна біграм. Винайшов його англійський фізик Чарльз Вітстон у 1854 році, але назву отримав в честь лорда Лайона Плейфера, який просував використання його у різних державних службах. Даний алгоритм є набагато складнішим що, у свою чергу його важче взаламати порівняно з шифром підстановок чи Віженера. Всього можливо 26 * 26 = 676 біграм порівняно з 26 монограмами, це стосується латинського алфавіту.

Для роботи шифр Плейфера використовує матрицю 5 * 5 для латинського алфавіту, та 4 * 8 для кириличного алфавіту. Також потрібно запам’ятати ключові слова і чотири правила. Ключові слова у матриці можна записувати любим чином. Далі потрібно інформацію розбити на біграми.

4 правила:

1. Якщо символи збігаються у біграмі, між ними ставимо символ X, зашифровуємо нову пару символів. Дехто замість X використовує Q.
2. Якщо символи які були у відкритій інформації зустрічаються у одному рядку, тоді ці символи замінюють на символи що знаходяться найближче з правої сторони. Якщо символ був останній, тоді звертаємся до першого символа рядка.
3. Якщо символи що були в одній біграмі зустрілися в одному стовпці, тоді вони стають в символи того ж ствопця тільки ті що під ними. Якщо символи останній дія відбувається з першим елементом стовпця.
4. Якщо ж символи не є в одному рядку чи стовпця, заміна відбувається з тими же рядками тільки відповідні інші кути прямокутника.

Для розшифрування повідомлення потрібно всі правила проробити інверсно.

Як і будь який шифр він піддається взлому. І перший опис цього процесу був описаний у 1914 році лейтенантом Джозеф О. Моуборном. Також криптоаналіз шифру був опоблікований у книзі Х. Ф. Гейнс «Cryptanalysis — a study of ciphers and their solution».

Вся проблема у певних закономірностях, те що одна зашифрована біграма відповідає розшифрованій іншій біграмі, наприклад AB | BA і BA | AB. І англійська мова має багато слів що мають цю проблему, наприклад REceivER і DEpartED. Це

допоможе знайти багато слів в результаті чого ми отримаємо фрагменти на основі яких зможемо відтворити ключ.

З часом шифр Плейфера намагалися вдосконалити і тому збільшили матрицю і додали нових слів, але це забрало основну вразливість шифру, завдяки якій можна відтворити інформацію.

2.1.2. Шифр Віженера

Це метод поліалфавітного шифрування буквеного тексту з використанням ключового слова. Він є спрощеною формою багатоалфавітної заміни. Він був розроблений декілька разів в різні періоди часу. Першим був Джовани Баттиста Белласо. Він описав його а книзі La cifra del. Sig. Giovan Battista Bellaso в 1553 році. Але у 19 столлітті отримав назву Блеза Виженера. Сам метод є доволі простий для розуміння, але не піддається простим методам криптоаналізу. Для доказу в його стійкості, його не могли взламати протягом 3 століть, в результаті чого отримав назву “Нерозгаданий шифр”.

По своїй будові він складається з декількох шифрів Цезарів з різними значеннями зсуву. Для шифрування використовується квадрат Віженера. Для латинського алфавіту таблиця буде становити 26 симолів в ширину і висоту. В результаті чого шифр Віженера для латиського алфавіту є сукупністю 26 шифрів Цезаря. Для українсткого ж алфавіту він становитеме 33.

Для прикладу візьмемо слово ATTACKATDAWN і слово LEMON для ключа. Щоб сформувати ключ потрібно слово LEMON повторювати стільки раз поки його довжина не буде відповідати слову яке потрібно зашифрувати, в результаті отримуємо LEMONLEMONLE. Першу букву A зашифровуємо рядком L з таблиці. Для другої букви T зашифровуємо рядком E і так весь текст. В результаті ми отримаємо LXFOPVEFRNHR. Для розшифрування виконуємо ці дії зворотньо. Для прикладу беремо першу букву зашифрованого тексту L і по рядку L отрмаємо початкову букву A.

Головним недоліком даного шифру є повторення його ключа. Також часто повторювальні символи. Для взлому потрібно визначити довжину ключа. Для цього варто використати тести Фридмана і Касиски.

2.1.3. S-блоки

S - блоки - це функція в коді програми або апаратній системі, що приймає на вхід n біт, обробляє їх по певному алгоритму і на виході вертає m біт. Їх використовують для блочних шифрів.

Для реалізації програмним способом використовують таблицю заміни. Апаратна реалізується методом використення схеми. Ці два способи є ідентичними, тобто зашифрувавши одним способом його можна розшифрувати іншим і навпаки.

Ідеальними S-блоками вважається якщо вихідний біт вираховується бент-функцією на основі значення вхідних біт, також люба лінійна комбінація вихідних біт є бент-функцією від вхідних даних.

Для програмної реалізації потрібно опрацювати дані на вході, виконати пошук по значенню в таблиці і отримати з комірки таблиці потрібні дані. Така таблиця вважається таблицею заміни. Таблиця може бути як статична так і генеруватися на основі ключа. Для прикладу для алгоритму шифрування DES використовується статична таблиця, тоді як для Twofish вона генерується на основі ключа.

Для апаратної реалізації потрібно: шифратор, дешифратор, система комутаторів.

Дешифратор перетворює n розрядний бінарний сигнал в сигнал 2^n. Система комутаторів виконує перестановку біт. Кожен вхідний біт відображає вихідний біт. Шифратор перетворює 2^n розрядний сигнал у бінарний n сигнал.

2.2. Шифри перестановки

Шифр перестановки - це симетричний шифр, що міняє елементи вхідного тексту місцями. В даному випадку це може бути як один символ так пара з декількох символів. Яскравим прикладом такого шифр є анаграми.

Даний шифр можна розділити на 2 групи: простої перестановки і складної перестановки. При простій перестановці елемент переміщається тільки один раз з початкової позиції. При складній перестановці елемент перемщається декілька разів з початкової точки.

Альтернативою даному шифру є шифр підстановки, в якому елементи не

міняють свого положення в тексті, відбувається їхня заміна.

Для шифру простої перестановки використовують таблицю перестановки.

Для перестановки існують декілька способів:

1. шифри маршрутної перестановки,
2. шифри табличної маршрутної перестановки,
3. шифри вертикальної перестановки,
4. шифри поворотна решітка.

Шифр маршрутної перестановки відбувається за допомогою спеціальної фігури. Даний шифр запусує потрібне нам повідомлення по певній траєкторії, а виписує по іншій. Прикладом такого шифру служить шифр Скитали.

Шифр табличної маршрутной перестановки є найбільш популярний серед усіх видів перестановки. В основі даного методу є таблиця. Повідомлення ми записуємо в таблицю з ліва на право, при завершинні рядка переход на наступний і продовжуємо його заповнювати. Для шифрування потрібно записати всі симоли з гори в низ, з права на ліво.

Шифрограма вертикальної перестановки в його основі лежить спосіб табличної маршрутної перестановки з різницею при шифруванні таблиці не з права на ліво, а згідно ключу в якому заданий порядок який стовбець за яким буде іти.

Шифр поворотної решітки, для даного способу потрібно виготовити трафарет на якому будуть вирізані комірки. Ці комірки повинні вирізані таким чином щоб кожна з клітинок при прокручуванні трафаретки по листку 4 способами по одному разу покривала певну область. При шифруванні трафарет накладають на таблицю і заповнюють клітинки. Після цього його прокручують ще 3 рази і повторють операцію з заповненням. Шифрограму виписують по отриманій таблиці по певному маршруту. Ключом для шифру служить трафарет, порядок його поворотів і маршрут виписування.

При шифрі складної перестановки текст записують по певному маршруту записують текст. Після чого по певному маршруту виписується шифрограма. Ключом до даного шифру служить розмір таблиця, маршрут порядок перестановки стовпців і рядків. Якщо маршрут фіксований, тоді кількість ключів n!m!, де n - кількість рядків, m - кількість стовпців.

2.3. Симетричні шифри

Симетричний алгоритм як видно з назви використовує один ключ як для шифрування так і для розшифрування. Переваги даного алгоритму в швидкодії, потреба в менших обчислювальних потужностях. Найпоширенішими прикладими серед симетричних алгоритмів: AES, RC4, DES, 3DES, RC5, RC6.

2.3.1. Стандарт шифрування DES

DES (Data Encryption Standard) - стандарт що був прийнятий урядом США з 1976 року. Через деякий час став популярний у всьому світі. Але отримав неоднозначні відгуки протягом деякого часу. Оскільки перед тим як прийняти даний шифр уряд відправив фого на аналіз до Національного Агентства Безпеки США, який його трохи змінив. Таблиці які генерувалися стали статичними, і люди почали побоюватися що завдяки даним змінним вони спростили собі роботу для можливості розшифрування інформації яка була защифрована даним шифром. Також критика була в сторону того що для даного шифру був потрібен доволі малий за розміром ключ. Але даний шифр дав основу для блочних алгоритмів та криптоаналізу.

На даний момент DES уже не вважається найдійним і уряд США віднього відмовився ще у 2002 році, замінивши його більш сучасним алгоритмом AES. Причиною послужило те шо 1999 році ключ був дишефрований публічно протягом 22 голин і 15 хвилин. Зараз же даний алгоритм трохи вдосконалили до версії під назвою 3-DES, але і для нього вже є теоретичні стратегії для взому.

Робота над даним алгоритмом почалася після оголошення конкурсу НБС над розробкою нового алгоритму шифрування. Спочатку організували конкурс 15 травня 1973 року, але він він задавав строгі архітектурні вимоги, в результаті чого була відсутність заявок. Саме тому, організували інший конкурс 27 серпня 1974 року. Цього разу IBM подала заявку яка задовільнала всім вимогам. Вони його розробляли протягом 1973-1974 року і в його основу було закладений шифр Люцифер, розроблений Хостом Фейля.

Алгоритму DES відносять до алгоритмів блочного шифрування, де всі елементи розбиваються на 64 бітні блоки. Ці дані подаються на вхід і отримаються

на виході роботи алгоритму. Щоб виконати операції шифрування і розшифрування ми використовуємо один і той же самий алгорим шифруванння.

Для роботи алгоритму потрібен ключ розміром 56 бітів. Хоча з джерела ми беремо 64 біти і 8 з них використовуємо для паритету. При підборі ключа можемо використувати довільну комбінацію, але не потрібно забувати що потужність алгоритму базується саме на його ключі, тому деякі комбінації можуть бути слабкими і їх краще не використовувати.

В самі основі алгоитму відбуваються дві основні дії підстановка і переміщування. І для роботи DES виконується поєднання цих двої технік.

2.3.2. Алгоритм DESX

Алгоритм DESX - це симетричний алгоритм шифрування, розроблений на основі блочного шифру DES. В його основі використовується метод відбілювання ключа з метою підвищити захист прото акаки грубої сили, тобто повним перебором.

Суть даного алгоритму в накладані операції XOR різних частин 64 бітного ключа як до, так і після виконання одноразового DES.

$$DES - X(M) = K_2 \oplus DES_K(M \oplus K_1) \qquad (2.1)$$

В результаті чого довжина ключа становить 56 + 64 + 64 = 184 біт.

DESX сумісний з DES у випадку K1 = K2 = 0. Розробники даного алгоритиму рекомендують використовувати алгоритм SHA-1 перед використанням. Також даним алгоритмом допущено використовувати 120 бітний ключ, де K1 = K2. Компанія RSA розширила його таким чином, що отримуємо K, K1, K2, де K2 є лише функцією 16 байтів. При потребі замість операції XOR можна замінити на операцію додавання по модулю.

Хоч для даного алгоритму додатково додали операції, що сповільнять його. В результаті швидкість алгоритму DESX приблизно рівна DES. Також із плюсів він має апаратну реалізацію.

Криптостійкість даного алгоритму по сьогоднішнім міркам низька, оскільки довжина ключа 184 біт, з якого ефективно використовується $56 + 64 - 1 - log_2(M) = 119$ біт, де M - число відомих пар відкритого тексту. Крім того він знижується до 88 біт при 2^{32} при використанні атаки на основі адаптивно

підібраного відкритого тексту.

З плюсів DESX покрашує стійкість до лінійного і диференційному криптоаналізу, хоч вони і не значні. Для лінійного криптоаналізу потрібно 2^{60} відритого тексту, тоді як для DES становить 2^{43}. Для диференційного криптоаналізу 2^{61}, порівняно з 2^{47} для DES.

2.3.3. Стандарт IDEA

IDEA - це симетричний блочний алгоритм, що запатентований швейцарською фірмою Ascom. Він був запропонований в рамках програми Європейської комісії IST як кандидат проекту NESSIE. IDEA є вдосконаленою версією мережі Фейстеля.

Для даного алгоритму потрібен 128 бітний ключ і 64 бітний розмірний блок. Для роботи алгоритму текст розбивається на частини по 64 біти. Але для більшості текстів останій блоку інформації не буде відповідати розміру 64 біти, в такому випадку текст доповнюють інформацієї певної послідовності. Для кращого захисту і захисту від витоку інформації, різні блоки шифруються різними алгоритмами. Всі блоки діляться на підблоки по 16 біт, оскільки алгебраїчні операції використовують 16 біт кожен. Для шифрування і розшифрування використовують один і той самий алгоритм.

Стандарт IDEA складається з:

1. Додавання по модулю 2^{16}
2. Множення по модулю $2^{16} + 1$
3. Побітова виключна диз'юнкція (XOR)

Перевагою використання цих операцій в їхньому не дотримання дистрибутивному закону, тобто формула 2.2

$$a \odot (b \oplus c) = / = (a \odot b) \oplus (a \odot c). \qquad (2.2)$$

В результаті чого він важче піддається криптоаналізу і не потребує S- блоків і таблиць заміни.

Генерація ключів відбувається у формування 16 бітних ключів у кількості 6 для кожних вісім раундів. А для вихідного перетворення потрібно 4 ключа, також по 16 біт кожен. В результаті потрібно 52 таких підключів.

Таблиця 2.1. Підключів для кожного раунду

Номер раунду	Підключів
1	$K_1^{(1)}K_2^{(1)}K_3^{(1)}K_4^{(1)}K_5^{(1)}K_6^{(1)}$
2	$K_1^{(2)}K_2^{(2)}K_3^{(2)}K_4^{(2)}K_5^{(2)}K_6^{(2)}$
3	$K_1^{(3)}K_2^{(3)}K_3^{(3)}K_4^{(3)}K_5^{(3)}K_6^{(3)}$
4	$K_1^{(4)}K_2^{(4)}K_3^{(4)}K_4^{(4)}K_5^{(4)}K_6^{(4)}$
5	$K_1^{(5)}K_2^{(5)}K_3^{(5)}K_4^{(5)}K_5^{(5)}K_6^{(5)}$
6	$K_1^{(6)}K_2^{(6)}K_3^{(6)}K_4^{(6)}K_5^{(6)}K_6^{(6)}$
7	$K_1^{(7)}K_2^{(7)}K_3^{(7)}K_4^{(7)}K_5^{(7)}K_6^{(7)}$
8	$K_1^{(8)}K_2^{(8)}K_3^{(8)}K_4^{(8)}K_5^{(8)}K_6^{(8)}$
Вихідне перетворення	$K_1^{(9)}K_2^{(9)}K_3^{(9)}K_4^{(9)}$

Як було вказано раніше для шифрування потрібно 8 раундів і одного вихідного перетворення. Незашифрований вихідний текст ми розбиваємо на блоки по 64 біти, що діляться на підблоки по 16 біт. Кожен блок має свою назву: D1, D2, D3, D4. Після чого кожен раунд має свій ключ для даних операцій.

Кожен раунд складається з операції множення по модулю 2^16 + 1, в якому замість нуля використовується 2^16, додавання по модулю 2^16, побітове виключення АБО. Після завершення раунду вихідні дані стають вхіднимим для наступного ранду. Після восьмого раунду відбуваються операції множення по модулю і додавання по модулю. В результаті після вихідного перетворення конкатенації підблоків є зашифрований текст, і береться наступний 64 бітний блок над яким проводиться аналогічні операції.

Математичний опис

$$A^{(i)} = D_1^{(i-1)} * K_1^{(i)} \quad (2.3)$$

$$B^{(i)} = D_2^{(i-1)} * K_2^{(i)} \quad (2.4)$$

$$C^{(i)} = D_3^{(i-1)} * K_3^{(i)} \quad (2.5)$$

$$D^{(i)} = D_4^{(i-1)} * K_4^{(i)} \quad (2.6)$$

$$E^{(i)} = A^{(i)} \oplus C^{(i)} \quad (2.7)$$

$$F^{(i)} = B^{(i)} \oplus D^{(i)} \quad (2.8)$$

Таблиця 2.2. Підключення для кожного раунду

Номер раунду	Підключі
1	$1/K_1^{(9)} - K_2^{(9)} - K_3^{(9)} 1/K_4^{(9)} K_5^{(8)} K_6^{(8)}$
2	$1/K_1^{(8)} - K_2^{(8)} - K_3^{(8)} 1/K_4^{(8)} K_5^{(7)} K_6^{(7)}$
3	$1/K_1^{(7)} - K_2^{(7)} - K_3^{(7)} 1/K_4^{(7)} K_5^{(6)} K_6^{(6)}$
4	$1/K_1^{(6)} - K_2^{(6)} - K_3^{(6)} 1/K_4^{(6)} K_5^{(5)} K_6^{(5)}$
5	$1/K_1^{(5)} - K_2^{(5)} - K_3^{(5)} 1/K_4^{(5)} K_5^{(4)} K_6^{(4)}$
6	$1/K_1^{(4)} - K_2^{(4)} - K_3^{(4)} 1/K_4^{(4)} K_5^{(3)} K_6^{(3)}$
7	$1/K_1^{(3)} - K_2^{(3)} - K_3^{(3)} 1/K_4^{(3)} K_5^{(2)} K_6^{(2)}$
8	$1/K_1^{(2)} - K_2^{(2)} - K_3^{(2)} 1/K_4^{(2)} K_5^{(1)} K_6^{(1)}$
Вихідне перетворення	$1/K_1^{(1)} - K_2^{(1)} - K_3^{(1)} 1/K_4^{(1)}$

$$D_1^{(i)} = A^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \quad (2.9)$$

$$D_2^{(i)} = C^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \quad (2.10)$$

$$D_3^{(i)} = B^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \quad (2.11)$$

$$D_4^{(i)} = D^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \quad (2.12)$$

I - номер раунду.

$$D_1^{(9)} = D_1^{(8)} * K_1^{(9)} \quad (2.13)$$

$$D_2^{(9)} = D_3^{(8)} + K_2^{(9)} \quad (2.14)$$

$$D_3^{(9)} = D_2^{(8)} + K_3^{(9)} \quad (2.15)$$

$$D_4^{(9)} = D_4^{(8)} * K_4^{(9)} \quad (2.16)$$

Для рощифрування потрібно використовувати ключі в зворотньому порядку.

2.4. Асиметричні шифри

Алгоритми з асиметричним ключем є більш безпечними ніж алгоритми з симетричним ключем. Даний алгоритм також називають криптосистема з відкритим ключем. В такій системі використовуютиься два ключі, один з них використовується для шифрування інформації і він називається відкритим, тоді як для розшифрування використовується інший ключ, який називається закритим.

Розшифрувати інформацію завдяки відкритому ключу не можливо, і він є публічним і може бути у вільному доступі. Це є безпечно, оскільки завдяки ньому максимум що можна зробити це зашифрувати інформацію. Але закритий ключ потрібно максимально ефективно захищати, і доступ до нього повинен бути обмежений.

Завдяки такій структурі люди, що не мають домовленостей про безпеку, надсилати один одному повідомлення, не переживаючи через серектретний канал завдяки якому потрібно обмінятися ключами. Криптографічні системи що використовують відкритий ключ відносяться RSA, Діффі-Геллмана, DSA.

Даний тип алгоритмів шифрування прийшов до нас як рішення до проблеми з передачі ключів між двома обєктами комунікації.

Ідея криптографії з відкритим ключем полягає в ідеї односторонніх функцій. До прикладу таких односторонніх функцій є коли ми ми можемо отримати f(x) з x, але не можеио отримати x з f(x). Тому такі алгоритми використовують лазійку. Лазійка - це секрет, що допомагає отримати x з f(x), для прикладу y. Яскравим прикладом такої лазійки буде, якщо ми розберемо години на деталі, але не будемо знати як його скласти назад. В такому випадку нам на допомогу прийде інструкція, якщо допоможе скласти його назад.

2.4.1. Алгоритм RSA

Даний алгоритом побудований на складності оюбчислення факторизації великих цілих чисел. В його будову входить: генерація ключів, шифрування, розшифрування та поширення ключів.

Для роботи даного алгоритму потрібно згенерувати ключі. Для цього обчислити добуток двох великих простих чисел q і p, n = p * q. Ця операція не є зворотною, оскільки для знаходження потрібних p і q, при $n = 2^{664}$, потрібно виконати 2^{23} операцій, що потебує багато часу для сучасних ЕОМ. Далі обчислюємо функцію Ейлора, де $\varphi(n) = (p - 1)(q - 1)$. Після чого генеруємо ціле число яке буде більше за 1 та менше за $\varphi(n)$ і взаємно простим з число $\varphi(n)$. Завдяки алгоритму Евкліда находимо число d яке задовільняє $e \cdot d \equiv 1 (mod\ \varphi(n))$ і d менше за n. Після цих операцій ми маємо відкритий ключ e і d, що є закритим. p і q не

повинні бути опублікованими.

Шифрування відбувається завдяки формулі:

$C = m^e mod$ n (2.17)

M - повідомлення яке потрібно зашифрувати

C - зашифроване повідомлення

Розшифрування відбувається завдяки формулі:

M = c^d mod n (2.18)

C - зашифроване повідомлення

M - повідомлення яке ми розшифрували

Даний алгоритм також використовується для цифрового підпису. Для цього потрібно скористатися формулою

S = m^d mod n (2.19)

S - підпис повідомлення m

Щоб переконатися що підпис є правильним, потрібно скористатися формулою:

m = s^e mod n (2.20)

Для безпечного використання алгоритма RSA потрібно щоб довжина ключа становила 512 біт, але ця інформація є достовірною станом на 2007 рік.

2.4.2. Криптосистема Рабіна

Дана система побудована на проблемі факторизації великих цілих чисел. Дана система була першою серед алгоритмів шифрування з асиметричним ключем шифрування де було доведено як отримувати початкові дані з зашифрованих. Також вона побудована на проблемі отримання квадратного корення за модулем складеного числа n = p * q.

Ми можемо отримати квадратний корінь за модулем n, маючи прості дільники даного числа n. Крім того вміння отримувати квадратний корінь за

модулем n.

Навединні вижче аргументи в сторону переваг даної криптосистеми показують перевагу його над RSA. Дана криптосистема швидше шифрує дані порівняно з іншими системами побудованих на асиметричних ключах. Але вона лягла в основу багатьох критосистема які є набагато потужнішими.

Щоб скоритсатися даного криптосистемою потрібно обрати два числа. Вимоги які стоять перед ними, вони повинні бути простимим, великими і виконувати умову $p \equiv q \equiv 3 \ (mod\ 4)$. Потреба в дотриманні даних вимог є по причині простоти добування корення за моделем до них. Також ці два числа служать як секретний ключ, а ось їхній добуток є відкритим ключем.

Для шифрування викоритовується формула:

$C = m^2 \pmod N$ (2.21)

M - повідомлення

C - зашифроване повідомлення

N - відкритий ключ

Для розшифрування формула:

$$M_p = c^{\frac{1}{4}(p+1)}(mod p) \quad (2.22)$$

$$M_q = c^{\frac{1}{4}(q+1)}(mod q) \quad (2.23)$$

Наступним кроком буде розшифрування алгоритма Евкліда знаходячи yp і yq, при умові yp * p + yq * q = 1

Для отримання повідомлення скористаємося китайською теоремою про залишки:

R1 = (yp * p * mq + yq * q * mp) (mod n) (2.24)

R2 = n - r1 (2.25)

R3 = (yp * p * mq - yq * q * mp) (mod n) (2.26)

R4 = n - r3 (2.27)

Одна з чотирьох результатів буде вірною. Щоб визначити яка з них є правильна, потрібно використати додаткову інформацію, завдяки якій визначимо правильний результат.

Криптосистема доказала свою стійкість перед атакою “все або нічого”, але при умові що розклад відкритого ключа на прості множники буде важко виконати. Але вона є беззахисною перед атакою з використанням підібраного шифротекста.

2.4.3. Криптосистема Ель-Гамаля

Дана криптосистема використовується як для формування цифрового підпису, так і для шифрування даних. Побудованна вона на складності в обрахунках над дискретними логарифмами в кінцевому полі.

Висновок

Проаналізовано алгоритми перестановок.

Проаналізовано алгоритми підстановок.

Проаналізовано алгоритми з симетричним ключем.

Проаналізовано алгоритми з асиметричним ключем.

3. АЛГОРИТМ ШИФРУВАННЯ AES

3.1. Математичні поняття алгоритму

В основі алгоритму шифрування AES лежать наступні поняття: поле Галуа, поліноміальне уявлення, незвідний многочлен, сумування двох поліномів, множення по модулю.

Поле Галуа, також відоме як скінченне поле - яке має скінченну кількість множини елементів. Найменше за розміром поле Гаула рівне 2, що складається з двох елементів 0 та 1. Усі арифметичні операції є традиційними для нього окрім 1 + 1 = 0, це звичайна практика у комп'ютерних науках, дискретній математиці і теорії кодування.

Ідея використання поля Гауса в доцільності розгляду послідовностей, що складаються з нулів та одиниць, які є алгебраїчною структурою векторного простору.

Поліномне уявлення двійкових чисел. В основі якого лежить заміна фиктивної зміни, наприклад X. Математична формула:

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2a^2 + a_1x + a_0x \qquad (3.1)$$

Байт B скаладається з a7, a6, a5, a4, a3, a2, a1, a0 записується у формі полінома з {0, 1}. В результаті отримуємо значення 87, що у двійковій формі має вигляд 01010111, а у формі полінома $x^6 + x^4 + x^2 + x + 1$.

Незвідний многочлен - це многочлен з коефіцієнтами для довільного поля, якщо він не рівний константі та не рівний добутку двом або більше многочленам, які не є константами. Кожен многочлен може бути розкладеним у добуток многочленів. Цей розклад на множники є визначеним з точністю до перестановки множників.

Сумування двох поліномів відбувається за рахунок побітового XOR.

3.2. Опис алгоритму

Даний алгоритм був побудований на основі алгоритму Rijndael. Складається він з таких елементів: Block, Cipher Key, ciphertext, Key Expansion, Round Key, State, S-box, Nb, Nk, Nr, Rcon[].

Block - це послідовність біт яка лижить в основі state, roundKey, input, output.

Cipher Key - секретний криптографічний ключ, який ми використовуємо в результаті для процедури Key Expansion, для генерування ключів Round Key.

Ciphertext - це дані які ми отримуємо після шифрування даних

State - це проміжковий результат шифрування

S-box - це таблиця для нелінійної заміни.

Nb - кількість стовпців, що має State, який рівний 4

Nk - число 32 бітних слів, який може бути 4, 6, 8.

Nr - кількість раундів, яка може бути як 10, так і 12 чи 14.

Rcon - масив даних, що складає біти 32 розрядного слова і є статичним для всіх раундів.

AddRoundKey - операція на Round Key і State завдяки XOR.

Довжина вхідних даних рівна 128 бітам, а ось ключа шифрування 128, 192 або 256. Хоча для початку шифрування копіює в масив State згідно правил де state[r, c] = input[r + 4c], коли 0 <= r < 4, 0 <= c < Nb. Наступним кроком буде використання методу AddRoundKey(), після чого процес трансформації інформації завдяки операціям SubBytes, ShiftRows, MixColumns, AddRoundKey згідно вказаного числа раундів вони будуть виконуватися. Для 128 біт кількість раундів буде рівна 10, для 192 - 12 і для 256 - 14 раундів. В результаті ми отримаємо output[r + 4c] = state[r,c].

3.2.1. Підстановка байтів

Дана операція виконує нелінійну заміну байтів, вона виконується над кожним байтом state. Така операції є зворотньою зарахунок поєднання двох перетворень що йдуть на вхід.

Таке перетворення виконується множенням інверсного байта на многочлен формолую $a(x) = x^4 + x^3 + x^2 + x^1 + x^0$ і сумування з многочленом $b(x) = x^6 + x^5 + x + 1$ в полі $f_2[x]/x^8 + 1$.

Для матриці вона буде виглядати наступним чином:

Якщо на вході ми отримаємо байт що рівний 0, тоді y = b.

На основі даної операції створенна S таблиця в шістнадцятковій системі

числення.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad (3.2)$$

Sbox = array{

0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,

0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,

0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,

0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,

0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,

0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,

0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,

0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,

0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,

0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,

0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,

0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,

0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,

```
 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf,
      0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0,
0x54, 0xbb, 0x16
   };
```

3.2.2. Зсув рядків

Ця операція виконує зсув рядків таблиці, тобто 0 рядок зсувається на 0 байт, 1 рядок зсувається на 1 байт і так далі.

3.2.3. Перемішування стовпців

Ця операція виконує змішування елементів колонки, але вона використовує зворотною лінійну трансформацію. Обробляє як стан по колонкам і трактує кожну як поліном третьої ступені. Над цим поліном відбувається множення $GF(2^8)$ по модулю $x^4 + 1$ на фіксований многочлен $c(x) = 3x^3 + x^2 + x + 2$. Матричний вигляд даної операції формула 3.3

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} -> \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \quad (3.3)$$

3.2.4. Додавання циклового ключа

Ця операція виконує операцію XOR над кожним байтом state і round key.

3.3. Утворення ключів

Генерація ключів відбувається завдяки методу KeyExpansion і передачі в неї cipherkey, в результаті ми отримуємо ключі для всіх раундів. Кількість таких ключів становить Nb * (Nr + 1).

Функція SubWord використовує чотирибайтне слово на вхід і використовує для кожного з чотирьох байтів S таблицю. Отримані дані йдуть на вихід. Для метода RotWood() надсилаємо дані у формі [a0, a1, a2, a3] в результаті перестановок ми отримаємо на виході дані [a1, a2, a3, a0]. Масив слів rCon , що є статичним, має

значення даних $[x^{(i-1)},0,0,0]$ де $x^{(i-1)}$ є степеню x в $GF(2^8)$.

Також перші символи з Nk слів заповненні розширеним ключем cipher key. Далі, кожне слово w[i] записуємо значення після опреації XOR w[i - 1] та w[i - Nk]. Для слів позиція яких рівна Nk перед операцією XOR з w[i] виконується XOR з rCon[i]. Виконуємо RotWord() і SubWord(). Різниця SubWord() і SubBytes() лише вхідних даних, де для SubWord - розміром з слово, а для SubByte - розміром з байти.

На кожній ітерації раундовий ключ отримуємо з масиву w, з інтервалом w[Nb * i] до w[Nb * (i + 1)]

3.4. Шифрування

Шифрування проходить в наступні етапи. Спочатку ми отримуємо значення state з конкретного блоку повідомлення. Після чого ми отримуємо потрібні нам ключі для кожного раунду завдяки методу KeyExtension. В результаті чого запускаємо метод AddRoundKey. Після чого ми циклічно викликаємо методи SubBytes, ShiftRows, MixColumns, AddRoundKey, згідно кількості визначиною довжиною шифрування.

Висновок

Проаналізовано математичні основи алгоритму AES.

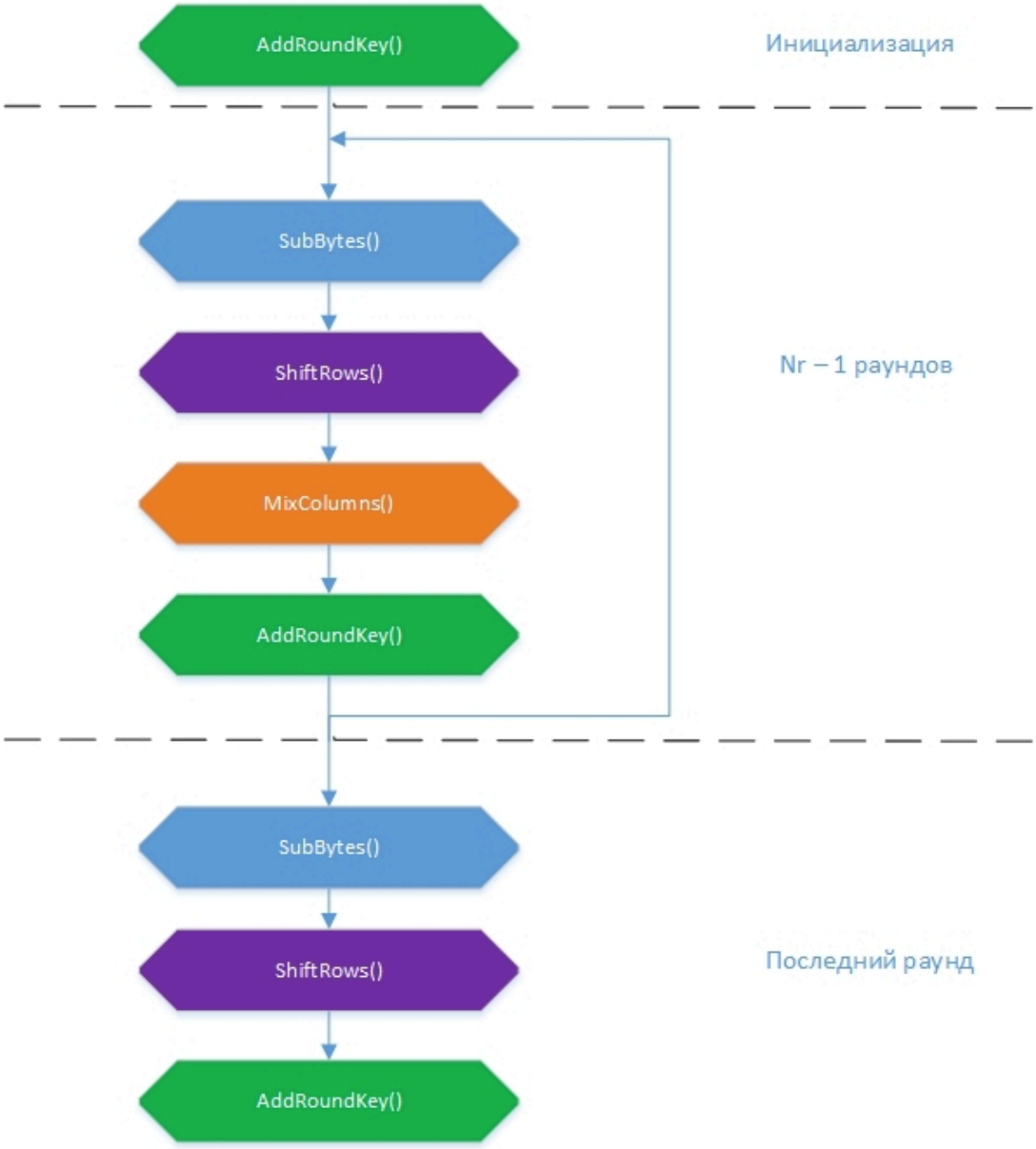
Проаналізовано принцип роботи AES.

Проаналізовано шифрування AES

4. ПРАКТИЧНА РЕАЛІЗАЦІЯ АЛГОРИТМУ AES

4.1. Блок схема алгоритму

Рисунок 4.1 Блок схема алгоритму



На рисунку 4.1 зображена блок схема виконня команд шифрування алгоритму AES.

4.2. Особливості реалізації алгоритму на мові Java

Для реалізації даного алгоритму на мові Java був розроблений клас Crypto що мав два публічних методи як для шифрування encrypt(String message, String key) так і для розшифрування decrypt(String encryptText, String key). Також були ще два публічних методи для перевірки довжини ключа verificateLengthOfKey(String key) і

перевірки вірності вказаного розмірності шифрування verificateOfBit(int bit).

Метод String encrypt(String message, String key) визнаає довжину повідомлення. Після чого створює масив ключів методом createArrayKeyFromString(key) з рядка. Також повідомлення розбивається на таблиці і кожна таблиця шифрується методом AES_Encrypt(Arrays.copyOfRange(paddedMessage, i, i + lengthOfKey), keyCharArray). Після чого вертає зашифроване повідомлення.

Метод createArrayKeyFromString перетворює ключ з рядка в масив символів.

Метод char[] AES_Encrypt(char[] message, char[] key) спосатку отримує ключі для кожного раунду шифрування після чого викликає метод AddRoundKey(state, key). Потім циклічно викликає методи SubBytes(state), ShiftRows(state), MixColumns(state), AddRoundKey(state, expandedKey) кількість визначиною для кожної розмірності шифрування. І на завершення викликаємо SubBytes(state), ShiftRows(state), AddRoundKey(state, Arrays.copyOfRange(expandedKey, 0, 159))

Піт час написання дипломної роботи досліджено принцип роботи популярного алгоритму шифрування AES. Даний алгоритм відноситься до типу з симетричним ключем шифрування. Також в його основу закладений блочний шифр.

У завданні на роботу була поставлена задача вивести такий алгоритм шифрування, щоб його псевдокод був універсальний для більшості мов програмування і він був універсальний, тобто його можна використати на різних мовах і платформах.

Для виконаня даного завдання було досліджено загально доступні ресурси де розкривається алгоритм AES. І більшість з них були саме привязанні до основ певної мови і перенесенна на інші були доволі проблематичні або не можливим. Також в цьому може бути проблема оскільки доволі багато мов мають або вбудовані пакети або додаткові пакети де міститься даний алгоритм і все що необхідно це просто викликати даний пакет, підставити ключ і дані для шифрування і більше нічого не потрібно від розробника. Але може бути проблема з тим що якась мова не має даного пакету або використання не є можливим. Саме для вирішення даної проблеми розроблений варіант, який є інваріантний до мов програмування і тому універсальний, для його роботи використовуються інструменти що є у більшості

мов як і програмування, так і скріптових.

Даний алгоритм може бути як 128, 192, 256 бітним. На даний момент безпечним вважається мінімум 128 біт. Але коли підніметься планка і 128 біт не буде вважатися безпечним, для цього алгоритму це не буде великою проблемою оскільки він може також використовувати ще 192 і 256 біт.

Для роботи даного алгоритму потрібно використовувати обов'язково дві таблиці, що мають назву s_box, r_con, ці таблиці є статичними.

Також для даного алгоритму обовязковими є 4 операції:

1. subBytes()
2. shiftRows()
3. mixColumns()
4. xorRoundKey()

Для написання даного алгоритму я використав мову програмування Java. Оскільки ця мова повністю побудована на ООП, вся розробка і пояснення буде використовувати саме стиль ООП.

Для роботи алгоритму був розблений клас Crypto, який мав два публічних методи encrypt, decrypt. А також клас Main, який є базовим для роботи програму, саме тут опрацьовується публічні методи обєкта crypto.

```
Crypto(int bit) {
   lengthOfKey = bit / 8;
   if (bit == 128) numberOfRounds = 9;
   if (bit == 192) numberOfRounds = 11;
   if (bit == 256) numberOfRounds = 13;
}
```

При створенні об'єкта crypto ми задаєм кількість бітів, що в свою чергу дає можливість визначити необхідну довжину ключа, а також кількість раундів циклу для шифрування

```
    public boolean verificateLengthOfKey(String key) {
        return key.length() >= lengthOfKey;
    }

    public boolean verificateOfBit(int bit) {
        return bit == 128 || bit == 192 || bit == 256;
    }
```

Щоб було простіше зрозуміти принцип роботи цих двох методів за основу візьмемо метод encrypt і розберем принцип його роботи. Перед тим як почати потрібно запустити два методи verificateOfBit() і verificateLengthOfKey(). Перший потрібен для того щоб перевірити що ми правильно задали бітність системи, тобто чи число співпадає з 128, 192, 256. Другий же потрібен для перевірки довжини ключа, чи він має потрібно довжину. Якщо ж хоч один із цих методів видасть про те що потрібні умови не дотриманні алгоритм не почне працювати. Ці два методи можна вбудувати в методи encypt і decrypt, або ж їх винести і запускати перед запуском.

```
    public String encrypt(String message, String key) {
        int originalLen = message.length();
        int lenOfPaddedMessage = originalLen;

        char[] keyCharArray = createArrayKeyFromString(key);

        if (lenOfPaddedMessage % lengthOfKey != 0) {
                lenOfPaddedMessage = (lenOfPaddedMessage / lengthOfKey + 1) *
lengthOfKey;
        }
```

```
        char[] paddedMessage = new char[lenOfPaddedMessage];
          for(int i = 0; i < lenOfPaddedMessage; i++) {
             if (i >= originalLen) paddedMessage[i] = 0;
             else paddedMessage[i] = message.charAt(i);
          }

          for(int i = 0; i < lenOfPaddedMessage; i += lengthOfKey) {
                char[] data = AES_Encrypt(Arrays.copyOfRange(paddedMessage, i, i +
lengthOfKey), keyCharArray);

             for (int y = 0; y < lengthOfKey; y++) {
                paddedMessage[i + y] = data[y];
             }
         }

          return new String(paddedMessage);
       }
```

Для початку визначається обсяг інформації яку потрібно буде зашифрувати, а також використовуємо метод createArrayKeyFromString() для опрацювання ключа і виведення його у формі масива для можливості його опрацьовувати в подальшому. Наступним кроком буде розбиття усієї інформації на блоки по довжинні самого ключа, яку ми визначили при створенні об’єкта. Це в свою чергу дає можливість сформувати необхідні таблиці для подальшого їх опрацювання в методі AES_Encrypt і сформування його назад в рядок в вивести його завдяки команді return.

```
       private char[] AES_Encrypt(char[] message, char[] key) {
          char[] state = new char[lengthOfKey];
          for (int i = 0; i < lengthOfKey; i++) {
             state[i] = message[i];
```

```
        }

        char[] expandedKey = new char[176];
        expandedKey = KeyExtension(key, expandedKey);

        state = AddRoundKey(state, key);

        for(int i = 0; i < numberOfRounds; i++) {
            state = SubBytes(state);
            state = ShiftRows(state);
            state = MixColumns(state);
            state = AddRoundKey(state, expandedKey);
        }

        state = SubBytes(state);
        state = ShiftRows(state);
        state = AddRoundKey(state, Arrays.copyOfRange(expandedKey, 0, 159));

        for(int i = 0; i < lengthOfKey; i++) {
            message[i] = state[i];
        }

        return message;
    }
```

В змінні message ми отримаємо таблицю, і ключ зі зміни char. Ініцілізуємо state, який є ідентичний таблиці з повідомленням. І отримуємо expandedKey з методу KeyExtension. Пізніше викликаємо метод AddRoundKey() і запускаємо цикл з довжиною яку визначили при створенні обєкта з кількості біт, для того щоб state пропустити через методи: SubBytes(), ShiftRows(), MixColumns(), AddRoundKey(). На завершення ще раз використовуємо SubBytes(), ShiftRows(), AddRoundKey() і

виводимо повідомлення.

```
private char[] KeyExtension(char[] inputKey, char[] expandedKeys) {
    for(int i = 0; i < lengthOfKey; i++) {
        expandedKeys[i] = inputKey[i];
    }

    int bytesGenerated = lengthOfKey;
    int rconIteration = 1;
    char[] temp = new char[4];

    while (bytesGenerated < 176) {
        for (int i = 0; i < 4; i++) {
            temp[i] = expandedKeys[i + bytesGenerated - 4];
        }

        if (bytesGenerated % lengthOfKey == 0) {
            KeyExtensionCore(temp, (char) rconIteration);
            rconIteration++;
        }
    for(int a = 0; a < 4; a++) {
             expandedKeys[bytesGenerated] = (char) (expandedKeys[bytesGenerated -
lengthOfKey] ^ temp[a]);
            bytesGenerated++;
        }
    }

    return expandedKeys;
}
```

Цей метод використовується для створення ключів для кожного раунду.

```
private char[] KeyExtensionCore(char[] in, char i) {
    in[0] = s_box[in[0]];
    in[1] = s_box[in[1]];
    in[2] = s_box[in[2]];
    in[3] = s_box[in[3]];

    in[0] ^= r_con[i];

    return in;
  }
```

Цей метод використовується для заміни елементів ключа з таблиці s_box, r_con.

```
private char[] SubBytes(char[] state) {
  for (int i = 0; i < lengthOfKey; i++) {
    state[i] = s_box[state[i]];
  }

  return state;
}
```

Цей метод потрібен для заміни елементів з таблиці state, на елементи з таблиці s_box.

```
private char[] ShiftRows(char[] state) {
  char[] tmp = new char[lengthOfKey];

  if (lengthOfKey == 16) {
    tmp[0] = state[0];
```

```
            tmp[1] = state[5];
            tmp[2] = state[10];
            tmp[3] = state[15];

            tmp[4] = state[4];
            tmp[5] = state[9];
            tmp[6] = state[14];
            tmp[7] = state[3];

            tmp[8] = state[8];
            tmp[9] = state[13];
            tmp[10] = state[2];
            tmp[11] = state[7];

            tmp[12] = state[12];
            tmp[13] = state[1];
            tmp[14] = state[6];
            tmp[15] = state[11];
        } else if (lengthOfKey == 24) {
            tmp[0] = state[0];
            tmp[1] = state[7];
            tmp[2] = state[14];
            tmp[3] = state[21];
            tmp[4] = state[4];
            tmp[5] = state[11];

            tmp[6] = state[6];
            tmp[7] = state[13];
            tmp[8] = state[20];
```

```
            tmp[9] = state[3];
            tmp[10] = state[10];
            tmp[11] = state[17];

            tmp[12] = state[12];
            tmp[13] = state[19];
            tmp[14] = state[2];
            tmp[15] = state[9];
            tmp[16] = state[16];
            tmp[17] = state[23];

            tmp[18] = state[18];
            tmp[19] = state[1];
            tmp[20] = state[8];
            tmp[21] = state[15];
            tmp[22] = state[22];
            tmp[23] = state[5];
        } else if (lengthOfKey == 32) {
            tmp[0] = state[0];
            tmp[1] = state[9];
            tmp[2] = state[18];
            tmp[3] = state[27];
            tmp[4] = state[4];
            tmp[5] = state[13];
            tmp[6] = state[22];
            tmp[7] = state[31];

            tmp[8] = state[8];
            tmp[9] = state[17];
            tmp[10] = state[26];
```

```
            tmp[11] = state[3];
            tmp[12] = state[12];
            tmp[13] = state[21];
            tmp[14] = state[30];
            tmp[15] = state[7];

            tmp[16] = state[16];
            tmp[17] = state[25];
            tmp[18] = state[2];
            tmp[19] = state[11];
            tmp[20] = state[20];
            tmp[21] = state[29];
            tmp[22] = state[6];
            tmp[23] = state[15];

            tmp[24] = state[24];
            tmp[25] = state[1];
            tmp[26] = state[10];
            tmp[27] = state[19];
            tmp[28] = state[28];
            tmp[29] = state[5];
            tmp[30] = state[14];
            tmp[31] = state[23];
        }

        for (int i = 0; i < lengthOfKey; i++) {
            state[i] = tmp[i];
        }

        return state;
```

```
    }
```

Цей метод потрібен для зсуву елементів таблиці. Перший рядок не зсувається. Другий рядок зсувається на одну позицію в ліво. Третій рядок зсувається на дві позиції в ліво. Четвертий рядок зсувається на три позиції в ліво. Пятий рядок не зсувається. Шостий рядок зсувається на одну позицію в ліво. Сьомий рядок зсувається на дві позиції в ліво. Восьмий рядок зсувається на три позиції в ліво.Але в залежності від розміру таблиці, цих рядків може бути 4 у випадку з 128 бітами, 6 у випадку з 192 і 8 у випадку з 256 рядків.

```
    private char[] MixColumns(char[] state) {
        char[] tmp = new char[lengthOfKey];

        for (int i = 0; i < lengthOfKey; i += 4) {
            tmp[i] = (char) (mul2[state[i]] ^ mul3[state[i + 1]] ^ state[i + 2] ^ state[i +
3]);
            tmp[i + 1] = (char) (state[i] ^ mul2[state[i + 1]] ^ mul3[state[i + 2]] ^ state[i
+ 3]);
            tmp[i + 2] = (char) (state[i] ^ state[i + 1] ^ mul2[state[i + 2]] ^ mul3[state[i +
3]]);
            tmp[i + 3] = (char) (mul3[state[i]] ^ state[i + 1] ^ state[i + 2] ^ mul2[state[i +
3]]);
        }

        for(int i = 0; i < lengthOfKey; i++) {
            state[i] = tmp[i];
        }

        return state;
    }
```

Даний метод додає state і ключ завдяки операції XOR.

```
private char[] AddRoundKey(char[] state, char[] roundKey) {
   for (int i = 0; i < lengthOfKey; i++) {
      state[i] ^= roundKey[i];
   }

   return state;
}
```

4.3. Результатом шифрування і дешифрування даних різних типів і розмірів

Для 128

Message: This is a message we will encrypte with AES.

Encrypted message: EYívËû§·¿XÕ÷ µÁíµ¾ÌèF¯õÁLRcTùá.

Decrypted message: This is a message we will encrypte with AES.

Для 192

Message: This is a message we will encrypte with AES.

Encrypted message: <=ùðGÆV!>¥üÏCµ¬z$l»²Â-Þ2Ìø]_sä¦ÓR+x

Decrypted message: This is a message we will encrypte with AES.

Для 256

Message: This is a message we will encrypte with AES.

Encrypted message: ÉOYÌÝÚÇ4ÖìÁ< Ò¿;÷Ãt®Ã_C-Vh×;æBTëÂòW/Îj? F¢ªT=:ðú

Decrypted message: This is a message we will encrypte with AES.

Для 250

Key have to equal 128, 192 or 256

Для ключа PasswordPasswor

Key have to length of more than 16 symbol

4.4. Реалізація блоків зсуву на Verilog

Реалізація блоків зсуву на Verilog

```
// AES-128 Encryption in ARM64 assembly
// 352 bytes

    .arch armv8-a
    .text

    .global E
```

Метод - Multiplication over GF(2**8)

```
M:
    and      w10, w14, 0x80808080
    mov      w12, 27
    lsr      w8, w10, 7
    mul      w8, w8, w12
    eor      w10, w14, w10
    eor      w10, w8, w10, lsl 1
    ret
```

Метод - SubByte(B x);

```
S:
    str      lr, [sp, -16]!
    ands     w7, w13, 0xFF
    beq      SB2

    mov      w14, 1
    mov      w15, 1
    mov      x3, 0xFF
```

```
SB0:
    cmp     w15, 1
    ccmp    w14, w7, 0, eq
    csel    w14, w15, w14, eq
    csel    w15, wzr, w15, eq
    bl      M
    eor     w14, w14, w10
    subs    x3, x3, 1
    bne     SB0

    and     w7, w14, 0xFF
    mov     x3, 4
SB1:
    lsr     w10, w14, 7
    orr     w14, w10, w14, lsl 1
    eor     w7, w7, w14
    subs    x3, x3, 1
    bne     SB1
SB2:
    mov     w10, 99
    eor     w7, w7, w10
    bfxil   w13, w7, 0, 8
    ldr     lr, [sp], 16
    ret
```

Метод - void E(void *s);

```
E:
    str     lr, [sp, -16]!
    sub     sp, sp, 32

    // copy plain text + master key to x
    // F(8)x[i]=((W*)s)[i];
    ldp     x5, x6, [x0]
    ldp     x7, x8, [x0, 16]
```

```
    stp     x5, x6, [sp]
    stp     x7, x8, [sp, 16]

    // c = 1
    mov     w4, 1
L0:
    // AddRoundKey, 1st part of ExpandRoundKey
    // w=k[3];F(4)w=(w&-256)|S(w),w=R(w,8),((W*)s)[i]=x[i]^k[i];
    mov     x2, xzr
    ldr     w13, [sp, 16+3*4]
    add     x1, sp, 16
L1:
    bl      S
    ror     w13, w13, 8
    ldr     w10, [sp, x2, lsl 2]
    ldr     w11, [x1, x2, lsl 2]
    eor     w10, w10, w11
    str     w10, [x0, x2, lsl 2]

    add     x2, x2, 1
    cmp     x2, 4
    bne     L1

    // AddRoundConstant, perform 2nd part of ExpandRoundKey
    // w=R(w,8)^c;F(4)w=k[i]^=w;
    eor     w13, w4, w13, ror 8
L2:
    ldr     w10, [x1]
    eor     w13, w13, w10
    str     w13, [x1], 4

    subs    x2, x2, 1
    bne     L2
```

```
    // if round 11, stop
    // if(c==108)break;
    cmp     w4, 108
    beq     L5
 // update round constant
    // c=M(c);
    mov     w14, w4
    bl      M
    mov     w4, w10

    // SubBytes and ShiftRows
    // F(16)((B*)x)[(i%4)+(((i/4)-(i%4))%4)*4]=S(s[i]);
L3:
    ldrb    w13, [x0, x2]
    bl      S
    and     w10, w2, 3
    lsr     w11, w2, 2
    sub     w11, w11, w10
    and     w11, w11, 3
    add     w10, w10, w11, lsl 2
    strb    w13, [sp, w10, uxtw]

    add     x2, x2, 1
    cmp     x2, 16
    bne     L3

    // if (c != 108)
    cmp     w4, 108
L4:
    beq     L0
    subs    x2, x2, 4

    // MixColumns
    // F(4)w=x[i],x[i]=R(w,8)^R(w,16)^R(w,24)^M(R(w,8)^w);
    ldr     w13, [sp, x2]
```

```
    eor     w14, w13, w13, ror 8
    bl      M
    eor     w14, w10, w13, ror 8
    eor     w14, w14, w13, ror 16
    eor     w14, w14, w13, ror 24
    str     w14, [sp, x2]

    b       L4
L5:
    add     sp, sp, 32
    ldr     lr, [sp], 16
    ret
```

4.5. Реалізація на мікроконтролері ESP-32-WROVER з використанням MicroPython

Для реалізації алгоритму шифрування AES з використанням мікроконтролера ESP-32 на реалізації мови Python - MicroPython. Даний мікроконтролер запрограмовано таким чином, що до нього можна підключитися через HTTP. В результаті чого переходячи по посиланню /crypto ми отримуємо зашифрований текст. Знаючи ключ ми можемо розшифрувати інформацію яку нам надіслав мікроконтролер. Доцільність надсилання зашифрованих даних мікроконтролером, якщо його використовують для систем безпеки приміщення або в інших системах осіб, до яких не повинні отрмати доступ треті особи.

Запропонована реалізація на MicroPython складається з трьох файлів: aes.py, server.py і main.py. Файл server.py містить клас MicroPyServer, що реалізує з мікроконтролера сервер для передачі даних через http. Файл aes.py містить клас Crypto, що реалізує шифрування завдяки алгоритму AES. Файл Main є головним файлом для запуску.

Висновки

Побудовано блок схему що відображає процес шифрування в алгоритмі AES.

Реалізовано алгоритм на мові програмування Java.

Показано особливості реалізації операцій шифрування на Verilog.

Реалізованно алгоритм на мікроконтролері AES з використанням MicroPython.

5. ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

Загальна вартість розробки універсального алгоритму шифрування AES, а також реалізація його на різних мова програмування не потребує фінансових вкладень. Але реалізація його на мікроконтролері становитиме вартості самого мікроконтролера. Для виконання кваліфікаційної роботи мною була використано мікроконтролер ESP-32-WROVER вартість якого становить 7 доларів. Для зчитування даних з мікроконтролера використовувався протокол HTTP. При потребі використовувати його з метою передачі секретних даних завдяки мікроконтролеру, вимогою перед мікроконтрлером буде наявність WI-FI інтерфейсу.

Також варто зазначити що даний алгоритм шифрування можна використовувати не тільки для мов програмування, але і у скріптових мовах та на апаратному рівні. Що дає можливість його застосовувати на всіх потрібних рівнях і системах.

Таблиця 5.1. Вартість комплектуючих

Найменування	Кількість штук	Ціна за од., $	Сума
ESP - 32	1	7$	7$

СПИСОК ЛІТЕРАТУРИ

1. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Криптографія
2. Шифрування: типи і алгоритми [Електронний ресурс]: https://wiki.hostpro.ua/ua/knowledgebase/shifruvannja-tipi-i-algoritmi/.
3. Data Encryption Standard [Електронний ресурс]: https://uk.wikipedia.org/wiki/Data_Encryption_Standard
4. ESP32 [Електронний ресурс]: https://ru.wikipedia.org/wiki/ESP32
5. Стенографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Стенографія
6. Стенографія [Електронний ресурс]: https://uk.campwaltblog.com/4149562-shorthand-what-is-it-the-meaning-of-the-word-quotshorthandquot-stenographerist-who-is-she
7. AES [Електронний ресурс]: https://ru.wikipedia.org/wiki/AES_(стандарт_шифрования)
8. Шифрування з симетричними ключами [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифрування_з_симетричними_ключами
9. Шифрування з асиметрисними ключами [Електронний ресурс]: https://uk.wikipedia.org/wiki/Асиметричні_алгоритми_шифрування.
10. Протокол Діффі - Гелмана [Електронний ресурс]: https://uk.wikipedia.org/wiki/Протокол_Діффі_—_Геллмана
11. Шифр [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифр
12. Шифрування [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифрування
13. Потоковий шифр [Електронний ресурс]: https://uk.wikipedia.org/wiki/Потоковий_шифр
14. Асиметричні алгоритми шифрування [Електронний ресурс]: https://uk.wikipedia.org/wiki/Асиметричні_алгоритми_шифрування
15. Шифрування з симетричними ключами [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифрування_з_симетричними_ключами
16. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Моделі_атаки_(криптоаналіз)
17. Криптографія [Електронний ресурс]: http://fit.univ.kiev.ua/archives/6154

1. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Підстановочний_шифр
2. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Біграмний_шифр
3. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифр_Плейфера
4. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифр_Плейфера
5. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Мікроконтролер
6. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/ESP32
7. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Система_на_кристалі
8. Криптографія [Електронний ресурс]: https://zakon.rada.gov.ua/laws/show/2657-12#Text
9. Криптографія [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифр_Цезаря
10. Криптографія [Електронний ресурс]: https://d-learn.pnu.edu.ua/data/users/4808/zikm/tema1_bazovi_ponattya_ZI.pdf
11. Криптографія [Електронний ресурс]: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80#Text
12. Криптографія [Електронний ресурс]: https://ru.wikipedia.org/wiki/Шифр_Виженера
13. Криптографія [Електронний ресурс]: https://ru.wikipedia.org/wiki/S-блок_(информатика)
14. Криптографія [Електронний ресурс]: https://ru.wikipedia.org/wiki/Перестановочный_шифр
15. Шифрування з симетричними ключами [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифрування_з_симетричними_ключами
16. Державна таємниця [Електронний ресурс]: https://uk.wikipedia.org/wiki/Державна_таємниця

1. DESX [Електронний ресурс]: https://ru.wikipedia.org/wiki/DESX
2. Поле Галуа [Електронний ресурс]: https://uk.wikipedia.org/wiki/Поле_Галуа
3. AES [Електронний ресурс]: https://habr.com/ru/post/497672/
4. Незвідний многочлен [Електронний ресурс]: https://uk.wikipedia.org/wiki/Незвідний_многочлен
5. Алгоритм Рабіна [Електронний ресурс]: https://d-learn.pnu.edu.ua/data/users/4808/zikm/ЛАб_роб/Лабораторна%20робота%20N10%20ДОСЛІДЖЕННЯ%20КРИПТОАЛГОРИТМУ%20ШИФРУВАННЯ%20РАБІНА.pdf
6. RSA, Ель Гамаля [Електронний ресурс]: https://d-learn.pnu.edu.ua/data/users/4808/zikm/Методичні%20рекомендації%20до%20лабораторних%20робіт%20з%20курсу%20Захист%20інформації%20в%20компютерних%20системах.pdf
7. IDEA [Електронний ресурс]: https://uk.wikipedia.org/wiki/IDEA_(%D1%88%D0%B8%D1%84%D1%80)
8. AES [Електронний ресурс]: https://bit.nmu.org.ua/ua/student/metod/cryptology/лекция%209.pdf
9. Криптосистема Рабіна [Електронний ресурс]: https://uk.wikipedia.org/wiki/Криптосистема_Рабіна
10. RSA [Електронний ресурс]: https://uk.wikipedia.org/wiki/RSA
11. Схема Ель - Гамаля [Електронний ресурс]: https://uk.wikipedia.org/wiki/Схема_Ель-Гамаля
12. AES-128 [Електронний ресурс]: https://modexp.wordpress.com/2018/10/30/arm64-assembly/#aes
13. Micropython [Електронний ресурс]: https://micropython.org

ВИСНОВКИ

Досліджено що таке криптографія і для чого вона потрібна.

Розглянуто законодавство України про інформацію

Наведено приклади алгоритмів шифрування і криптосистем, а також слабкі місця перед криптоаналізом.

Розроблено універсальний псевдокод для алгоритму шифрування AES який підійде до більшості мов програмування

Розроблено блок схему, код, а також результати виконання алгоритму AES реалізованого на мові програмування Java.

Реалізовано алгоритму AES на мікроконтролері ESP-32 з використанням MicroPython.

ДОДАТОК 1

Реалізація AES на мові програмування Java

```
public class Crypto {


   private final int lengthOfKey;
   private int numberOfRounds;


   Crypto(int bit) {
      lengthOfKey = bit / 8;
      if (bit == 128) numberOfRounds = 9;
      if (bit == 192) numberOfRounds = 11;
      if (bit == 256) numberOfRounds = 13;
   }


   private final char[] s_box = {
         0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76,
         0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0,
         0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15,
         0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75,
         0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
```

```
0xb3, 0x29, 0xe3, 0x2f, 0x84,

    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf,

    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8,

    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2,

    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73,

    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb,

    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79,

    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08,

    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a,

    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e,

    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf,

    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16,

  };
```

```
private final char[] mul2 = {
        0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14,
0x16, 0x18, 0x1a, 0x1c, 0x1e,
        0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34,
0x36, 0x38, 0x3a, 0x3c, 0x3e,
        0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54,
0x56, 0x58, 0x5a, 0x5c, 0x5e,
        0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74,
0x76, 0x78, 0x7a, 0x7c, 0x7e,
        0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94,
0x96, 0x98, 0x9a, 0x9c, 0x9e,
        0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4,
0xb6, 0xb8, 0xba, 0xbc, 0xbe,
        0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4,
0xd6, 0xd8, 0xda, 0xdc, 0xde,
        0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4,
0xf6, 0xf8, 0xfa, 0xfc, 0xfe,
        0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f,
0x0d, 0x03, 0x01, 0x07, 0x05,
        0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f,
0x2d, 0x23, 0x21, 0x27, 0x25,
        0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f,
0x4d, 0x43, 0x41, 0x47, 0x45,
        0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f,
0x6d, 0x63, 0x61, 0x67, 0x65,
```

```
        0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f,
0x8d, 0x83, 0x81, 0x87, 0x85,

        0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf,
0xad, 0xa3, 0xa1, 0xa7, 0xa5,

        0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf,
0xcd, 0xc3, 0xc1, 0xc7, 0xc5,

        0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed,
0xe3, 0xe1, 0xe7, 0xe5
    };


    private final char[] mul3 = {
        0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e,
0x1d, 0x14, 0x17, 0x12, 0x11,

        0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e,
0x2d, 0x24, 0x27, 0x22, 0x21,

        0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e,
0x7d, 0x74, 0x77, 0x72, 0x71,

        0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e,
0x4d, 0x44, 0x47, 0x42, 0x41,

        0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde,
0xdd, 0xd4, 0xd7, 0xd2, 0xd1,

        0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed,
0xe4, 0xe7, 0xe2, 0xe1,

        0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe,
0xbd, 0xb4, 0xb7, 0xb2, 0xb1,
```

```
        0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e,
0x8d, 0x84, 0x87, 0x82, 0x81,

        0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85,
0x86, 0x8f, 0x8c, 0x89, 0x8a,

        0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5,
0xb6, 0xbf, 0xbc, 0xb9, 0xba,

        0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6,
0xef, 0xec, 0xe9, 0xea,

        0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5,
0xd6, 0xdf, 0xdc, 0xd9, 0xda,

        0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45,
0x46, 0x4f, 0x4c, 0x49, 0x4a,

        0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75,
0x76, 0x7f, 0x7c, 0x79, 0x7a,

        0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25,
0x26, 0x2f, 0x2c, 0x29, 0x2a,

        0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15,
0x16, 0x1f, 0x1c, 0x19, 0x1a
    };


    private final char[] mul9 = {
        0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a,
0x53, 0x6c, 0x65, 0x7e, 0x77,

        0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xa6, 0xaf, 0xd8, 0xd1, 0xca,
0xc3, 0xfc, 0xf5, 0xee, 0xe7,
```

0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,

0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1, 0xf8, 0xc7, 0xce, 0xd5, 0xdc,

0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c, 0x25, 0x1a, 0x13, 0x08, 0x01,

0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5, 0x8a, 0x83, 0x98, 0x91,

0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17, 0x1e, 0x21, 0x28, 0x33, 0x3a,

0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87, 0x8e, 0xb1, 0xb8, 0xa3, 0xaa,

0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf, 0x80, 0x89, 0x92, 0x9b,

0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26, 0x2f, 0x10, 0x19, 0x02, 0x0b,

0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d, 0x84, 0xbb, 0xb2, 0xa9, 0xa0,

0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d, 0x14, 0x2b, 0x22, 0x39, 0x30,

0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0, 0xc9, 0xf6, 0xff, 0xe4, 0xed,

0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50, 0x59, 0x66, 0x6f, 0x74, 0x7d,

0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb, 0xf2, 0xcd, 0xc4, 0xdf, 0xd6,

```
        0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b,
0x62, 0x5d, 0x54, 0x4f, 0x46
    };


    private final char[] r_con = {
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
0x6c, 0xd8, 0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
0xfa, 0xef, 0xc5, 0x91, 0x39,
        0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc, 0x83, 0x1d, 0x3a,
        0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
0x80, 0x1b, 0x36, 0x6c, 0xd8,
        0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
0xd4, 0xb3, 0x7d, 0xfa, 0xef,
        0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
0x4a, 0x94, 0x33, 0x66, 0xcc,
        0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,
0x10, 0x20, 0x40, 0x80, 0x1b,
        0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35, 0x6a, 0xd4, 0xb3,
        0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
0xc2, 0x9f, 0x25, 0x4a, 0x94,
        0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
0x02, 0x04, 0x08, 0x10, 0x20,
```

```
        0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
0xbc, 0x63, 0xc6, 0x97, 0x35,

        0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,
0xd3, 0xbd, 0x61, 0xc2, 0x9f,

        0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
0xcb, 0x8d, 0x01, 0x02, 0x04,

        0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
0x9a, 0x2f, 0x5e, 0xbc, 0x63,

        0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,
0x39, 0x72, 0xe4, 0xd3, 0xbd,

        0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
0x3a, 0x74, 0xe8, 0xcb, 0x8d
    };


  private char[] KeyExtensionCore(char[] in, char i) {
        in[0] = s_box[in[0]];
        in[1] = s_box[in[1]];
        in[2] = s_box[in[2]];
        in[3] = s_box[in[3]];


        in[0] ^= r_con[i];


        return in;
    }
```

```
    private char[] KeyExtension(char[] inputKey, char[] expandedKeys) {
        for(int i = 0; i < lengthOfKey; i++) {
            expandedKeys[i] = inputKey[i];
        }

        int bytesGenerated = lengthOfKey;
        int rconIteration = 1;
        char[] temp = new char[4];

        while (bytesGenerated < 176) {
            for (int i = 0; i < 4; i++) {
                temp[i] = expandedKeys[i + bytesGenerated - 4];
            }
            if (bytesGenerated % lengthOfKey == 0) {
                KeyExtensionCore(temp, (char) rconIteration);
                rconIteration++;
            }

            for(int a = 0; a < 4; a++) {
                expandedKeys[bytesGenerated] = (char)
(expandedKeys[bytesGenerated - lengthOfKey] ^ temp[a]);
                bytesGenerated++;
            }
```

```
    }

    return expandedKeys;
}

private char[] SubBytes(char[] state) {
    for (int i = 0; i < lengthOfKey; i++) {
        state[i] = s_box[state[i]];
    }

    return state;
}

private char[] DecryptSubBytes(char[] state) {
    for (int i = 0; i < 277181; i++) {
        SubBytes(state);
    }

    return state;
}

private char[] ShiftRows(char[] state) {
```

```
char[] tmp = new char[lengthOfKey];


if (lengthOfKey == 16) {
    tmp[0] = state[0];
    tmp[1] = state[5];
    tmp[2] = state[10];
    tmp[3] = state[15];

    tmp[4] = state[4];
    tmp[5] = state[9];
    tmp[6] = state[14];
    tmp[7] = state[3];

    tmp[8] = state[8];
    tmp[9] = state[13];
    tmp[10] = state[2];
    tmp[11] = state[7];
    tmp[12] = state[12];
    tmp[13] = state[1];
    tmp[14] = state[6];
    tmp[15] = state[11];
} else if (lengthOfKey == 24) {
```

```
tmp[0] = state[0];
tmp[1] = state[7];
tmp[2] = state[14];
tmp[3] = state[21];
tmp[4] = state[4];
tmp[5] = state[11];

tmp[6] = state[6];
tmp[7] = state[13];
tmp[8] = state[20];
tmp[9] = state[3];
tmp[10] = state[10];
tmp[11] = state[17];

tmp[12] = state[12];
tmp[13] = state[19];
tmp[14] = state[2];
tmp[15] = state[9];
tmp[16] = state[16];
tmp[17] = state[23];
tmp[18] = state[18];
tmp[19] = state[1];
```

```
        tmp[20] = state[8];
        tmp[21] = state[15];
        tmp[22] = state[22];
        tmp[23] = state[5];
    } else if (lengthOfKey == 32) {
        tmp[0] = state[0];
        tmp[1] = state[9];
        tmp[2] = state[18];
        tmp[3] = state[27];
        tmp[4] = state[4];
        tmp[5] = state[13];
        tmp[6] = state[22];
        tmp[7] = state[31];

        tmp[8] = state[8];
        tmp[9] = state[17];
        tmp[10] = state[26];
        tmp[11] = state[3];
        tmp[12] = state[12];
        tmp[13] = state[21];
        tmp[14] = state[30];
        tmp[15] = state[7];
```

```
        tmp[16] = state[16];
        tmp[17] = state[25];
        tmp[18] = state[2];
        tmp[19] = state[11];
        tmp[20] = state[20];
        tmp[21] = state[29];
        tmp[22] = state[6];
        tmp[23] = state[15];

        tmp[24] = state[24];
        tmp[25] = state[1];
        tmp[26] = state[10];
        tmp[27] = state[19];
        tmp[28] = state[28];
        tmp[29] = state[5];
        tmp[30] = state[14];
        tmp[31] = state[23];
    }


    for (int i = 0; i < lengthOfKey; i++) {
        state[i] = tmp[i];
```

```
        }

        return state;
    }

    private char[] DecryptShiftRows(char[] state) {
        for (int i = 0; i < 3; i++) {
            ShiftRows(state);
        }

        return state;
    }

    private char[] MixColumns(char[] state) {
        char[] tmp = new char[lengthOfKey];

        for (int i = 0; i < lengthOfKey; i += 4) {
            tmp[i] = (char) (mul2[state[i]] ^ mul3[state[i + 1]] ^ state[i + 2] ^
state[i + 3]);
            tmp[i + 1] = (char) (state[i] ^ mul2[state[i + 1]] ^ mul3[state[i + 2]] ^
state[i + 3]);
            tmp[i + 2] = (char) (state[i] ^ state[i + 1] ^ mul2[state[i + 2]] ^
mul3[state[i + 3]]);
```

```
            tmp[i + 3] = (char) (mul3[state[i]] ^ state[i + 1] ^ state[i + 2] ^
mul2[state[i + 3]]);
        }


        for(int i = 0; i < lengthOfKey; i++) {
            state[i] = tmp[i];
        }


        return state;
    }


    private char[] DecryptMixColumns(char[] state) {
        for (int i = 0; i < 3; i++) {
            MixColumns(state);
        }


        return state;
    }


    private char[] AddRoundKey(char[] state, char[] roundKey) {
        for (int i = 0; i < lengthOfKey; i++) {
            state[i] ^= roundKey[i];
        }
```

```
    return state;
}

private char[] DecryptAddRoundKey(char[] state, char[] roundKey) {
    return AddRoundKey(state, roundKey);
}

private char[] AES_Encrypt(char[] message, char[] key) {
    char[] state = new char[lengthOfKey];
    for (int i = 0; i < lengthOfKey; i++) {
        state[i] = message[i];
    }

    char[] expandedKey = new char[176];
    expandedKey = KeyExtension(key, expandedKey);

    state = AddRoundKey(state, key);

    for(int i = 0; i < numberOfRounds; i++) {
        state = SubBytes(state);
        state = ShiftRows(state);
```

```
            state = MixColumns(state);
            state = AddRoundKey(state, expandedKey);
        }

        state = SubBytes(state);
        state = ShiftRows(state);
        state = AddRoundKey(state, Arrays.copyOfRange(expandedKey, 0,
159));

        for(int i = 0; i < lengthOfKey; i++) {
            message[i] = state[i];
        }

        return message;
    }

    private char[] AES_Decrypt(char[] message, char[] key) {
        char[] state = new char[lengthOfKey];
        for (int i = 0; i < lengthOfKey; i++) {
            state[i] = message[i];
        }

        char[] expandedKey = new char[176];
```

```
        expandedKey = KeyExtension(key, expandedKey);


        state = DecryptAddRoundKey(state,
Arrays.copyOfRange(expandedKey, 0, 159));
        state = DecryptShiftRows(state);
        state = DecryptSubBytes(state);


        for(int i = 0; i < numberOfRounds; i++) {
            state = DecryptAddRoundKey(state, expandedKey);
            state = DecryptMixColumns(state);
            state = DecryptShiftRows(state);
            state = DecryptSubBytes(state);
        }


        state = DecryptAddRoundKey(state, key);


        for(int i = 0; i < lengthOfKey; i++) {
            message[i] = state[i];
        }


        return message;
    }
```

```
private char[] createArrayKeyFromString(String key) {
    char[] keyCharArray = new char[key.length()];
    for (int i = 0; i < lengthOfKey; i++) {
        keyCharArray[i] = key.charAt(i);
    }

    return keyCharArray;
}

public boolean verificateLengthOfKey(String key) {
    return key.length() >= lengthOfKey;
}

public boolean verificateOfBit(int bit) {
    return bit == 128 || bit == 192 || bit == 256;
}


public String encrypt(String message, String key) {
    int originalLen = message.length();
    int lenOfPaddedMessage = originalLen;
```

```
        char[] keyCharArray = createArrayKeyFromString(key);


        if (lenOfPaddedMessage % lengthOfKey != 0) {
            lenOfPaddedMessage = (lenOfPaddedMessage / lengthOfKey + 1)
* lengthOfKey;
        }


        char[] paddedMessage = new char[lenOfPaddedMessage];
        for(int i = 0; i < lenOfPaddedMessage; i++) {
            if (i >= originalLen) paddedMessage[i] = 0;
            else paddedMessage[i] = message.charAt(i);
        }


        for(int i = 0; i < lenOfPaddedMessage; i += lengthOfKey) {
            char[] data = AES_Encrypt(Arrays.copyOfRange(paddedMessage,
i, i + lengthOfKey), keyCharArray);


            for (int y = 0; y < lengthOfKey; y++) {
                paddedMessage[i + y] = data[y];
            }
        }


        return new String(paddedMessage);
```

```
    }

    public String decrypt(String encryptText, String key) {
        int originalLen = encryptText.length();
        int lenOfPaddedMessage = originalLen;

        char[] keyCharArray = createArrayKeyFromString(key);

        if (lenOfPaddedMessage % lengthOfKey != 0) {
            lenOfPaddedMessage = (lenOfPaddedMessage / lengthOfKey + 1)
* lengthOfKey;
        }

        char[] paddedMessage = new char[lenOfPaddedMessage];
        for(int i = 0; i < lenOfPaddedMessage; i++) {
            if (i >= originalLen) paddedMessage[i] = 0;
            else paddedMessage[i] = encryptText.charAt(i);
        }

        for(int i = 0; i < lenOfPaddedMessage; i += lengthOfKey) {
            char[] data = AES_Decrypt(Arrays.copyOfRange(paddedMessage,
i, i + lengthOfKey), keyCharArray);
```

```
            for (int y = 0; y < lengthOfKey; y++) {
                paddedMessage[i + y] = data[y];
            }
        }

        return new String(paddedMessage);
    }

}
```

ДОДАТОК 2

Реалізація AES на мові програмування Python з реалізацією на MicroPython

Файл - aes.py

```
class Crypto:
    __s_box = [
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
0x9c, 0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1,
0x71, 0xd8, 0x31, 0x15,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
0x50, 0x3c, 0x9f, 0xa8,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2,
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb,
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79,
        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08,
        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e,
        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
```

```
0xe9, 0xce, 0x55, 0x28, 0xdf,
      0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16,
   ]

   __mul2 = [
      0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14,
0x16, 0x18, 0x1a, 0x1c, 0x1e,
      0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34,
0x36, 0x38, 0x3a, 0x3c, 0x3e,
      0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54,
0x56, 0x58, 0x5a, 0x5c, 0x5e,
      0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74,
0x76, 0x78, 0x7a, 0x7c, 0x7e,
      0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94,
0x96, 0x98, 0x9a, 0x9c, 0x9e,
      0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4,
0xb6, 0xb8, 0xba, 0xbc, 0xbe,
      0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4,
0xd6, 0xd8, 0xda, 0xdc, 0xde,
      0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6,
0xf8, 0xfa, 0xfc, 0xfe,
      0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f,
0x0d, 0x03, 0x01, 0x07, 0x05,
      0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f,
0x2d, 0x23, 0x21, 0x27, 0x25,
      0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f,
0x4d, 0x43, 0x41, 0x47, 0x45,
      0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f,
0x6d, 0x63, 0x61, 0x67, 0x65,
      0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f,
0x8d, 0x83, 0x81, 0x87, 0x85,
      0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf,
0xad, 0xa3, 0xa1, 0xa7, 0xa5,
```

```
    0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf,
0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
    0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed,
0xe3, 0xe1, 0xe7, 0xe5
  ]

  __mul3 = [
    0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e,
0x1d, 0x14, 0x17, 0x12, 0x11,
    0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e,
0x2d, 0x24, 0x27, 0x22, 0x21,
    0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e,
0x7d, 0x74, 0x77, 0x72, 0x71,
    0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e,
0x4d, 0x44, 0x47, 0x42, 0x41,
    0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde,
0xdd, 0xd4, 0xd7, 0xd2, 0xd1,
    0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed,
0xe4, 0xe7, 0xe2, 0xe1,
    0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe,
0xbd, 0xb4, 0xb7, 0xb2, 0xb1,
    0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e,
0x8d, 0x84, 0x87, 0x82, 0x81,
    0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85,
0x86, 0x8f, 0x8c, 0x89, 0x8a,
    0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5,
0xb6, 0xbf, 0xbc, 0xb9, 0xba,
    0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6,
0xef, 0xec, 0xe9, 0xea,
    0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5,
0xd6, 0xdf, 0xdc, 0xd9, 0xda,
    0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45,
0x46, 0x4f, 0x4c, 0x49, 0x4a,
    0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75,
```

```
0x76, 0x7f, 0x7c, 0x79, 0x7a,
      0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25,
0x26, 0x2f, 0x2c, 0x29, 0x2a,
      0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15,
0x16, 0x1f, 0x1c, 0x19, 0x1a
   ]

   __mul9 = [
      0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a,
0x53, 0x6c, 0x65, 0x7e, 0x77,
      0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xa6, 0xaf, 0xd8, 0xd1, 0xca,
0xc3, 0xfc, 0xf5, 0xee, 0xe7,
      0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61,
0x68, 0x57, 0x5e, 0x45, 0x4c,
      0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1,
0xf8, 0xc7, 0xce, 0xd5, 0xdc,
      0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c,
0x25, 0x1a, 0x13, 0x08, 0x01,
      0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5,
0x8a, 0x83, 0x98, 0x91,
      0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17,
0x1e, 0x21, 0x28, 0x33, 0x3a,
      0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87,
0x8e, 0xb1, 0xb8, 0xa3, 0xaa,
      0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf,
0x80, 0x89, 0x92, 0x9b,
      0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26,
0x2f, 0x10, 0x19, 0x02, 0x0b,
      0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d,
0x84, 0xbb, 0xb2, 0xa9, 0xa0,
      0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d,
0x14, 0x2b, 0x22, 0x39, 0x30,
      0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0,
0xc9, 0xf6, 0xff, 0xe4, 0xed,
      0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50,
```

```
0x59, 0x66, 0x6f, 0x74, 0x7d,
        0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb,
0xf2, 0xcd, 0xc4, 0xdf, 0xd6,
        0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b,
0x62, 0x5d, 0x54, 0x4f, 0x46
    ]

    __r_con = [
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
0x6c, 0xd8, 0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
0xfa, 0xef, 0xc5, 0x91, 0x39,
        0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc, 0x83, 0x1d, 0x3a,
        0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
0x80, 0x1b, 0x36, 0x6c, 0xd8,
        0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
0xd4, 0xb3, 0x7d, 0xfa, 0xef,
        0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
0x4a, 0x94, 0x33, 0x66, 0xcc,
        0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,
0x10, 0x20, 0x40, 0x80, 0x1b,
        0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35, 0x6a, 0xd4, 0xb3,
        0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
0xc2, 0x9f, 0x25, 0x4a, 0x94,
        0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
0x02, 0x04, 0x08, 0x10, 0x20,
        0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
0xbc, 0x63, 0xc6, 0x97, 0x35,
        0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,
0xd3, 0xbd, 0x61, 0xc2, 0x9f,
        0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
0xcb, 0x8d, 0x01, 0x02, 0x04,
        0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
```

```
0x9a, 0x2f, 0x5e, 0xbc, 0x63,
      0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,
0x39, 0x72, 0xe4, 0xd3, 0xbd,
      0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
0x3a, 0x74, 0xe8, 0xcb, 0x8d
   ]

   def __init__(self, bit):
      self.__lengthOfKey = int(bit / 8)

      if bit == 128:
         self.__numberOfRounds = 9
      if bit == 192:
         self.__numberOfRounds = 11
      if bit == 256:
         self.__numberOfRounds = 13

   def __KeyExtensionCore(self, inArray, i):
      inArray[0] = self.__s_box[inArray[0]]
      inArray[1] = self.__s_box[inArray[1]]
      inArray[2] = self.__s_box[inArray[2]]
      inArray[3] = self.__s_box[inArray[3]]

      inArray[0] ^= self.__r_con[i]

      return inArray

   def __KeyExtension(self, inputKey, expandedKeys):
      for i in range(self.__lengthOfKey):
         expandedKeys[i] = inputKey[i]

      bytesGenerated = self.__lengthOfKey
      rconIteration = 1
      temp = [0] * 4
```

```
        while bytesGenerated < 176:
            for i in range(4):
                temp[i] = expandedKeys[i + bytesGenerated - 4]

            if bytesGenerated % self.__lengthOfKey == 0:
                temp = self.__KeyExtensionCore(temp, rconIteration)
                rconIteration += 1

            for a in range(4):
                expandedKeys[bytesGenerated] =
expandedKeys[bytesGenerated - self.__lengthOfKey] ^ temp[a]
                bytesGenerated += 1

        return expandedKeys

    def __SubBytes(self, state):
        for i in range(self.__lengthOfKey):
            state[i] = self.__s_box[state[i]]

        return state

    def __DecryptSubBytes(self, state):
        for i in range(277181):
            self.__SubBytes(state)

        return state

    def __ShiftRows(self, state):
        tmp = [0] * self.__lengthOfKey

        if self.__lengthOfKey == 16:
            tmp[0] = state[0]
            tmp[1] = state[5]
            tmp[2] = state[10]
            tmp[3] = state[15]
```

```
        tmp[4] = state[4]
        tmp[5] = state[9]
        tmp[6] = state[14]
        tmp[7] = state[3]

        tmp[8] = state[8]
        tmp[9] = state[13]
        tmp[10] = state[2]
        tmp[11] = state[7]

        tmp[12] = state[12]
        tmp[13] = state[1]
        tmp[14] = state[6]
        tmp[15] = state[11]
    elif self.__lengthOfKey == 24:
        tmp[0] = state[0]
        tmp[1] = state[7]
        tmp[2] = state[14]
        tmp[3] = state[21]
        tmp[4] = state[4]
        tmp[5] = state[11]

        tmp[6] = state[6]
        tmp[7] = state[13]
        tmp[8] = state[20]
        tmp[9] = state[3]
        tmp[10] = state[10]
        tmp[11] = state[17]

        tmp[12] = state[12]
        tmp[13] = state[19]
        tmp[14] = state[2]
        tmp[15] = state[9]
        tmp[16] = state[16]
        tmp[17] = state[23]
```

```
        tmp[18] = state[18]
        tmp[19] = state[1]
        tmp[20] = state[8]
        tmp[21] = state[15]
        tmp[22] = state[22]
        tmp[23] = state[5]
    elif self.__lengthOfKey == 32:
        tmp[0] = state[0]
        tmp[1] = state[9]
        tmp[2] = state[18]
        tmp[3] = state[27]
        tmp[4] = state[4]
        tmp[5] = state[13]
        tmp[6] = state[22]
        tmp[7] = state[31]

        tmp[8] = state[8]
        tmp[9] = state[17]
        tmp[10] = state[26]
        tmp[11] = state[3]
        tmp[12] = state[12]
        tmp[13] = state[21]
        tmp[14] = state[30]
        tmp[15] = state[7]

        tmp[16] = state[16]
        tmp[17] = state[25]
        tmp[18] = state[2]
        tmp[19] = state[11]
        tmp[20] = state[20]
        tmp[21] = state[29]
        tmp[22] = state[6]
        tmp[23] = state[15]
```

```
        tmp[24] = state[24]
        tmp[25] = state[1]
        tmp[26] = state[10]
        tmp[27] = state[19]
        tmp[28] = state[28]
        tmp[29] = state[5]
        tmp[30] = state[14]
        tmp[31] = state[23]

    for i in range(self.__lengthOfKey):
        state[i] = tmp[i]

    return state

  def __DecryptShiftRows(self, state):
    for i in range(3):
        state = self.__ShiftRows(state)

    return state

  def __MixColumns(self, state):
    tmp = [0] * self.__lengthOfKey

    for i in range(0, self.__lengthOfKey, 4):
        tmp[i] = self.__mul2[state[i]] ^ self.__mul3[state[i + 1]] ^ state[i + 2]
^ state[i + 3]
        tmp[i + 1] = state[i] ^ self.__mul2[state[i + 1]] ^ self.__mul3[state[i +
2]] ^ state[i + 3]
        tmp[i + 2] = state[i] ^ state[i + 1] ^ self.__mul2[state[i + 2]] ^
self.__mul3[state[i + 3]]
        tmp[i + 3] = self.__mul3[state[i]] ^ state[i + 1] ^ state[i + 2] ^
self.__mul2[state[i + 3]]

    for i in range(self.__lengthOfKey):
        state[i] = tmp[i]
```

```
        return state

    def __DecryptMixColumns(self, state):
        for i in range(3):
            state = self.__MixColumns(state)

        return state

    def __AddRoundKey(self, state, roundKey):
        for i in range(self.__lengthOfKey):
            state[i] ^= roundKey[i]

        return state

    def __DecryptAddRoundKey(self, state, roundKey):
        return self.__AddRoundKey(state, roundKey)

    def __AES_Encrypt(self, message, key):
        state = [0] * self.__lengthOfKey

        for i in range(self.__lengthOfKey):
            state[i] = message[i]

        expandedKey = [0] * 176
        expandedKey = self.__KeyExtension(key, expandedKey)

        state = self.__AddRoundKey(state, key)

        for i in range(self.__numberOfRounds):
            state = self.__SubBytes(state)
            state = self.__ShiftRows(state)
            state = self.__MixColumns(state)
            state = self.__AddRoundKey(state, expandedKey)
```

```
        state = self.__SubBytes(state)
        state = self.__ShiftRows(state)
        state = self.__AddRoundKey(state, expandedKey[0: 159])

        for i in range(self.__lengthOfKey):
            message[i] = state[i]

        return message

    def __AES_Decrypt(self, message, key):
        state = [0] * self.__lengthOfKey

        for i in range(self.__lengthOfKey):
            state[i] = message[i]

        expandedKey = [0] * 176
        expandedKey = self.__KeyExtension(key, expandedKey)

        state = self.__DecryptAddRoundKey(state, expandedKey[0:159])
        state = self.__DecryptShiftRows(state)
        state = self.__DecryptSubBytes(state)

        for i in range(self.__numberOfRounds):
            state = self.__DecryptAddRoundKey(state, expandedKey)
            state = self.__DecryptMixColumns(state)
            state = self.__DecryptShiftRows(state)
            state = self.__DecryptSubBytes(state)

        state = self.__DecryptAddRoundKey(state, key)

        for i in range(self.__lengthOfKey):
            message[i] = state[i]

        return message
```

```
    def __createArrayKeyFromString(self, key):
        keyCharArray = [0] * len(key)

        for i in range(self.__lengthOfKey):
            keyCharArray[i] = ord(key[i])

        return keyCharArray

    def verificateLengthOfKey(self, key):
        return len(key) >= self.__lengthOfKey

    def verificateOfBit(self, bit):
        return bit == 128 or bit == 192 or bit == 256

    def encrypt(self, message, key):
        originalLen = len(message)
        lenOfPaddedMessage = int(originalLen)

        keyCharArray = self.__createArrayKeyFromString(key)

        if lenOfPaddedMessage % self.__lengthOfKey != 0:
            lenOfPaddedMessage = int(int((lenOfPaddedMessage /
self.__lengthOfKey) + 1) * self.__lengthOfKey)

        paddedMessage = [0] * lenOfPaddedMessage
        for i in range(int(lenOfPaddedMessage)):
            if i >= originalLen:
                paddedMessage[i] = 0
            else:
                paddedMessage[i] = ord(message[i])

        for i in range(0, int(lenOfPaddedMessage), self.__lengthOfKey):
            data = self.__AES_Encrypt(paddedMessage[i:i +
self.__lengthOfKey], keyCharArray)
```

```
            for y in range(self.__lengthOfKey):
                paddedMessage[i + y] = data[y]

        string = ""
        for element in paddedMessage:
            string += str(chr(element))

        return string

    def decrypt(self, encryptText, key):
        originalLen = len(encryptText)
        lenOfPaddedMessage = originalLen

        keyCharArray = self.__createArrayKeyFromString(key)
        if lenOfPaddedMessage % self.__lengthOfKey != 0:
            lenOfPaddedMessage = int(int((lenOfPaddedMessage /
self.__lengthOfKey) + 1) * self.__lengthOfKey)

        paddedMessage = [0] * int(lenOfPaddedMessage)

        for i in range(lenOfPaddedMessage):
            if i >= originalLen:
                paddedMessage[i] = 0
            else:
                paddedMessage[i] = ord(encryptText[i])

        for i in range(0, lenOfPaddedMessage, self.__lengthOfKey):
            data = self.__AES_Decrypt(paddedMessage[i:i +
self.__lengthOfKey], keyCharArray)

            for y in range(self.__lengthOfKey):
                paddedMessage[i + y] = data[y]

        string = ""
```

```
        for element in paddedMessage:
            string += str(chr(element))

        return string
```

Файл - server.py

```
import re
import socket
import sys
import io


class MicroPyServer(object):

    def __init__(self, host="0.0.0.0", port=80):
        """ Constructor """
        self._host = host
        self._port = port
        self._routes = []
        self._connect = None
        self._on_request_handler = None

    def start(self):
        """ Start server """
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
        sock.bind((self._host, self._port))
        sock.listen(1)
       while True:
            try:
                self._connect, address = sock.accept()
                request = self._get_request()
```

```
            if len(request) == 0:
                self._connect.close()
                continue
            if self._on_request_handler:
                if not self._on_request_handler(request, address):
                    continue
            route = self.find_route(request)
            if route:
                route["handler"](request)
            else:
                self.not_found()
        except Exception as e:
            self.internal_error(e)
        finally:
            self._connect.close()

    def add_route(self, path, handler, method="GET"):
        """ Add new route  """
        self._routes.append({"path": path, "handler": handler, "method":
method})

    def send(self, response, status=200, content_type="Content-Type:
text/plain", extra_headers=[]):
        """ Send response to client """
        if self._connect is None:
            raise Exception("Can't send response, no connection instance")

        status_message = {200: "OK", 400: "Bad Request", 403: "Forbidden",
404: "Not Found",
                          500: "Internal Server Error"}
        self._connect.sendall("HTTP/1.0 " + str(status) + " " +
status_message[status] + "\r\n")
        self._connect.sendall(content_type + "\r\n")
        for header in extra_headers:
            self._connect.sendall(header + "\r\n")
```

```
        self._connect.sendall("X-Powered-By: MicroPyServer\r\n")
        self._connect.sendall("\r\n")
        self._connect.sendall(response)

    def find_route(self, request):
        """ Find route """
        lines = request.split("\r\n")
        method = re.search("^([A-Z]+)", lines[0]).group(1)
        path = re.search("^[A-Z]+\\s+(/[-a-zA-Z0-9_.]*)", lines[0]).group(1)
        for route in self._routes:
            if method != route["method"]:
                continue
            if path == route["path"]:
                return route
            else:
                match = re.search("^" + route["path"] + "$", path)
                if match:
                    print(method, path, route["path"])
                    return route

    def not_found(self):
        """ Not found action """
        self.send("404", status=404)

    def internal_error(self, error):
        """ Catch error action """
        output = io.StringIO()
        sys.print_exception(error, output)
        str_error = output.getvalue()
        output.close()
        self.send("Error: " + str_error, status=500)

    def on_request(self, handler):
        """ Set request handler """
        self._on_request_handler = handler
```

```
    def _get_request(self):
        """ Return request body """
        return str(self._connect.recv(4096), "utf8")
```

Файл - main.py

```
import network
import webrepl
from server import MicroPyServer
from aes import Crypto

server = MicroPyServer()

def connectToWiFi():
    print("Wi-Fi")

    sta_if = network.WLAN(network.STA_IF)

    print("Active: ", sta_if.active())

    sta_if.active(True)
    sta_if.connect('wifi', 'password1234567890')
    print("Is connect: ", sta_if.isconnected())
    print(sta_if.ifconfig())

def connectWebRepl():
    webrepl.start(password="qwertyu")
    print("http://micropython.org/webrepl/#192.168.0.107:8266")

def show_index_page(request):
    server.send("THIS IS INDEX PAGE")

def show_info_page(request):
    server.send("THIS IS INFO PAGE")
```

```
def show_crypto_page(request):
    message = "This is a message we will encrypte with AES."
    key = "PasswordPasswordPasswordPassword"
    bit = 256

    crypto = Crypto(bit)
    encryptText = crypto.encrypt(message, key)

    server.send(str(encryptText))

def connectToServer():
    server.add_route("/crypto", show_crypto_page)
    server.add_route("/info", show_info_page)
    server.add_route("/", show_index_page)
    server.start()

def run():
    connectToWiFi()
    connectWebRepl()
    connectToServer()

run()
```