КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

ВИВЧЕННЯ ТА АНАЛІЗ МЕТОДІВ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ В ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

Виконала студентка 4-го курсу

Вікторія ДОБРИДЕНЬ _____

(підпис)

Науковий керівник:

доцент,
кандидат фізико-математичних наук

Олександр ГАЛКІН _____

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем

«____» _____ 2023 р.,

протокол №
Завідувач кафедри
Олександр ПРОВОТАР _____

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 63 сторінки, 20 ілюстрацій, 8 джерел посилань.

АЛГОРИТМ ЕЛЬ-ГАМАЛЯ, АЛГОРИТМИ КРИПТОШИФРУВАННЯ, АСИМЕТРИЧНЕ ШИФРУВАННЯ, ЕЛІПТИЧНІ КРИВІ, ЕФЕКТИВНІСТЬ, СИМЕТРИЧНЕ ШИФРУВАННЯ, СИСТЕМИ ЗАХИСТУ ДАНИХ, СТІЙКІСТЬ ДО ЗЛАМУ, ХЕШ-ФУНКЦІЇ, ЦИФРОВІ ПІДПИСИ.

Об’єктом роботи є програмне забезпечення, в якому застосовуються методи криптографічного захисту інформації. Предметом роботи є методи криптографічного захисту інформації в програмному забезпеченні, різновиди методів шифрування, їх особливості та способи застосування.

Метою роботи є вивчення існуючих методів криптографічного захисту в програмних одиницях, їх аналіз та застосування в програмному забезпеченні, демонстрація роботи методів на прикладі розроблених програми, порівняння та аналіз різних методів.

Методи розроблення: мова програмування Python, середовище розробки PyCharm, бібліотеки cryptography, pycryptodone.

Результати роботи: проаналізовано основні алгоритми криптошифрування та системи захисту, реалізовані методи Ель-Гамаля та Ель-Гамаля в еліптичній криптографії. Здійснено порівняння цих методів за ефективністю та стійкістю до зламу.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогоднішній день захист інформації є критично важливим аспектом в сфері програмного забезпечення. Розвиток технологій та збільшення кількості зв'язку та обміну даними призвели до зростання загрози безпеці інформації. Відповідно, розробники програмного забезпечення використовують методи криптографічного захисту для забезпечення конфіденційності, цілісності та автентичності даних.

Сучасний стан об'єкта дослідження відображає значний прогрес у сфері криптографічного захисту в програмному забезпеченні. З'явилися потужні алгоритми шифрування, хеш-функції та протоколи обміну ключами, що дозволяють ефективно захищати інформацію від несанкціонованого доступу.

Програмне забезпечення для криптографічного захисту надає широкий спектр можливостей, таких як шифрування даних, генерація цифрових підписів, управління ключами та багато інших. Криптографічні бібліотеки та інструменти розробки дозволяють розробникам ефективно використовувати ці методи для забезпечення безпеки своїх програм.

Актуальність роботи та підстави її виконання.

1. Зростання загрози кібербезпеці: У сучасному світі, коли цифрова інформація відіграє важливу роль у різних сферах, зростає загроза кібератак та несанкціонованого доступу до даних. Криптографічні методи захисту інформації в програмному забезпеченні стають невід'ємною складовою в боротьбі з цими загрозами.
2. Вимоги до конфіденційності та захисту даних: Компанії, установи та користувачі дедалі більше підтримують потребу у забезпеченні конфіденційності своїх даних. Криптографічні методи забезпечують

можливість шифрування даних, збереження цілісності та автентичності інформації, задовольняючи ці вимоги.

3. Розвиток програмного забезпечення: Розробка програмного забезпечення стає все більш складною та розмаїтою задачею, а тим самим збільшується і складність забезпечення безпеки. Використання методів криптографічного захисту є ефективним способом підвищення рівня безпеки програмного забезпечення.

4. Нові технології та виклики: З'явлення нових технологій, таких як хмарні обчислення, Інтернет речей (IoT) та блокчейн, вимагає впровадження ефективних методів криптографічного захисту. Вивчення та аналіз сучасних методів криптографічного захисту в програмному забезпеченні є важливим для розуміння та вирішення цих викликів.

Мета й завдання роботи.

Метою кваліфікаційної роботи є вивчення та застосування на практиці, демонстрація роботи методів криптографічного захисту інформації в програмній інженерії. Для досягнення мети завдання поділене на кроки:

- Огляд літератури та наявних досліджень в обраній області.
- Вивчення та аналіз різних методів криптографічного захисту, включаючи алгоритми шифрування, цифрові підписи та інші засоби забезпечення безпеки.
- Вивчення принципів роботи обраних методів та їх відмінностей.
- Опис математичних основ криптографічних алгоритмів та протоколів.
- Вибір двох методів криптографічного захисту, які будуть реалізовані.
- Проектування та розробка програмного забезпечення, що виконує вибрані методи.

- Зіставлення результатів роботи програмного забезпечення, реалізованого для кожного з методів.
- Аналіз ефективності, безпеки та швидкодії кожного методу.
- Висновки та порівняння здатності обраних методів задовольняти критерії безпеки та вимоги до захисту інформації в програмному забезпеченні

Можливі сфери застосування об'єкту роботи.

Методи криптографічного захисту інформації в програмній інженерії можуть бути застосовані в різних сферах для забезпечення безпеки та конфіденційності інформації. Деякі з можливих сфер застосування:

1. Комунікаційні мережі: Криптографічні методи можуть бути використані для захисту комунікаційних мереж від несанкціонованого доступу та перехоплення даних. Вони допомагають забезпечити безпечну передачу інформації через мережу, включаючи захист від атак типу "людина посередник" (man-in-the-middle) та шифрування переданих даних.
2. Криптографічні протоколи: У програмній інженерії використовуються криптографічні протоколи для забезпечення безпеки під час взаємодії між різними системами або компонентами програмного забезпечення. Це можуть бути протоколи обміну ключами, протоколи аутентифікації, протоколи захищеного з'єднання та інші.
3. Зберігання даних: Криптографічні методи можуть бути використані для захисту даних, які зберігаються в системах зберігання, базах даних або файлових системах. Шифрування даних допомагає запобігти несанкціонованому доступу до інформації в разі втрати або крадіжки фізичного носія або несанкціонованого доступу до системи.
4. Конфіденційність та цілісність даних: Криптографічні методи дозволяють забезпечити конфіденційність даних, забезпечуючи, що

інформація залишається секретною та доступною тільки авторизованим користувачам. Вони також допомагають забезпечити цілісність даних, запобігаючи несанкціоновані зміни або підробки інформації.

5. Аутентифікація та авторизація: Криптографічні методи можуть бути використані для підтвердження ідентичності користувачів, аутентифікації та авторизації доступу до систем або ресурсів. Вони дозволяють забезпечити безпеку при вході в систему, контролюючи доступ до конфіденційної інформації або функціональності системи.

Це лише кілька прикладів сфер застосування методів криптографічного захисту інформації в програмній інженерії. Відповідно до потреб і вимог проекту, можуть бути використані різні комбінації криптографічних методів для забезпечення безпеки інформації.

РОЗДІЛ 1

ОГЛЯД ЛІТЕРАТУРИ ТА НАЯВНИХ ДОСЛІДЖЕНЬ

1.1 Теоретичні відомості про криптографічні методи захисту інформації

Ю.Мінгальова у своїй статті "Новітні криптографічні методи захисту інформації" [1] розглядає сучасні симетричні та асиметричні методи криптографічного захисту інформації, квантову криптографію та особливості криптосистем на основі еліптичних кривих.

Криптогра́фія — наука про математичні методи забезпечення конфіденційності, цілісності і автентичності інформації.

Розвиток інформаційних технологій та комп'ютерних систем в усіх сферах людської діяльності призвів до зростання інтересу до проблеми інформаційного захисту. Захист інформації включає методи і засоби, що забезпечують цілісність, конфіденційність та доступність інформації в умовах різних загроз. Криптографія відіграє провідну роль у забезпеченні інформаційної безпеки, забезпечуючи конфіденційність, цілісність та автентичність передаваних даних.

Сфери застосування криптографічних методів в програмній інженерії включають шифрування комунікаційних мереж, захист даних у системах зберігання, аутентифікацію та авторизацію користувачів, а також забезпечення безпеки в електронній торгівлі, системах управління та контролю. Розробка високопродуктивних методів шифрування з високою криптографічною стійкістю є важливим аспектом розв'язання проблеми інформаційної безпеки.

Методи криптографічного захисту інформації включають системи шифрування, алгоритми захисту від фальшивої інформації, криптографічні протоколи розподілу ключів, автентифікації та підтвердження факту передачі інформації. Криптографічна стійкість визначається здатністю криптографічних алгоритмів і протоколів протистояти методам дешифрування.

Сучасна криптографія використовує відкриті алгоритми шифрування, які передбачають використання обчислювальних засобів. На сьогоднішній день існує багато перевірених методів шифрування, які за умови використання достатньо довгих ключів і правильної реалізації алгоритму забезпечують недоступність зашифрованого тексту для криптоаналізу.
Для криптографічних методів захисту інформації існують загальні вимоги, які можна сформулювати наступним чином:

1. Зашифроване повідомлення може бути розшифроване тільки за наявності правильного ключа, який використовується для шифрування повідомлення.
2. Кількість операцій, необхідних для визначення використаного ключа шифрування за фрагментом повідомлення та відповідним йому відкритим текстом, повинна бути не менше загальної кількості можливих ключів.
3. Кількість операцій, необхідних для розшифрування інформації шляхом перебору можливих ключів, повинна бути значною, виходячи за межі сучасних можливостей комп'ютерів (з урахуванням можливості мережних обчислень).
4. Знання алгоритму шифрування не повинно впливати на надійність захисту інформації.
5. Навіть незначна зміна ключа повинна призводити до значних змін в зашифрованому повідомленні, навіть якщо використовується той самий ключ.

6. Алгоритм має бути придатним для як програмної, так і апаратної реалізації, при цьому зміна довжини ключа не повинна суттєво погіршувати якість алгоритму шифрування.

1.2 Методи криптографічного захисту в державних інформаційних ресурсах

Т.І. Каткова, доктор технічних наук, доцент, професор кафедри кібербезпеки та

інформаційних технологій, університету митної справи та фінансів у своїй науковій статті “ЗАБЕЗПЕЧЕННЯ КРИПТОГРАФІЧНОГО ЗАХИСТУ ДЕРЖАВНИХ ІНФОРМАЦІЙНИХ РЕСУРСІВ” [2] в процесі роботи розглянула криптографічні методи та засоби захисту інформації, які є одним з елементів комплексної системи захисту інформації, висвітлила види шифрування, а також сутність електронних підписів. Наведено правові підстави застосування криптографічних методів захисту інформації.

Для повного задоволення потреб сучасного суспільства існує потреба в інформаційному забезпеченні в усіх сферах людської діяльності і, зокрема, надійному захисті інформації. Особливо гостро ця проблема постає у зв'язку з масовою комп'ютеризацією, асоціацією комп’ютерів у комп’ютерних мережах та використання Інтернету. Теорія захисту інформації доводить, що якщо система захисту будується з урахуванням усіх сучасних методів і засобів захисту, а також, якщо підприємство ретельно відібрало та навчило персонал, який не робить помилок, то навмисні дії зловмисників у такій системі є неможливими. Однак це не зовсім так. З часом система захисту застаріває, змінюється персонал і втрачається пильність, зловмисники знаходять нові способи нападу та способи подолання захисту, які були невідомими на момент розробки системи захисту. Отже, якщо у Вас є розумні очікування щодо надійності вашої системи захисту інформації, все ж слід пам'ятати

основне правило інформаційної безпеки: жодна система захисту не витримає цілеспрямовані дії вмілого зловмисника, давно озброєного сучасною технікою. Це правило було розроблене багаторічним досвідом фахівців з інформаційної безпеки та є універсальним. Це не залежить від рівня захисту, цілісності користувачів і адміністраторів, апаратного та програмного забезпечення. Правило стверджує, що проблема не в тому, чи зловмисники зламають систему захисту, а в тому, коли вони це зроблять. Мета захисту інформації полягає в тому, щоб збій системи стався якомога пізніше.

Проблемі створення та функціонування засобів криптографічного захисту інформації присвячено достатньо публікацій у відкритих джерелах, такі вчені як Пономаренко В.С., Вербицький О.В., Хорошко В. А., Фаль О.М. тощо. Пономаренко В.С. у своїх роботах у систематизованому вигляді розглядає питання створення симетричних і асиметричних криптографічних систем захисту інформації. Науковець Вербицький О.В. вивчає проблеми протидії та розслідування злочинів, вчинених при використанні електронно-обчислювальних машин, систем і комп'ютерних мереж. Вчені Хорошко В. А. та Фаль О. М. описують засоби теорії дискового шифрування, ключ процедури управління, основи розробки та впровадження криптографічних протоколів та механізм електронного цифрового підпису.

Мета цієї статті полягає в визначенні та обробці основних криптографічних методів та засобів захисту, видів шифрування, аналіз програмного забезпечення іноземних та українських розробників, призначене для криптографічного захисту інформації.

Один із елементів комплексної інформації системи захисту - це криптографічний захист інформації. Такий вид захисту інформації реалізується шляхом перетворення інформації за допомогою ключів на основі математичних методів. Є дві цілі використання криптографічних методів - приховати інформацію шляхом її шифрування і підтвердити значення документів з використанням електронного цифрового підпису. Іншими

словами, на думку В.В. Поповського, криптографічні методи вирішують дві проблеми - забезпечення конфіденційності інформації шляхом запобігання витягу зловмисником інформації з каналу зв’язку та забезпечення

цілісності інформації шляхом запобігання зміни інформації та внесенню в неї неправдивого змісту. Є два розділи науки, пов'язані з криптографічними методами: криптографія і криптоаналіз, які разом утворюють криптологію. Криптографія вивчає математичні перетворення, які дозволяють зашифрувати інформацію. Криптоаналіз вивчає методи дешифрування без знання секретного ключа. Засоби криптографічного захисту інформації поділяються на:

- засоби, що реалізують криптографічні алгоритми перетворення інформації;

- засоби, системи та комплекси захисту від нав'язування з використанням неправдивої інформації криптографічних алгоритмів перетворення інформації;

-засоби, системи та комплекси, призначені для виготовлення та розповсюдження ключів криптографічного захисту інформації;

- системи та комплекси, що входять до комплексів захисту інформації від несанкціонованих дій

-доступ та використання криптографічних алгоритмів перетворення інформації.

Засоби криптографічного захисту разом з ключем та іншими видами документації, які забезпечують необхідний рівень захисту, формують криптографічну систему. Шифрування дозволяє захистити інформацію шляхом перетворення її на незрозумілий текст (шифртекст) з можливістю подальшого дешифрування. Шифрувати можна як прості тексти, так і комп’ютерні файли.

Шифрування поділяється на симетричне та асиметричне. Симетричне шифрування використовує один секретний ключ як для шифрування, так і для дешифрування. Асиметричне шифрування використовує відкритий ключ і інший секретний ключ для дешифрування, згенерований за допомогою псевдовипадкового генератора чисел.

Асиметричне шифрування також називають шифруванням з відкритим ключем. Недолік симетричного шифрування - це необхідність передати ключ особі, до якої адресується текст, що тягне за собою його розкриття і розшифрування інформації зловмисниками. Перевагою симетричного шифрування є його вища швидкість, ніж асиметричного шифрування, оскільки асиметричне шифрування використовує довші ключі, що збільшує час шифрування.

Спосіб шифрування тексту базується на алгоритмі, і зашифрований текст може бути лише розшифрованим за допомогою ключа. Один алгоритм з різними ключами можна використовувати для надсилання повідомлень різним одержувачам. Секретність визначається ключем, а не алгоритмом, оскільки більшість алгоритмів відомі громадськості. У зв’язку зі збільшенням продуктивності комп’ютера кількість комбінацій для підбору ключа збільшується, тому нам доводиться використовувати все довші та довші варіанти, що збільшує час для шифрування. Важливою характеристикою методів шифрування є їх криптографічна стійкість, тобто для криптографічного захисту інформації в комп'ютерній мережі, необхідно створити спеціальну службу, яка генерує ключі та розповсюджує їх серед користувачів мережі.

Для створення електронного підпису контрольна сума та додаткова інформація шифруються за допомогою закритого ключа відправника. Щоб уникнути перехоплення та повторного використання, до підпису додається порядковий номер. Електронний підпис дозволяє підтвердити авторство документа та гарантує цілісність інформації і відсутність спроб її спотворити.

Документ складається з тексту, електронного підпису і сертифіката користувача, що містить дані користувача, його ідентифікаційне ім’я та відкритий ключ розшифровки для перевірки підпису адресата на документі.

Електронний підпис дозволяє захистити інформацію від таких злочинних дій:

– «відмова від авторства», коли автор документа відмовляється від авторства;

- «фальсифікація», коли одержувач документа підробляє його;

- «зміна», коли одержувач документа вносить до нього зміни;

- «маскування», коли користувач маскується під іншого користувача.

Для підтвердження повідомлення мають бути виконані такі умови:

– відправник повинен поставити підпис у повідомленні, що містить додаткову інформацію, яка залежить від повідомлення та одержувача повідомлення, але відомий лише відправнику;

- правильний підпис неможливо зробити без додаткової інформації;

- підпис має залежати від часу, щоб старі повідомлення не могли бути використані; це відрізняє електронний підпис із власноручного;

- одержувач повинен мати можливість перевірити, що підпис належить відправнику та є правильним.

Таким чином, електронний підпис - це різновид пароля, який залежить від відправника, одержувача та змісту повідомлення.

Відповідно до Закону України «Про електронні документи та електронний документообіг». Електронний підпис – обов’язковий реквізит електронного документа, який використовується для ідентифікації автора та/або підписувача електронного документа іншими суб’єктами електронного документообігу та накладення електронного підпису завершує створення електронного документа. Закон України «Про електронний цифровий підпис» визначає правовий статус електронного цифрового підпису, згідно з якою

електронний цифровий підпис - це вид електронного підпису, отриманий у результаті криптографічного перетворення набору електронних даних, приєднаних до цього набору або логічно об’єднаних з ним і дозволяє

підтвердити його цілісність та ідентифікувати підписанта.

Електронний цифровий підпис накладається за допомогою закритого ключа та перевіряється за допомогою відкритого. Порядок криптографічного захисту інформації з обмеженим доступом, розголошення якої може завдати шкоди державі, суспільству, особі в Україні визначається Положенням про порядок криптографічного захисту інформації в Україні. Для криптографічного захисту конфіденційної інформації криптосистеми використовуються засоби криптографічного захисту, які мають сертифікат відповідності. Існує велика кількість програмних продуктів, призначених для криптографічного захисту інформації, і іноземних розробників, і українських.

Однією з кращих програм для шифрування інформації є BestCrypt від фінської компанії Jetico. Він дозволяє створити зашифрований контейнер для зберігання інформації на будь-якому типі носія і призначений для роботи як під Windows, так і під Linux. Програма за бажанням може використовувати один із найсильніших алгоритмів, реалізованих з 256-бітним ключем: Rijndael (AES), Blowfish і Twofish. Новіші версії алгоритма Blowfish може використовувати 448-бітний ключ.

Відома також програма «Private Disk» від молдовської компанії «Dekart». Вона дозволяє створити зашифрований віртуальний диск для зберігання інформації. Шифрування здійснюється за допомогою алгоритму AES 256. При роботі з інформацією файли на віртуальному диску мають ті ж властивості, що й незашифровані, доки користувач не блокує доступ. Віртуальний диск захищений від використання вірусів, троянських і шпигунських програм, вбудований дисковий брандмауер. Діє система криптографічного захисту інформації «Карма» від української компанії

«NetCom Technology». Вона призначена для забезпечення використання електронного цифрового підпису та шифрування, зокрема, в юридично значимому електронному документообігу. Особливістю цієї системи є можливість додавати до електронного цифрового підпису зображення власноручного підпису. В результаті електронний документ буде виглядати як паперовий.

ТОВ «СКЗ «КріптоСофт» пропонує програмний комплекс криптографічного захисту інформації. «Криптосервер» для роботи під MS Windows 8, MS Windows 10. Цей комплекс забезпечує захист даних, що передаються через незахищені публічні (Інтернет) або відкриті (наприклад, виділені лінії, MPLS) канали. Дані захищені шифруванням на основі вітчизняних алгоритмів шифрування. Максимальний рівень доступу обмежений щодо інформації, яка захищається цим пакетом, є «конфіденційною».

1.3 Криптографічне перекодування інформації

Рудницький В.М., Бабенко В.Г., Рудницький С.В. у своїй статті “Метод синтезу матричних моделей операцій криптографічного перекодування інформації” [3] пропонують математичний апарат, який покладений в основу розробки методу синтезу матричних моделей операцій криптографічного перекодування інформації на основі заданих вхідної та вихідної операцій криптографічного кодування.

Зараз основними засобами захисту інформації в системах і мережах є криптографічні методи шифрування. Проте існуючі засоби шифрування не завжди відповідають вимогам продуктивності, особливо високошвидкісних комп'ютерних системах. Тому виникає потреба в розробці нових методів шифрування, які б забезпечували стійкий захист інформації та високу продуктивність.

Дослідження показують, що для забезпечення криптостійкості блокових шифрів потрібно використовувати бент-функції, які базуються на спеціальних логічних операціях. Проблема полягає у науковому обґрунтуванні можливості синтезу стійких до криптоаналізу логічних функцій, розробці методів шифрування і програмних засобів для їх реалізації.

Недавні дослідження зосереджуються на моделі синтезу групи операцій перекодування, які забезпечують криптографічне перетворення без залучення інверсій. Пропонується метод синтезу базових операцій криптографічного перетворення зі збереженням інформативності. Однак, не існує математичного апарату для побудови операцій перекодування і відсутні методи знаходження матриці перекодування за заданими матрицями кодування.

Метою статті є розробка методу синтезу матричних моделей операцій криптографічного перекодування інформації. Основна частина роботи полягає у знаходженні операцій перекодування з використанням композиції логічних операцій, які формуються з визначеного набору операцій кодування. Мета полягає в тому, щоб операція перекодування також належала множині операцій кодування, що дозволяє зменшити кількість перетворень, необхідних для отримання даних, закодованих другим користувачем.

В результаті проведених досліджень була розроблена математична модель синтезу групи операцій перекодування, що базуються на двох, трьох та чотирьохрозрядних логічних операціях. Ці операції забезпечують криптографічне перетворення без врахування інверсій, відповідно до матричного представлення вхідних та вихідних логічних операцій.

У даній роботі було запропоновано метод побудови математичної моделі матриці перекодування на основі відомих вхідних та вихідних матриць кодування з використанням операції суми за модулем два.

Дослідження включає математичний апарат, що лежить в основі розробки методу синтезу матричних моделей операцій криптографічного перекодування інформації. Крім того, на прикладах моделей матриць двох, трьох та чотирьохрозрядних операцій криптографічного перетворення підтверджено правильність застосування запропонованого методу.

РОЗДІЛ 2

КРИПТОГРАФІЧНІ АЛГОРИТМИ

2.1 Симетричне та асиметричне шифрування

Метод симетричного шифрування використовує один і той самий криптографічний ключ як для шифрування, так і для дешифрування даних. Це приводить до простоти процесу.

Приклад: Припустимо, ми маємо повідомлення, яке потрібно зашифрувати: "Hello, World!" І ми обираємо секретний ключ, наприклад, "SECRETKEY".

Крок 1: Генерація шифрованого тексту Симетричні шифри, такі як AES, використовують блочну криптографію, де повідомлення розбивається на блоки фіксованого розміру, які потім шифруються. У цьому прикладі ми розбиваємо повідомлення на блоки по 8 символів.

- Блок 1: "Hello, W"
- Блок 2: "orld!"

Крок 2: Шифрування Кожен блок повідомлення шифрується за допомогою секретного ключа. Процес шифрування використовується для перетворення блоків в шифрований вигляд за допомогою ключа.

- Застосовуємо шифрування до Блоку 1 з використанням ключа "SECRETKEY" і отримуємо шифрований блок 1.
- Застосовуємо шифрування до Блоку 2 з використанням ключа "SECRETKEY" і отримуємо шифрований блок 2.

Крок 3: Отримання шифрованого повідомлення Після шифрування всіх блоків отримуємо шифровані блоки.

- Зашифрований Блок 1: "x1jfl2oi"
- Зашифрований Блок 2: "o34h78d2"

Крок 4: Розшифрування Для розшифрування шифрованого повідомлення використовується той самий ключ "SECRETKEY".

- Застосовуємо розшифрування до Зашифрованого Блоку 1 з використанням ключа "SECRETKEY" і отримуємо розшифрований Блок 1.
- Застосовуємо розшифрування до Зашифрованого Блоку 2 з використанням ключа "SECRETKEY" і отримуємо розшифрований Блок 2.

Крок 5: Отримання початкового повідомлення Після розшифрування отримуємо початкові блоки.

- Розшифрований Блок 1: "Hello, W"
- Розшифрований Блок 2: "orld!"

Крок 6: Об'єднання блоків З'єднуємо розшифровані блоки разом, і отримуємо початкове повідомлення.

- Початкове повідомлення: "Hello, World!"

Однією з найвідоміших особливостей симетричного шифрування є його простота процесу, оскільки використовується один ключ як для шифрування, так і для дешифрування. Саме через це симетричне шифрування є відмінним варіантом, коли потрібно зашифрувати велику кількість даних. Основні переваги алгоритмів симетричного шифрування включають:

1. Вищу швидкість порівняно з асиметричним шифруванням
2. Менші обчислювальні вимоги.
3. Відсутність впливу на швидкість Інтернету.

Існує безліч алгоритмів симетричного шифрування! Найпоширеніші серед них включають AES, RC4, DES, 3DES, RC5, RC6 та інші. Розглянемо три найпопулярніші з них.

DES (стандарт шифрування даних) - це найстаріший симетричний метод шифрування, представлений в 1976 році. DES був розроблений компанією IBM для захисту конфіденційних урядових даних і в 1977 році був офіційно прийнятий федеральними агентствами США. DES був використовуваний у версіях 1.0 і 1.1 протоколу TLS (захист транспортного рівня).

DES перетворює блоки відкритого тексту розміром 64 біти в зашифрований текст, розділяючи його на два окремих блоки по 32 біти і застосовуючи процес шифрування до кожного з них. Процес включає 16 циклів, що включають розширення, перестановку, заміну та інші операції, які здійснюються над даними для створення зашифрованого тексту. В результаті отримуються блоки зашифрованого тексту розміром 64 біти.

У 2005 році DES було оголошено застарілим і його було замінено алгоритмом AES. Одним з найбільших недоліків DES була коротка довжина ключа шифрування, що полегшувало його злам. Протокол TLS 1.2, який широко використовується в даний час, вже не використовує DES для шифрування.

3DES, також відомий як TDEA (трійний алгоритм шифрування даних), є вдосконаленою версією алгоритму DES. Цей алгоритм був розроблений для усунення недоліків DES і був запроваджений наприкінці 1990-х років. 3DES використовує трициклі DES для обробки кожного блоку даних. Це зробило

3DES набагато більш стійким до зламу, ніж його попередник DES. TDEA став популярним алгоритмом шифрування в платіжних системах та інших фінансових технологіях. Він також використовується в криптографічних протоколах, таких як TLS, SSH, IPsec і OpenVPN.

Проте, всі шифрувальні алгоритми нарешті стають вразливими до часу, і 3DES не є винятком. Уразливість Sweet32 була виявлена Картікеяном Бхаварганом і Гаетаном Леурентом. Це відкриття спонукало галузь безпеки переглянути стан алгоритму, і Національний інститут стандартів і технологій США (NIST) офіційно оголосив про це в проекті управління, опублікованому в 2019 році.

Згідно з цим проектом, використання 3DES має бути припинено в нових додатках після 2023 року. Важливо відзначити, що TLS 1.3, останній стандарт протоколів SSL/TLS, також відмовився використовувати 3DES.
AES (advanced encryption system), також відомий як Rijndael, є одним з найпопулярніших алгоритмів шифрування. Він був розроблений як альтернатива DES і після того, як NIST затвердив його в 2001 році, став новим стандартом шифрування. AES є сімейством блокових шифрів з різними розмірами блоків і довжиною ключів.

AES використовує методи підстановки і перестановки. Спочатку незашифровані дані поділяються на блоки, а потім застосовується шифрування з використанням ключа. Процес шифрування включає різні кроки, такі як зсуви рядків, змішування стовпців і додавання ключів. Кількість таких трансформацій (раундів) залежить від довжини ключа і може бути 10, 12 або 14. Важливо відзначити, що останній раунд відрізняється від попередніх і не включає процес змішування стовпців.

Переваги використання алгоритму шифрування AES в тому, що AES є безпечним, швидким і гнучким алгоритмом шифрування. Порівняно з DES,

AES працює значно швидше. Однак, найбільшою перевагою є можливість використання ключів різної довжини, що забезпечує більшу стійкість шифрування. Чим довший ключ, тим важче його зламати.

На сьогоднішній день AES є найпопулярнішим алгоритмом шифрування і використовується в багатьох різних застосунках. Він застосовується для безпеки бездротових мереж, шифрування файлів і процесорів, забезпечення безпеки веб-сайтів за допомогою протоколів SSL / TLS, захисту Wi-Fi мереж, шифрування мобільних додатків, віртуальних приватних мереж (VPN) та багатьох інших. Багато урядових установ США також використовують AES для захисту конфіденційної інформації.

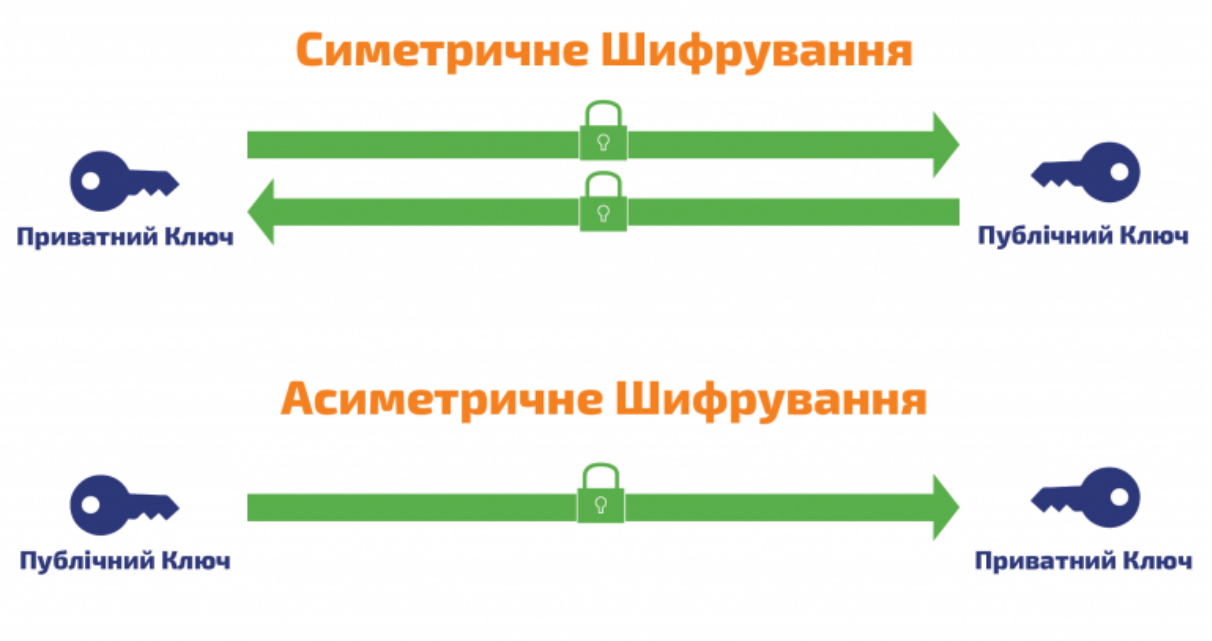

Рис. 2.1 Різниця між симетричним та асиметричним шифруванням

У асиметричного шифрування є свої переваги. Асиметричне шифрування, на відміну від симетричного, використовує кілька ключів для шифрування та розшифрування даних, які математично пов'язані один з одним. Цей метод шифрування також відомий як "криптографія з відкритим ключем".

Порівняно зі симетричним шифруванням, асиметричний підхід має кілька переваг. Перша і найочевидніша перевага - безпека. У цьому методі відкритий ключ, який є загальнодоступним, використовується для шифрування даних, тоді як розшифрування здійснюється за допомогою закритого ключа, який повинен бути добре захищений. Це гарантує, що дані залишаються захищеними від атак "людина посередині" (MITM). Для веб-серверів і серверів електронної пошти, що обслуговують багато клієнтів, це означає, що їм потрібно управляти лише одним ключем і добре захищати його. Також, асиметричне шифрування дозволяє створювати зашифровані з'єднання без необхідності зустрічатися особисто для обміну ключами.

Друга важлива особливість асиметричного шифрування - це аутентифікація. Дані, зашифровані відкритим ключем, можуть бути розшифровані тільки за допомогою пов'язаного з ним закритого ключа. Це забезпечує, що лише правильний одержувач зможе переглянути і розшифрувати дані. Це підтверджує, що ви спілкуєтесь або обмінюєтесь інформацією з вірною особою або організацією.

Отже, асиметричне шифрування забезпечує безпеку і аутентифікацію, роблячи його ефективним методом для захисту даних та спілкування з багатьма користувачами.

Подивимося на два основні типи алгоритмів асиметричного шифрування.

1. Алгоритм RSA для асиметричного шифрування: У 1977 році Рон Рівест, Аді Шамір і Леонард Адлеман з Массачусетського технологічного інституту розробили алгоритм RSA, який став найпоширенішим методом асиметричного шифрування. Цей алгоритм базується на методі "первинної факторизації". Для створення ключів обираються два випадкових простих числа заданого розміру, які множаться, утворюючи велике число. Вирішення завдання полягає в тому, щоб визначити

вихідні прості числа з цього великого числа. Ця задача практично нерозв'язна для сучасних суперкомп'ютерів та людей.

2. Алгоритм ECC для асиметричного шифрування: У 1985 році Ніл Кобліц і Віктор Міллер запропонували використання еліптичних кривих в криптографії, і ця ідея була реалізована в алгоритмі, відомому як ECC (Elliptic Curve Cryptography), у 2004-2005 роках.

У процесі шифрування методом ECC використовується еліптична крива, яка представляється набором точок, що задовольняють математичне рівняння (y^2 = x^3 + ax + b).

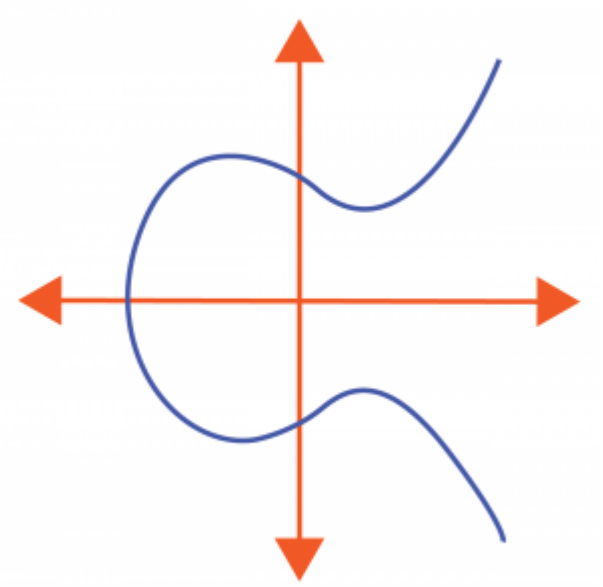

Рис. 2.2 Еліптична крива

Подібно до RSA, алгоритм ECC також працює на основі незворотності. Іншими словами, в ECC число, яке представляє точку на еліптичній кривій, множиться на інше число і дає нову точку на кривій. Для розкриття цієї головоломки потрібно знайти нову точку на кривій. Математика ECC побудована таким чином, що знаходження нової точки практично неможливе, навіть якщо ви маєте вихідну точку.

Однією з переваг використання алгоритму шифрування ECC є його вища безпека в порівнянні з RSA, навіть при коротшій довжині ключа. Також

використання коротших ключів у ECC призводить до покращення продуктивності. Короткі ключі потребують меншого мережевого навантаження та обчислювальної потужності, що особливо важливо для пристроїв з обмеженими можливостями. Використання алгоритму ECC в сертифікатах SSL/TLS допомагає прискорити процес шифрування та дешифрування, що призводить до швидшого завантаження веб-сторінок. Алгоритм ECC застосовується у шифруванні, цифрових підписах, псевдовипадкових генераторах та інших додатках.

Однак, масове використання ECC стикається з проблемою відсутності підтримки ECC SSL/TLS сертифікатів у багатьох серверних програмах та панелях управління. Очікується, що ця ситуація зміниться у майбутньому, але наразі RSA залишається найбільш поширеним алгоритмом асиметричного шифрування.

Гібридне шифрування поєднує симетричне та асиметричне шифрування і використовує переваги обох методів, створюючи потужну систему шифрування.

Кожен з цих методів має свої переваги та обмеження. Наприклад, симетричне шифрування швидке та ефективне для обробки великих обсягів даних, але не забезпечує перевірку особистості, що є важливим для забезпечення безпеки в Інтернеті. З іншого боку, асиметричне шифрування забезпечує перевірку особистості, але може бути повільним у процесі шифрування.

Ідея гібридного шифрування виникла, коли стало необхідним шифрувати дані швидко і одночасно забезпечувати перевірку особистості.

У SSL/TLS сертифікатах використовується гібридне шифрування під час "TLS handshake" – послідовного зв'язку між серверами та клієнтами (веб-браузерами). Спочатку відбувається перевірка особистості обох сторін з

використанням закритого та відкритого ключів. Після підтвердження особистості використовується симетричне шифрування з використанням ефемерного (сеансового) ключа для шифрування даних. Цей підхід дозволяє швидко передавати великі обсяги даних в Інтернеті.

Щодо питання про те, який метод шифрування кращий, немає однозначної відповіді. З точки зору безпеки, асиметричне шифрування є надійнішим через можливість аутентифікації. Однак, продуктивність також важлива, і тому симетричне шифрування завжди залишатиметься необхідним. Більшість сучасних SSL сертифікатів використовують гібридний підхід, поєднуючи асиметричне та симетричне шифрування. Це дозволяє досягти двох основних цілей: аутентифікації та забезпечення конфіденційності. Такі сертифікати надійно захищають особисті дані користувачів, такі як контактна інформація, номери банківських карт, логіни, паролі, адреси електронної пошти та інші, від перехоплення або підміни зловмисниками.

2.2 Хеш-функції

Хешування — перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються функцією хешування, або хеш-функцією.

Хеш-функція - це функція h(K), яка бере ключ K і повертає адресу, за допомогою якої можна знайти відповідну інформацію в хеш-таблиці. Колізія виникає, коли h(K1) = h(K2), що означає, що потрібно знайти нове місце для зберігання даних. Ідеальна хеш-функція повинна задовольняти дві вимоги:

1. Швидкість обчислення хеш-функції повинна бути дуже високою.
2. Кількість колізій повинна бути мінімальною.

Таким чином, перша вимога залежить від швидкості обчислювального пристрою, а друга - від властивостей даних. У реальних умовах рідко зустрічаються повністю випадкові дані, тому потрібно створити хеш-функцію, яка враховує всі аспекти ключа. Хоча теоретично неможливо створити хеш-функцію, яка генерує повністю випадкові дані з будь-яких невипадкових файлів, на практиці можна створити досить ефективну хеш-функцію за допомогою простих арифметичних операцій. Крім того, часто можна використовувати особливості даних для створення хеш-функцій з меншою кількістю колізій, ніж при випадкових даних.

Один з найпростіших методів хешування - метод середини квадрата, де ключ підноситься до квадрату, і з результату береться кілька цифр з середини. Проте, цей метод працює добре до тих пір, поки в середині результату немає багато нулів. Численні тести показали, що існують два основних типи хешування, один з них заснований на розподілі, а інший на множенні, які добре працюють. Однак, це не єдині і оптимальні методи хешування, існує багато інших підходів.

Вимоги до хеш-функцій включають такі аспекти:

1. Незворотність: Значення хеш-функції m повинно бути обчислювально неможливо відновити для будь-якого блоку даних X, такого що H(X)=m.
2. Стійкість до колізій першого роду: Обчислювально неможливо підібрати інше повідомлення N, таке що H(N) = H(M), для заданого повідомлення M.
3. Стійкість до колізій другого роду: Обчислювально неможливо знайти пару повідомлень (M, M'), які мають однаковий хеш.

Цікаво, що швидкість обчислення хеш-функцій може бути бажаною або небажаною залежно від конкретного використання. Для структур даних,

таких як хеш-таблиці, потрібні швидкі хеші з хорошою розподіленістю, тоді як для хешування паролів бажано використовувати функції, які ускладнюють обчислення на спеціалізованих пристроях, таких як FPGA, ASIC або GPU, з метою забезпечення високої обчислювальної складності (наприклад, Argon2, bcrypt, scrypt).

Хеш-функції знаходять застосування у різних областях, зокрема:

1. Кешування.
2. Хеш-таблиці та інші структури даних.
3. Цифрові підписи для забезпечення цілісності та автентифікації.
4. Пошук дублікатів.
5. Перевірка цілісності інформації, аналогічно роботі контрольних сум.

Метод ділення є простим способом хешування, де використовується залишок від ділення на число M:

$$h(K) = K \bmod M$$

Важливо ретельно обирати значення константи M. Наприклад, якщо обрати M рівним 100 і ключем буде рік народження, то розподіл хешів буде нерівномірним для деяких завдань, наприклад, ідентифікації гравців у юнацькій бейсбольній лізі. Крім того, якщо M - парне число, то значення хеш-функції буде парним для парного ключа і непарним для непарного ключа, що може призвести до небажаного результату. Ситуація стає ще гіршою, якщо M є степенем числа двійки, оскільки результат буде залежати від декількох останніх цифр ключа.

Також варто уникати значень M, кратних трьом, оскільки для символьних ключів, що відрізняються лише перестановкою літер, можуть виникати числові значення з різницею, кратною трьом. Загалом, вважається,

що краще використовувати просте число як значення M, що в більшості випадків є задовільним вибором.

Наприклад, якщо ключем є рядок "C++", то можна застосувати хеш-функцію, яка обчислює суму першого і останнього символів і використовує залишок від ділення на розмір таблиці. Однак ця хеш-функція призводить до колізій, якщо рядки мають однакові перші і останні символи. Наприклад, рядки "start" і "slant" будуть мати один і той самий хеш-індекс 29. Те саме стосується хеш-функції, яка обчислює суму всіх символів рядка. Рядки "bad" і "dab" також перетворюються в один і той же хеш-індекс. Кращі результати можна отримати, застосовуючи хеш-функцію, яка перемішує біти у символах.

На практиці метод ділення є найпоширенішим способом хешування.

Метод множення використовується для мультиплікативного хешування за допомогою наступної формули [3]:

$h(K) = [M * ((C * K) \bmod 1)]$

У цьому методі ключ помножений на константу C, яка належить інтервалу [0..1]. Потім від цього виразу взята дробова частина, яка далі помножена на константу M. Константа M обирається таким чином, щоб результат не виходив за межі хеш-таблиці. Оператор [] повертає найбільше ціле число, яке менше за аргумент.

Вибір правильної константи C є складним завданням і вимагає обережного підходу. Якщо константа C вибрана правильно, можна досягти дуже ефективних результатів. Однак, Дональд Кнут зауважує, що у деяких випадках множення може бути швидшим за ділення. Метод множення добре працює на основі того, що реальні дані часто мають невипадковий характер. Наприклад, багато ключів можуть представляти арифметичні прогресії, де

ключі у файлі представлені як (K, K + d, K + 2d, ..., K + td). Метод множення перетворює таку арифметичну прогресію на набір хеш-значень h(K), h(K + d), h(K + 2d), ..., що розрізняються, зменшуючи кількість колізій порівняно з випадковими даними. Проте варто відзначити, що метод ділення також володіє цією властивістю.

Методи хешування, які були описані раніше, є статичними, оскільки передбачають фіксований розмір хеш-таблиці і певні константи для хеш-функцій. Проте цей підхід не підходить для задач, де розмір бази даних часто і значно змінюється. Існують деякі можливості для роботи з розширюваною базою даних:

1. Використовувати початкову хеш-функцію, але це може призвести до збільшення колізій і втрати продуктивності зі зростанням бази даних.
2. Обрати хеш-функцію з "запасом", щоб мати достатньо місця для збереження даних, але це може призвести до невикористання частини дискового простору без необхідності.
3. Періодично змінювати функцію хешування і перераховувати всі адреси, що потребує багато ресурсів і може призвести до недоступності бази даних на певний час.

Однак існує техніка, відома як динамічне хешування, яка дозволяє гнучко змінювати розмір хеш-структури. У цьому підході хеш-функція генерує псевдоключ, який використовується частково для доступу до елементів. Псевдоключ є достатньо довгою бітовою послідовністю, що може адресувати всі потенційно можливі елементи. Використовуючи цей метод, розмір використаної пам'яті пропорційний кількості елементів у базі даних, що відрізняється від статичного хешування, де потрібна велика таблиця.

У динамічному хешуванні кожен запис зберігається в блоках, які відповідають фізичним блокам на пристрої зберігання даних. Якщо блок

заповнюється, його можна розділити на два, і вказівник на нові блоки додається на місце оригінального блоку. Задача полягає в побудові бінарного дерева, де кінці гілок містять покажчики на блоки, а навігація здійснюється на основі псевдоключа. Вузли дерева можуть бути двох типів: вузли, які показують на інші вузли, або вузли, які показують на блоки. Наприклад, вузол може мати наступний вигляд, якщо він показує на блок:

| Zero | Null | | Bucket | Покажчик | | One | Null |

Якщо вузол вказує на два інші вузли, він може мати такий вигляд:

| Zero | Адреса a | | Bucket | Null | | One | Адреса b |

На початку існує лише покажчик на динамічно виділений порожній блок. При додаванні нового елемента обчислюється псевдоключ, і його біти послідовно використовуються для визначення місця розташування блоку.
Хешування паролів використовує метод, що дозволяє користувачам запам'ятовувати паролі, які є осмисленими виразами або послідовностями символів, замість складних ключів. Ураховуючи, що користувачами системи є люди, а не автоматичні системи, важливо, щоб паролі були зручними для запам'ятовування. Замість використання довгих ключів, таких як 128-бітні, які складно запам'ятати, зазвичай користуються паролями довжиною від 8 до 12 символів. Якщо вимагати від користувачів працювати зі складними ключами, це може призвести до запису ключів на папері або в електронних файлах, що знижує захищеність системи. Щоб вирішити цю проблему, були розроблені методи перетворення паролів, осмислених рядків, у ключі заздалегідь заданої довжини. У більшості випадків для цього використовуються хеш-функції, які перетворюють блок даних, такий як пароль, в хеш-значення з заданим діапазоном значень. Хеш-функція має кінцеву область значень, є незворотною і зміна навіть одного біта вхідних даних призводить до зміни близько половини бітів вихідного хешу. Ці

властивості дозволяють хеш-функціям приймати паролі різної довжини та мови і генерувати ключі, які розподіляються рівномірно по заданій області значень.

2.3 Цифрові підписи

Електронний цифровий підпис (ЕЦП) (англ. *digital signature*) — вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних, який додається до цього набору або логічно з ним поєднується і дає змогу підтвердити його цілісність та ідентифікувати підписувача. Електронний цифровий підпис накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа.

Надійний засіб електронного цифрового підпису — засіб електронного цифрового підпису, що має сертифікат відповідності або позитивний експертний висновок за результатами державної експертизи у сфері криптографічного захисту інформації.

Одним з елементів обов'язкового реквізиту є електронний підпис, який використовується для аутентифікації автора та/або підписувача електронного документа іншими суб'єктами електронного документообігу.

Оригіналом електронного документа вважається електронний примірник з електронним цифровим підписом автора.

Електронний цифровий підпис є складовою частиною інфраструктури відкритих ключів.

Електронний цифровий підпис призначений для використання фізичними та юридичними особами — суб'єктами електронного документообігу:

- для аутентифікації підписувача;
- для підтвердження цілісності даних в електронній формі.

ЕЦП як спосіб ідентифікації підписувача електронного документа, дозволяє однозначно визначати походження інформації (джерело інформації), що міститься у документі. Завдяки цьому ЕЦП є також надійним засобом розмежування відповідальності за інформаційну діяльність у суспільстві, зокрема, відповідальності за дезінформування.

Електронний цифровий підпис накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа. За правовим статусом він прирівнюється до власноручного підпису (печатки). Електронний підпис не може бути визнаний недійсним лише через те, що він має електронну форму або не ґрунтується на посиленому сертифікаті ключа.

За умови правильного зберігання власником секретного (особистого) ключа його підробка неможлива. Електронний документ також не можливо підробити: будь-які зміни, несанкціоновано внесені в текст документа, будуть миттєво виявлені.

Особистий ключ ЕЦП формується на підставі абсолютно випадкових чисел, що генеруються давачем випадкових чисел, а відкритий ключ обчислюється з особистого ключа ЕЦП так, щоб одержати другий з першого було неможливо.

Особистий ключ ЕЦП є унікальною послідовністю символів довжиною 264 біти, яка призначена для створення Електронного цифрового підпису в електронних документах. Працює особистий ключ тільки в парі з відкритим ключем. Особистий ключ необхідно зберігати в таємниці, адже будь-хто, хто дізнається його, зможе підробити Електронний цифровий підпис.

Документ підписується ЕЦП за допомогою особистого ключа ЕЦП, який існує в одному екземплярі тільки у його власника. Цьому особистому ключу відповідає відкритий ключ, за допомогою якого можна перевірити відповідність ЕЦП його власнику.

Відкритий ключ використовується для перевірки ЕЦП одержуваних документів (файлів). Відкритий ключ працює тільки в парі з особистим ключем. Відкритий ключ міститься в Сертифікаті відкритого ключа, і підтверджує приналежність відкритого ключа ЕЦП певній особі. Крім самого відкритого ключа, Сертифікат відкритого ключа містить в собі персональну інформацію про його власника (ім'я, реквізити), унікальний реєстраційний номер, термін дії Сертифікату відкритого ключа. З метою забезпечення цілісності представлених у Сертифікаті даних він підписується особистим ключем Центру сертифікації ключів. Сертифікат відкритого ключа може публікуватися на сайті відповідного ЦСК відповідно до Договору про надання послуг ЕЦП.

Декілька сертифікатів можуть бути згенеровані з одним і тим же ключем. Автоматичне, без запиту користувача, формування нового сертифікату раніше засвідченого відкритого ключа може здійснюватися надавачем електронних довірчих послуг у період воєнного стану та протягом місяця з дня його закінчення. При цьому особистий ключ користувача не змінюється.

При підписанні електронного документа його початковий зміст не змінюється, а додається блок даних, так званий «Електронний цифровий підпис». Отримання цього блоку можна розділити на два етапи:

- На першому етапі за допомогою програмного забезпечення і спеціальної математичної функції обчислюється так званий «відбиток повідомлення» (англ. *message digest*).

Цей відбиток має такі властивості:

- фіксовану довжину, незалежно від довжини повідомлення;
- унікальність відбитку для кожного повідомлення;
- неможливість відновлення повідомлення за його відбитком.

Таким чином, якщо документ був модифікований, то зміниться і його відбиток, що відобразиться при перевірці Електронного цифрового підпису.

- На другому етапі відбиток документа шифрується за допомогою програмного забезпечення й особистого ключа автора.

Розшифрувати ЕЦП і одержати початковий відбиток, який відповідатиме документу, можна тільки використовуючи Сертифікат відкритого ключа автора.

Таким чином, обчислення відбитку документа захищає його від модифікації сторонніми особами після підписання, а шифрування особистим ключем автора підтверджує авторство документа.

Криптосистема RSA належить до числа перших криптосистем з відкритим ключем з підтримкою електронного цифрового підпису.

У широкому вжитку також знаходяться криптосистеми DSA та ECDSA.

У 2011 році була представлена розширена криптосистема цифрового підпису Меркле (англ. *eXtended Merkle Signature Scheme*, XMSS), яка має такі важливі властивості, як пряма секретність та стійкість до криптоаналізу із використанням квантових комп'ютерів. Заради спрощення впровадження даної криптосистеми були розпочаті роботи над стандартом RFC 8391.

2.4 Алгоритм Ель-Гамаля

Алгоритм Ель-Гамаля є альтернативою алгоритму RSA і забезпечує такий же рівень криптостійкості при однаковій довжині ключа. Безпека цього алгоритму базується на складності обчислення дискретних логарифмів. Він може використовуватись як для шифрування повідомлень, так і для створення цифрових підписів, і не обмежений патентами США.

Суть алгоритму Ель-Гамаля полягає в наступному: учасники обміну повідомленнями обирають просте число p і ціле число q, яке є первинним коренем за модулем p. Одержувач повідомлення генерує секретний ключ

$k_a < p$ і за його допомогою обчислює відкритий ключ $Y_a \equiv q^{k_a}(mod\, p)$. Відправник генерує число $k_b < p$ і зашифровує передане повідомлення M таким чином: $Y_b \equiv q^{k_b}(mod\, p)$ і $(C = M \oplus (Y_a^{k_b}\ (mod\, p))$. Значення M є послідовністю двійкових символів, які передаються через канал зв'язку. Значення $Y_a^{k_b}\ (mod\, p)$ перетворюється на послідовність двійкових символів перед підсумовуванням. Одержувач приймає повідомлення у формі Y_b і C та відновлює його: $M = (Y_b^{k_a}\ (mod\, p)) \oplus C$.

2.5 Криптосистеми на базі еліптичних кривих

Криптографія на еліптичних кривих — напрям асиметричного шифрування даних, що швидко розвивається з використанням сучасних О.К. Юдін, К.А. Вадясов, 2010 ISSN 2075-0781. Наукоємні технології, 2010. № 2(6) 59 інформаційних технологій. У криптографії на еліптичних кривих усі обчислення (наприклад, вибір значення ключа) проводяться над точками еліптичної кривої, тобто, замість звичайного складання двох чисел виконується за певними правилами складання двох точок кривої, при цьому як результат виходить третя точка. Цифровий підпис файлів або електронних поштових повідомлень виконується з використанням криптографічних алгоритмів, що використовують несиметричні ключі. Власне для підпису використовується секретний ключ, а для перевірки чужого підпису відкритий. Ключі є числами досить великої довжини (від 512 до 4096 біт) математично або функціонально пов’язаними між собою. Криптографічним алгоритмом, що стандартно використовується для методів шифрування симетричними

ключами (для цілей поширення) є RSA (Rivest, Shamir і Adleman). Хоча RSA має високу міру захисту і широко застосовується. Його застосування пов'язане з деякими проблемами та питанням криптостійкості при сучасному розвитку технологій. Альтернативна технологія криптографії на еліптичних кривих, заснована на математичному методі використання функції еліптичних кривих та дає істотні переваги перед RSA. В алгоритмах цифрового підпису активно використовуються обчислення в кінцевих полях Галуа. Ціле позитивне число a порівняно з b за модулем $p(a \equiv b \, mod \, p)$, якщо залишок від ділення b на p дорівнює a. Можна ввести операції складання і множення за модулем p. Результатом складання двох чисел за модулем p, вважатиметься залишок від ділення їх суми на число p. Неважко помітити, що результати операцій складання або множення пари довільних ненегативних цілих чисел за модулем p не перевершуватимуть число p. У результаті, можна обмежиться розглядом безлічі чисел 0,1...p – 1 із заданими на них операціями складання і множення за модулем p. Множина 0,1...p – 1 із заданими операціями складання і множення, що підкоряються звичайним законам складання, множення і розкриття дужок, утворюють кільце класів розрахунків за модулем p. Елемент b називається зворотним до елементу a, якщо ab = 1. Зворотній елемент позначається a^{-1}. Оперуючи тільки цілими ненегативними числами, неважко ввести операцію ділення як множення на зворотний елемент, операцію віднімання і навіть негативні числа. Виявляється, якщо p — просте число, то зворотний елемент існує для усіх елементів кільця (окрім природно числа O). Кільце класів розрахунків, для кожного елементу якого (окрім O) існує зворотний елемент, називають простим полем (чи кінцевим полем, або полем Галуа) і позначається GF(p). Еліптичною кривою називають безліч пар точок (X, Y), що задовольняють рівнянню: $y^2 = x^3 + ax + b$. Можна накласти обмеження на безліч значень змінних x, y і коефіцієнтів a, b, c. Обмежуючи область визначення рівняння значущою для застосувань числовою множиною ми отримаємо

еліптичну криву, задану над даним полем. У додатку до ДСТУ 4145-2002 еліптична крива над кінцевим простим полем GF(p) визначається як безліч пар (x, y), таких що $x,\ y\ \equiv\ GF(p)$, що задовольняють рівнянню:

$$y^2\ =\ x^3\ +\ ax\ +\ b\ mod\ p,\ a, b \equiv\ GF(p)$$

(1) Пари (x, y) називатимемо точками. Точки еліптичної кривої можна складати. Сума двох точок, у свою чергу, теж лежить на еліптичній кривій. Математична властивість, яка робить еліптичні криві корисними для криптографії, полягає в тому, що якщо взяти дві різні точки на кривій, то хорда, що сполучає їх, перетне криву в третій точці (оскільки ми маємо кубічну криву). Дзеркально відбивши цю точку по осі X, ми отримуємо ще одну точку на кривій (оскільки крива симетрична відносно осі X). Якщо позначити дві первинні точки як P і Q, то отримаємо останню — відбиту точку P + Q (рис. 1). Це складання задовольняє всім відомим правилам алгебри для цілих чисел.

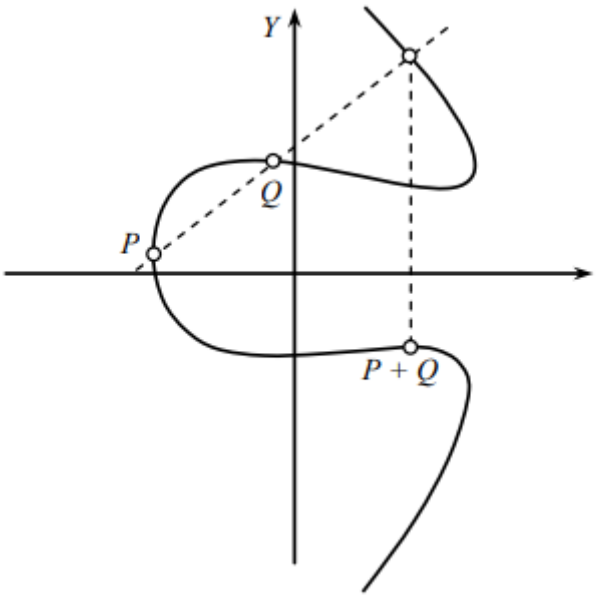

Рис.2.3 Додавання точок на еліптичній кривій

Окрім точок, що лежать на еліптичній кривій, розглядається також нульова точка. Вважається, що сума двох точок — A з координатами (XA, YA) і B з координатами (XB, YB) — рівна O, якщо XA = XB, YA = –YB (mod p). Нульова точка не лежить на еліптичній кривій, але, проте, бере участь в обчисленнях. Її можна розглядати як нескінченно видалену точку.

РОЗДІЛ 3
ЗАСТОСУВАННЯ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

3.1 IPsec

IPSec - це протокол безпеки, що використовується для забезпечення захисту мережевої системи на рівні IP. Він автентифікує та шифрує пакети даних, що передаються через IP-мережу.

Особливості IPSec:

- Захист загального пакету даних, який формується на рівні IP, включаючи заголовки вищих рівнів.
- Робота між двома різними мережами, що спрощує впровадження функцій безпеки без внесення змін до запущених програм.
- Забезпечення безпеки на рівні хоста.
- Головне завдання IPSec - захист віртуальної приватної мережі (VPN) між двома різними мережевими об'єктами.

Функції безпеки:

- Забезпечення можливості передачі повідомлень між вузлами джерела та призначення у зашифрованому вигляді для збільшення конфіденційності пакетів даних.
- Підтримка автентифікації та забезпечення цілісності даних.
- Захист від вірусних атак через управління ключами.

Робота IPSec: Робота IPSec складається з двох підчастин: зв'язок IPSec та обмін ключами в Інтернеті (IKE). Зв'язок IPSec відповідає за безпечний зв'язок між двома вузлами, використовуючи протоколи безпеки, такі як заголовок автентифікації (AH) та інкапсульований пакетній рівень безпеки (ESP). Ця частина також включає функції інкапсуляції, шифрування пакетів

даних та обробки дейтаграм IP. IKE - це протокол управління ключами, який використовується в IPSec. Хоча управління ключами може бути виконано вручну, особливо для невеликих мереж, протокол IKE застосовується великими мережами для автоматизації цього процесу.

3.2 Віртуальні приватні мережі

VPN (Virtual Private Network - віртуальна приватна мережа) є загальною назвою для технологій, які дозволяють створювати одне або кілька з'єднань мереж (логічну мережу) поверх іншої мережі, такої як Інтернет.

VPN - це технологія, яка створює зашифровані канали в Інтернеті, що дозволяє забезпечити анонімний доступ до мережі та обходити блокування.

Скорочення VPN означає Virtual Private Network - віртуальна приватна мережа. Просто кажучи, VPN - це як тунель між користувачем і захищеним VPN-сервером. У цьому тунелі VPN забезпечує захист, шифрування та зміну даних, що передаються між комп'ютером користувача та веб-сайтами або веб-сервісами.

VPN-тунель - це віртуальне зашифроване з'єднання, що можна уявити як непрозору трубу, де один кінець підключений до комп'ютера користувача, а інший - до спеціалізованого сервера, який знаходиться зазвичай в іншій країні.

Залежно від використовуваних протоколів і призначення, VPN може забезпечувати три типи з'єднань: вузол-вузол, вузол-мережа і мережа-мережа.

Сучасні типи VPN-підключень включають:

- PPTP (Point-to-point tunneling protocol) - це тунельний протокол "точка-точка", який дозволяє комп'ютеру користувача створювати захищене з'єднання з сервером шляхом створення спеціального тунелю в незахищеній мережі. Цей протокол став популярним через його підтримку в операційних системах Microsoft.
- OpenVPN - це вільна реалізація технології VPN з відкритим вихідним кодом, яка створює зашифровані "точка-точка" або "сервер-клієнт" канали між комп'ютерами. Вона може встановлювати з'єднання між комп'ютерами, що знаходяться за мережевим фаерволом NAT, без потреби внесення змін до налаштувань фаерволу. Використання цієї технології вимагає установки додаткового програмного забезпечення на операційні системи.
- L2TP (Layer 2 Tunneling Protocol) - це мережевий протокол тунелювання на канальному рівні, який поєднує в собі протоколи L2F (Layer 2 Forwarding), розроблений Cisco, і Microsoft. Він дозволяє створювати VPN з пріоритетами доступу, але не містить вбудованих засобів шифрування та аутентифікації (для забезпечення захищеної VPN його часто використовують разом з протоколом IPSec). Вважається одним з найбільш безпечних варіантів VPN-підключення, хоча налаштування може бути складним.

VPN-сервіси зазвичай пропонують два типи протоколів: OpenVPN або PPTP. Після активації підписки на VPN, ви зможете вибрати тип підключення та сервер (США, Нідерланди, Велика Британія і т. д.).

Приклади VPN:

1. IPSec (IP security) - часто використовується поверх Ipv4.
2. PPTP (point-to-point tunneling protocol) - розроблявся спільними зусиллями декількох компаній, включаючи Microsoft.
3. PPPoE (PPP (Point-to-Point Protocol) over Ethernet)

4. L2TP (Layer 2 Tunnelling Protocol) - використовується в продуктах компаній Microsoft і Cisco.
5. L2TPv3 (Layer 2 Tunnelling Protocol version 3).
6. OpenVPN SSL VPN з відкритим вихідним кодом , підтримує режими PPP, bridge, point-to-point, multi-client server
7. freelan SSL P2P VPN з відкритим вихідним кодом
8. Hamachi - програма для створення тимчасової VPN-мережі.
9. NeoRouter - zeroconfig, пряме з'єднання комп'ютерів за NAT, можна вибрати свій сервер.

Розподіл параметрів класифікації рішень VPN:

1. За ступенем захищеності використовуваного середовища: 1.1. Захищені VPN (IPSec, OpenVPN, PPTP) 1.2. Довірчі VPN (MPLS, L2TP)
2. За способом реалізації: 2.1. Спеціальне програмно-апаратне забезпечення 2.2. Програмне рішення 2.3. Інтегроване рішення
3. По призначенню: 3.1. Intranet VPN 3.2. Remote Access VPN 3.3. Extranet VPN 3.4. Internet VPN 3.5. L2TP 3.6. Client/Server VPN
4. За типом протоколу: TCP/IP, IPX, AppleTalk (переважає TCP/IP)
5. За рівнем мережевого протоколу (відповідно до ISO/OSI):
    - Вирішення, які забезпечують конфіденційність, анонімність в мережі.
    - Розблокування заблокованих сайтів та сервісів.
    - Забезпечення безпеки Wi-Fi підключення.

Ці параметри допомагають відповісти на різні вимоги та задачі, які можуть виникнути при використанні VPN.

Усі вищезазначені методи криптографічного захисту інформації активно використовуються в сучасному програмному забезпеченні та запобігають несанкціонованому доступу до неї.

РОЗДІЛ 4

РЕАЛІЗАЦІЯ ТА ПОРІВНЯННЯ АЛГОРИТМІВ ELGAMAL ТА EC ELGAMAL

4.1 Програмна реалізація методу Ель-Гамаля.

Ця програма на мові Python реалізує криптосистему Ель-Гамаля. Програма може шифрувати та розшифровувати повідомлення. Під час виконання користувач буде запрошений ввести три значення:1) число n, яке вказує довжину простого числа, яке буде згенеровано 2) число t, яке вказує бажану достовірність того, що згенероване просте число є простим числом 3) назву файлу, що містить повідомлення, яке ви хочете зашифрувати та розшифрувати Після надання користувачем необхідної інформації програма згенерує пару ключів (K1, K2), які використовуються для шифрування та розшифрування. K1 є публічним ключем і містить три цілі числа (p, g, h).p - просте число довжиною n біт. Ймовірність того, що p є простим числом, дорівнює $1 - (2^{-t})$ g - квадрат примітивного кореня mod p $h = g^x \, mod \, p$; x вибирається випадково, 1 <= x < p; h обчислюється за допомогою швидкого модульного піднесення до степені, реалізованого як modexp(base, exp, modulus) K2 є приватним ключем і містить три цілі числа (p, g, x), які описані вище. K1 та K2 записуються в файли з назвами K1 та K2. Далі програма кодує байти повідомлення у цілі числа z[i] < p. Модуль для цього називається encode(). Після того, як повідомлення було закодовано у цілі числа, ці числа шифруються та записуються у файл Ciphertext. Процедура шифрування реалізована в encrypt(). Вона працює наступним чином: Кожному числу z[i] відповідає пара (c, d), яка записується в Ciphertext. Для кожного цілого числа z[i]: $c[i] = g^y (mod \, p)$; $d[i] = z[i] * h^y \, (mod \, p)$ де y вибирається випадково, 0 <= y < p

Модуль розшифрування decrypt() зчитує кожну пару цілих чисел з Ciphertext та перетворює їх назад у закодовані цілі числа. Він реалізований наступним чином: $s = c[i]^{x} (mod\, p)$; $z[i] = d[i] * s^{-1} (mod\, p)$ Модуль decode() бере цілі числа, отримані з модуля розшифрування, і розбиває їх на байти, які були отримані в початковому повідомленні. Ці байти записуються у файл Plaintext.

```
class PrivateKey(object):
    def __init__(self, p=None, g=None, x=None, iNumBits=0):
        self.p = p
        self.g = g
        self.x = x
        self.iNumBits = iNumBits

class PublicKey(object):
    def __init__(self, p=None, g=None, h=None, iNumBits=0):
        self.p = p
        self.g = g
        self.h = h
        self.iNumBits = iNumBits
```

Рис.4.1 створення класів публічного та приватного ключів

Обчислюємо найбільший спільний знаменник a і b. припускаємо, що a > b

```
def gcd( a, b ):
        while b != 0:
            c = a % b
            a = b
            b = c
        return a
```

Рис. 4.2 Обчислення найбільшого спільного знаменника a і b

Знаходимо примітивний корінь для простого p:

```
def find_primitive_root( p ):
        if p == 2:
                return 1
        #the prime divisors of p-1 are 2 and (p-1)/2 because
        #p = 2x + 1 where x is a prime
        p1 = 2
        p2 = (p-1) // p1

        #test random g's until one is found that is a primitive root mod p
        while( 1 ):
                g = random.randint( 2, p-1 )
                #g is a primitive root if for all prime factors of p-1, p[i]
                #g^((p-1)/p[i]) (mod p) is not congruent to 1
                if not (modexp( g, (p-1)//p1, p ) == 1):
                        if not modexp( g, (p-1)//p2, p ) == 1:
                                return g
```

Рис. 4.3 Знаходження примітивного кореня для простого p

Кодуємо байти в цілі числа mod p. Читаємо байти з файлу.

```
def encode(sPlaintext, iNumBits):
        byte_array = bytearray(sPlaintext, 'utf-16')

        #z is the array of integers mod p
        z = []
        k = iNumBits//8

        #j marks the jth encoded integer
        #j will start at 0 but make it -k because j will be incremented during first iteration
        j = -1 * k
        #num is the summation of the message bytes
        num = 0
        #i iterates through byte array
        for i in range( len(byte_array) ):
                #if i is divisible by k, start a new encoded integer
                if i % k == 0:
                        j += k
                        num = 0
                        z.append(0)
                #add the byte multiplied by 2 raised to a multiple of 8
                z[j//k] += byte_array[i]*(2**(8*(i%k)))
        return z
```

Рис. 4.4 Кодування байтів

Декодуємо цілі числа до байтів вихідного повідомлення:

```
def decode(aiPlaintext, iNumBits):
        #bytes array will hold the decoded original message bytes
        bytes_array = []
        k = iNumBits//8

        #num is an integer in list aiPlaintext
        for num in aiPlaintext:
                #get the k message bytes from the integer, i counts from 0 to k-1
                for i in range(k):
                        #temporary integer
                        temp = num
                        #j goes from i+1 to k-1
                        for j in range(i+1, k):
                                #get remainder from dividing integer by 2^(8*j)
                                temp = temp % (2**(8*j))
                        #message byte representing a letter is equal to temp divided by 2^(8*i)
                        letter = temp // (2**(8*i))
                        #add the message byte letter to the byte array
                        bytes_array.append(letter)
                        #subtract the letter multiplied by the power of two from num so
                        #so the next message byte can be found
                        num = num - (letter*(2**(8*i)))

        decodedText = bytearray(b for b in bytes_array).decode('utf-16')

        return decodedText
```

Рис. 4.5 Декодування цілих чисел до байтів

Генеруємо відкритий ключ K1 (p, g, h) і закритий ключ K2 (p, g, x):

```
def generate_keys(iNumBits=256, iConfidence=32):
        #p is the prime
        #g is the primitve root
        #x is random in (0, p-1) inclusive
        #h = g ^ x mod p
        p = find_prime(iNumBits, iConfidence)
        g = find_primitive_root(p)
        g = modexp( g, 2, p )
        x = random.randint( 1, (p - 1) // 2 )
        h = modexp( g, x, p )

        publicKey = PublicKey(p, g, h, iNumBits)
        privateKey = PrivateKey(p, g, x, iNumBits)

        return {'privateKey': privateKey, 'publicKey': publicKey}
```

Рис. 4.6 Генерування ключів

Шифруємо рядок sPlaintext за допомогою відкритого ключа k:

```
def encrypt(key, sPlaintext):
        z = encode(sPlaintext, key.iNumBits)

    #cipher_pairs list will hold pairs (c, d) corresponding to each integer in z
        cipher_pairs = []
        #i is an integer in z
        for i in z:
                #pick random y from (0, p-1) inclusive
                y = random.randint( 0, key.p )
                #c = g^y mod p
                c = modexp( key.g, y, key.p )
                #d = ih^y mod p
                d = (i*modexp( key.h, y, key.p)) % key.p
                #add the pair to the cipher pairs list
                cipher_pairs.append( [c, d] )

        encryptedStr = ""
        for pair in cipher_pairs:
                encryptedStr += str(pair[0]) + ' ' + str(pair[1]) + ' '

        return encryptedStr
```

Рис. 4.7 Шифрування

Виконуємо дешифрування пар шифрів, знайдених у Cipher за допомогою приватного ключа K2 і записуємо розшифровані значення у файл Plaintext.

```
def decrypt(key, cipher):
        #decrpyts each pair and adds the decrypted integer to list of plaintext integers
        plaintext = []

        cipherArray = cipher.split()
        if (not len(cipherArray) % 2 == 0):
                return "Malformed Cipher Text"
        for i in range(0, len(cipherArray), 2):
                #c = first number in pair
                c = int(cipherArray[i])
                #d = second number in pair
                d = int(cipherArray[i+1])

                #s = c^x mod p
                s = modexp( c, key.x, key.p )
                #plaintext integer = ds^-1 mod p
                plain = (d*modexp( s, key.p-2, key.p)) % key.p
                #add plain to list of plaintext integers
                plaintext.append( plain )

        decryptedText = decode(plaintext, key.iNumBits)

    #remove trailing null bytes
        decryptedText = "".join([ch for ch in decryptedText if ch != '\x00'])

        return decryptedText
```

Рис. 4.8 Дешифрування пар шифрів

```
def test():
        assert (sys.version_info >= (3,4))
        keys = generate_keys()
        priv = keys['privateKey']
        pub = keys['publicKey']
        message = "Lorem ipsum dolor sit amet, consectetur adipiscing el
        cipher = encrypt(pub, message)
        plain = decrypt(priv, cipher)

        return message == plain
```

Рис. 4.9 Тестування алгоритму

Результати роботи алгоритму - зашифрований та розшифрований текст:

```
>>> import elgamal
>>> d = elgamal.generate_keys()
>>> privateKey = d['privateKey']
>>> publicKey = d['publicKey']
>>> cipher = elgamal.encrypt(publicKey, "This is the message I want to encrypt")
>>> print(cipher)
153263149850055314130342167953040317125805967828897060594790006360319340423 1 606627965884834367577836578428610927680367753426908001828201763200296868131 53 562827695573990486996383945425742872250376088190991444261281330605173510194 37 348203025117346583424762848060583397759834889771786036407926478619202038185 9 241351838430218685709817179334210437775062215248534717424484052680378221620 60 185742226775059496175783487126676781765235910326448203048712222704517651548 57
>>> plaintext = elgamal.decrypt(privateKey, cipher)
>>> plaintext = elgamal.decrypt(privateKey, cipher)
>>> print(plaintext)
This is the message I want to encrypt
>>>
```

Рис. 4.10 Результати

4.2 Програмна реалізація методу Ель-Гамаля в еліптичній криптографії.

Реалізуємо криптографічний алгоритм ElGamal на основі еліптичних кривих. Він використовує введені користувачем параметри n, a і b, обчислює точки на кривій, обирає базову точку, генерує публічний ключ відправника, шифрує повідомлення і розшифровує його отримувачем.

Обчислюємо значення LHS (ліва частина) і RHS (права частина) для кожного і в діапазоні від 0 до n. Функція використовується для обчислення точок на еліптичній кривій.

```
def polynomial(LHS, RHS, n):
    for i in range(0, n):
        LHS[0].append(i)
        RHS[0].append(i)
        LHS[1].append((i * i * i + a * i + b) % n)
        RHS[1].append((i * i) % n)
```

Рис. 4.11 Обчислення точок на еліптичній кривій

Генеруємо точки на еліптичній кривій, використовуючи значення LHS і RHS. Функція порівнює значення LHS[1][i] з RHS[1][j] і, якщо вони співпадають, додає точку (LHS[0][i], RHS[0][j]) до масивів arr_x і arr_y. Кількість знайдених точок повертається як результат.

```
def points_generate(arr_x, arr_y, n):
    count = 0
    for i in range(0, n):
        for j in range(0, n):
            if (LHS[1][i] == RHS[1][j]):
                count += 1
                arr_x.append(LHS[0][i])
                arr_y.append(RHS[0][j])
    return count
```

Рис. 4.12 Генерація точок на еліптичній кривій

Вводимо значення n, a і b. n відповідає довжині еліптичної кривої, a a і b визначають її форму. polynomial() обчислює значення LHS і RHS для введених n, a і b. Створюємо порожні масиви arr_x і arr_y і викликаємо

функцію points_generate(), щоб знайти точки на еліптичній кривій і отримати кількість знайдених точок.

```
n = int(input())
LHS = [[]]
RHS = [[]]
LHS.append([])
RHS.append([])
print("Enter value of 'a':")
a = int(input())
print("Enter value of 'b':")
b = int(input())

# Polynomial
polynomial(LHS, RHS, n)

arr_x = []
arr_y = []
# Generating base points
count = points_generate(arr_x, arr_y, n)
```

Рис. 4.13 Введення значень, обчислення LHS, RHS

Виводимо знайдені точки на екран. Вибираємо першу знайдену точку (bx, by) як базову точку кривої і виводять її на екран. Користувач вводить приватний ключ d. Якщо d більше або дорівнює n, виводиться повідомлення про помилку. В іншому випадку обчислюється публічний ключ Q за формулою Q = d * (bx, by), і його значення виводиться на екран.

```
# Calculation of Base Point
bx = arr_x[0]
by = arr_y[0]
print("Base Point taken is:\t(", bx, ",", by, ")\n")

print("Enter the random number 'd' i.e. Private key of Sender (d<n):")
d = int(input())
if (d >= n):
    print("'d' should be less than 'n'.")
else:
    # Q i.e. sender's public key generation
    Qx = d * bx
    Qy = d * by
    print("Public key of sender is:\t(", Qx, ",", Qy, ")\n")
```

Рис. 4.14 Виведення знайдених точок на екран

Далі виконується розшифрування шифротексту, і результат виводиться на екран як отримане повідомлення Mx.

```
    # Cipher text 1 generation
    C1x = k * bx
    C1y = k * by
    print("Value of Cipher text 1 i.e. C1:\t(", C1x, ",", C1y, ")\n")

    # Cipher text 2 generation
    C2x = k * Qx + M
    C2y = k * Qy + M
    print("Value of Cipher text 2 i.e. C2:\t(", C2x, ",", C2y, ")\n")

    # Decryption
    Mx = C2x - d * C1x
    My = C2y - d * C1y
    print("The message recieved by reciever is:\t", Mx)
```

Рис. 4.15 Виведення зашифрованого тексту, розшифрування

Результати роботи програми:

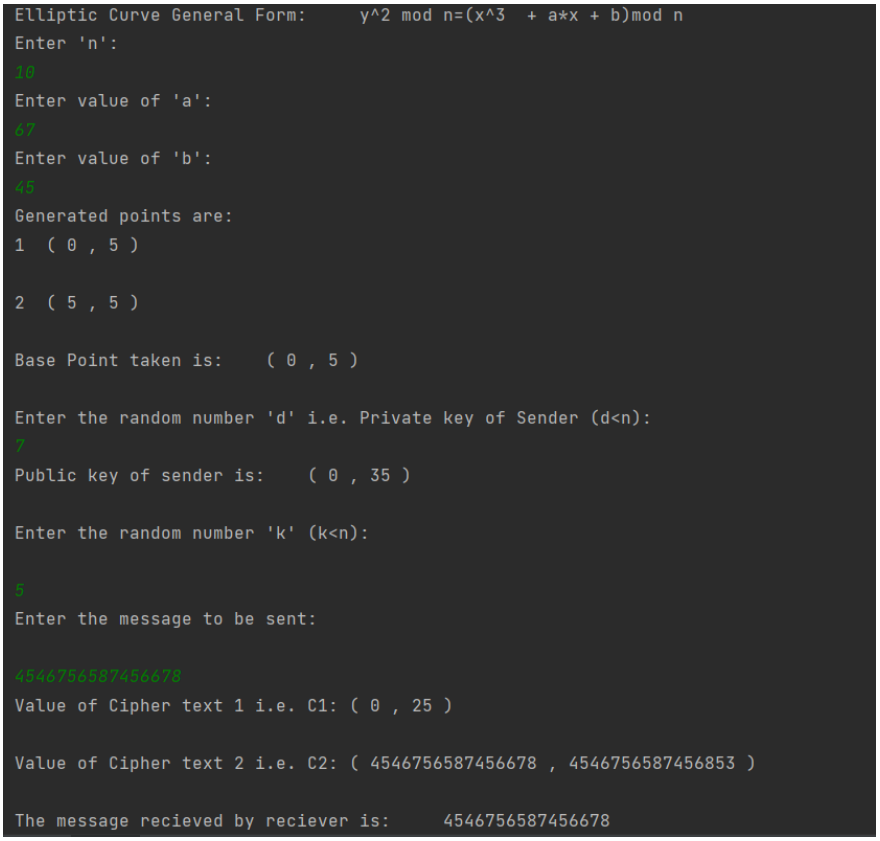

Рис. 4.16 Виведення результатів в консоль

4.3 Порівняння методів ElGamal та EC ElGamal

4.3.1 Порівняння за ефективністю

Обрахуємо час, за який виконуються алгоритми для однакових даних. Запустимо алгоритми з файлами в 1 мб, 5 мб, 10 мб та довжиною ключа 256 біт. Маємо такі результати:

Encryption time regular for 1 mb file with 256 -bit key 1858 ms

Encryption time elliptic for 1 mb file with 256-bit key 280241 ms

Encryption time regular for 5 mb file with 256 -bit key 9563 ms

Encryption time elliptic for 5 mb file with 256-bit key 1 401 276 ms

Encryption time regular for 10 mb file with 256 -bit key 19583 ms

Encryption time elliptic for 10 mb file with 256-bit key 2803615 ms

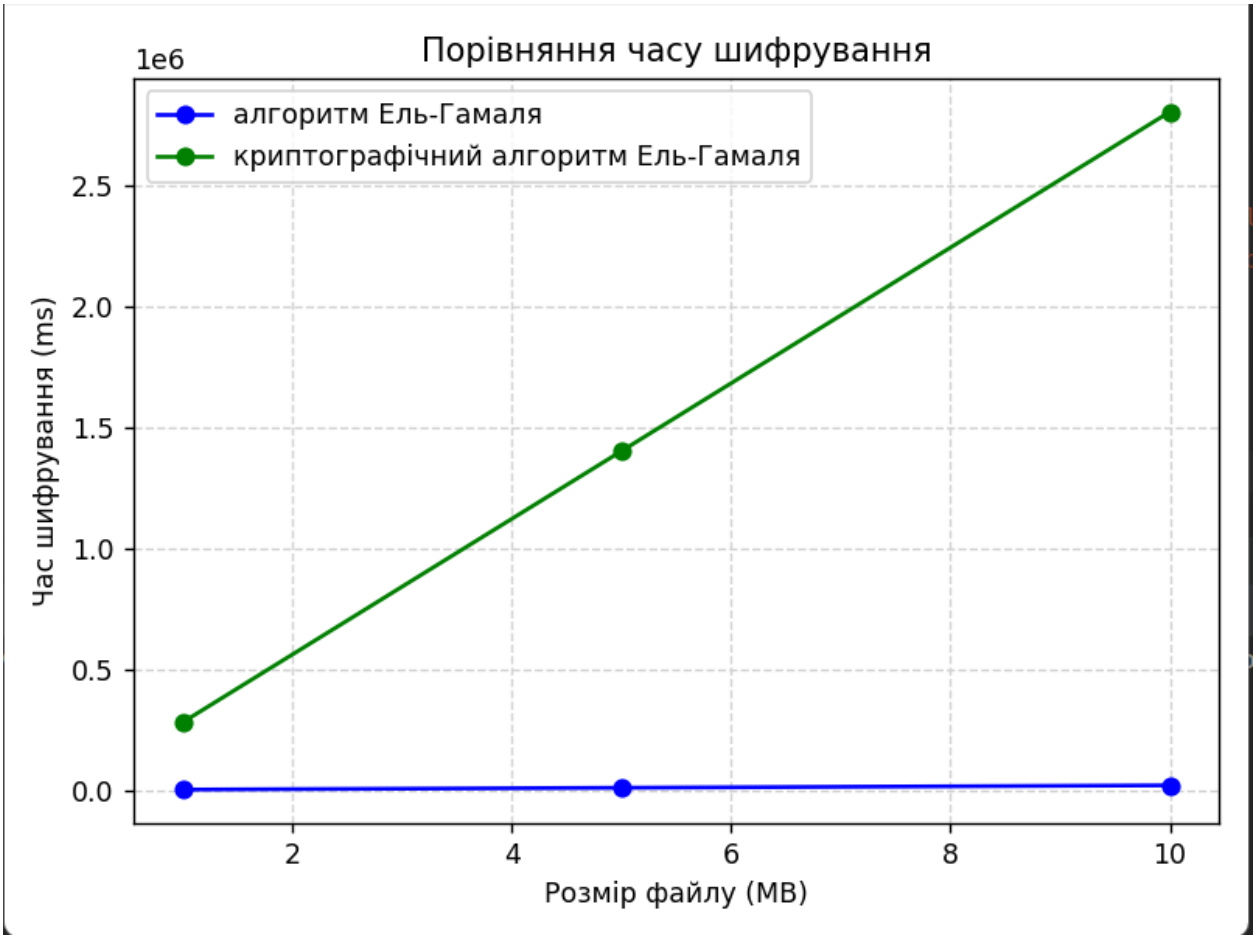

Рис. 4.17 Графік залежності часу виконання алгоритмів від розміру файла

Бачимо, що еліптичний алгоритм приблизно у 100 раз повільніший, ніж звичайний.

Теоретично, звісно, має бути навпаки: загально кажучи, Ель-Гамаль в еліптичній криптографії (EC ElGamal) має потенціал для більш швидшої роботи порівняно зі звичайним Ель-Гамалем. Це стосується особливо випадків, коли використовуються довжини ключа, які відповідають поточним рекомендаціям щодо безпеки.

Основна причина полягає у тому, що в еліптичній криптографії використовуються операції над точками на еліптичних кривих, які можуть бути виконані швидше порівняно зі стандартними арифметичними операціями, використовуваними в звичайному Ель-Гамалі.

Еліптична криптографія зазвичай використовує ключі меншої довжини, щоб забезпечити еквівалентний рівень безпеки, порівняно зі звичайними криптосистемами. Це означає, що обчислення в EC ElGamal можуть бути швидшими, оскільки вони вимагають меншої кількості операцій над числами.

Проте, ефективність алгоритмів може залежати від конкретної реалізації, використовуваних бібліотек або обчислювальних ресурсів. Швидкість виконання може різнитися в залежності від рівня оптимізації, апаратного забезпечення та конкретних вхідних даних.
Власне, маємо зворотний випадок. Через використання різних бібліотек для обох алгоритмів, співвідношення часу виконання не виправдовує очікувань.

Однак, наприклад, для досягнення 128-бітної безпеки, довжина ключа EC ElGamal може бути близько 256 біт, тоді як для звичайного Ель-Гамаля може знадобитися ключ довжиною близько 3072 біт. Тобто варто порівнювати ці 2 алгоритми з різними довжинами ключів: для рівної безпеки звичайному алгоритму потрібен ключ в 12 разів більший, ніж еліптичному. Збільшимо довжину ключа в звичайного алгоритма до 3072 біт. Маємо:
Encryption time regular for 1 mb file with 3072-bit key 367095 ms

Encryption time regular for 2 mb file with 3072-bit key 735 500 ms

Encryption time regular for 5 mb file with 3072-bit key 1 824 322 ms

Порівнюючи з вищезазначеними результатами для еліптичного алгоритму з довжиною ключа в 256 з тими ж файлами, бачимо, що еліптичний алгоритм в нашій реалізації в цілому досі повільніший, але для маленьких файлів він працює швидше:

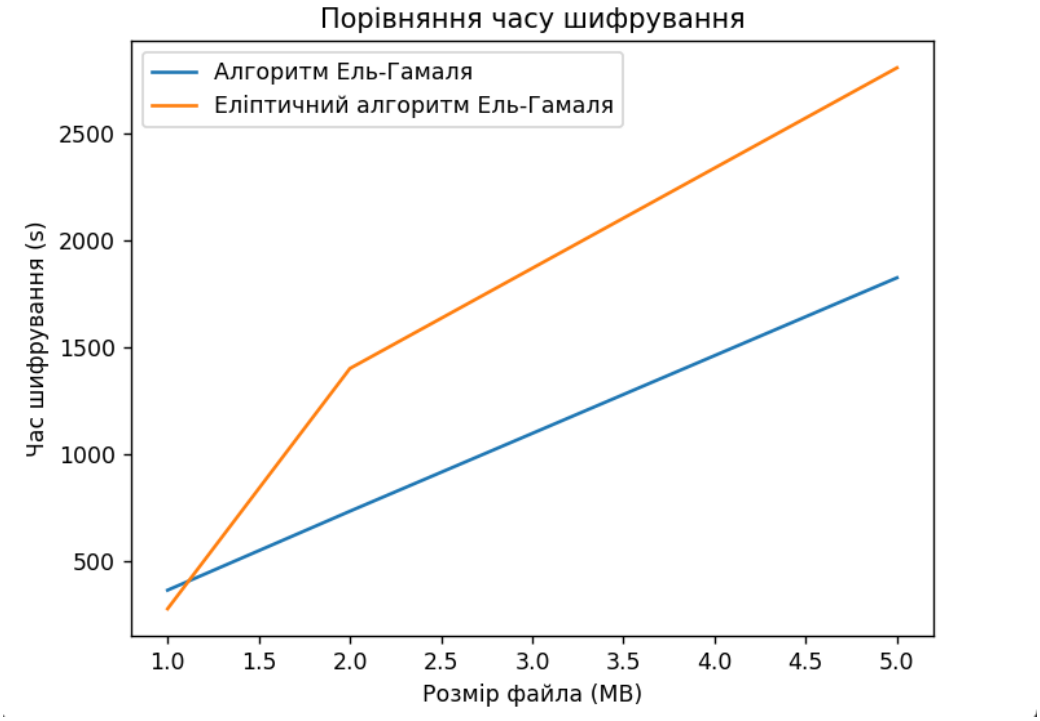

Рис. 4.18 Порівняння часу виконання алгоритмів за різних довжин ключів

4.3.2 Порівняння за стійкістю до зламу.

Алгоритм Ель-Гамаля в еліптичній криптографії (EC ElGamal) вважається більш стійким до зламу за довжиною ключа порівняно зі звичайним алгоритмом Ель-Гамаля (ElGamal) при тому ж рівні безпеки. Це обумовлено використанням еліптичних кривих, які мають властивість вищої безпеки при меншій довжині ключа порівняно з класичними арифметичними операціями над простими числами.

Зазвичай, для досягнення одного й того ж рівня безпеки, довжина ключа EC ElGamal може бути меншою, ніж довжина ключа звичайного Ель-Гамаля. Наприклад, для досягнення 128-бітної безпеки, довжина ключа EC ElGamal може бути близько 256 біт, тоді як для звичайного Ель-Гамаля може знадобитися ключ довжиною близько 3072 біт.

Безпека алгоритмів може залежати від багатьох факторів, включаючи вибір конкретних параметрів, реалізацію, атаки, які є доступними у даному часі.

Для еліптичних кривих і базових точок рішення таких рівнянь представляє дуже і дуже велику трудність. З точки ж зору криптографії, ми маємо можливість визначити нову криптографічну систему на основі еліптичних кривих. Врахуйте, що будь-яка стандартна система, заснована на проблемі дискретного логарифма, аналогічна системі заснованій на проблемі дискретного логарифма еліптичної кривої. Наприклад, Еліптична крива DSA вже стандартизована (ANSI X9.62) і на її основі може бути реалізований протокол відкритого обміну ключами Дефі– Хелмана. Для кожної еліптичної кривої кількість точок в групі звичайно, але досить велике. Оцінка порядку (кількості елементів) групи точок еліптичної кривої m така:

$$p + 1 - 2\sqrt{p} \leq m \leq p + 1 + 2\sqrt{p} ,$$

де p — порядок поля, над яким визначена крива. Якщо в схемі Эль–Гамаля рекомендується використовувати число p близько 2512, то у разі еліптичної кривої достатньо взяти p > 2255. Кратні точки еліптичної кривої є аналогом мір чисел в простому полі. Завдання обчислення кратності точки еквівалентне завданню обчислення дискретного логарифма. Власне, на складності обчислення кратності точки еліптичної кривої і заснована надійність цифрового підпису та взагалі алгоритму. Хоча еквівалентність завдання дискретного логарифмування і завдання обчислення кратності і доведена, друга має велику складність. Саме тому при побудові алгоритмів підпису в групі точок еліптичної кривої виявилося можливим обійтися коротшими ключами порівняно з простим полем при забезпеченні більшої стійкості. Секретним ключем, як і раніше, покладемо деяке випадкове число x. Відкритим ключем вважатимемо координати точки на еліптичній кривій P, визначену як P = xQ, де Q — вибрана точка еліптичної кривої (базова точка).

Координати точки Q разом з коефіцієнтами рівняння, задаючого криву, є параметрами схеми підпису і мають бути відомі усім учасникам обміну повідомленнями. Вибір точки Q залежить від використовуваних алгоритмів і дуже непростий. При побудові конкретного алгоритму, що реалізовує обчислення цифрового підпису, американський стандарт припускає використання алгоритму DSA. Деякі фахівці відмічають, що опис алгоритму цифрового підпису Эль–Гамаля на еліптичній кривій простіший і природніший. За очевидної трудності використання криптоаналізу (зламу) до алгоритму, криптографію на еліптичних кривих можна застосовувати для високо захищених систем, забезпечуючи порівнянний рівень безпеки. Алгоритм має значно менші розміри ключа, чим, наприклад, алгоритми RSA або DSA.

Використання еліптичних кривих дозволяє будувати високо захищені системи з ключами явно менших розмірів порівняно з аналогічними традиційними системами типу RSA або DSA. Такі системи менш вимогливі до обчислювальної потужності і об’єму пам’яті устаткування і тому добре підходять, наприклад, для літаків або супутників.

Довжина ключа	Ель-Гамаль	Ель-Гамаль в еліптичній криптографії
128 бітів	Добре підходить	Добре підходить
192 біти	Добре підходить	Добре підходить
256 бітів	Не рекомендується	Добре підходить
384 біти	-	Добре підходить
512 бітів	-	Добре підходить
1024 біти	-	Рекомендується
2048 бітів	-	Рекомендується

Отже, Ель-Гамаль в еліптичній криптографії (EC ElGamal) вважається більш ефективним з точки зору розміру ключів та обчислювальної складності порівняно зі звичайним Ель-Гамалем. Використання еліптичних кривих дозволяє досягти того самого рівня безпеки при коротших ключах, що зменшує обчислювальні витрати та вимоги до пам'яті.

Таким чином, якщо ми порівнюємо стійкість до атак, Ель-Гамаль в еліптичній криптографії може бути перевагою, оскільки він дозволяє досягти того ж рівня стійкості при коротших ключах.

ВИСНОВКИ

У роботі представлені види криптографічних алгоритмів, а саме симетричне та асиметричне шифрування, хеш-функції, цифрові підписи, та методів систем захисту інформації в програмному забезпеченні: IPsec, віртуальні приватні мережі. Були розглянуті та проаналізовані наявні дослідження в даній сфері, їх мета, об’єкти роботи та результати.

Описані та реалізовані на Python алгоритми Ель-Гамаля та Ель-Гамаля в еліптичній криптографії. Виконане порівняння алгоритмів за ефективністю та стійкістю до зламу, а саме за довжиною ключа. В рамках дослідження було зроблено порівняння швидкості роботи алгоритмів, що реалізують еліптичний метод і класичний. Це було зроблено шляхом заміру використаного часу при виконанні обчислень для генерації ключів та шифрування повідомлення.

Обґрунтовано підвищену безпеку еліптичного алгоритму Ель-Гамаля порівняно з класичним та використання в першому менших ключів. Складено порівняльну таблицю використання різних довжин ключів у розглянутих алгоритмах, побудовано графіки порівняння швидкості зазначених алгоритмів шифрування в залежності від розміру файлу та довжини ключа.

Були отримані такі результати: виявлено, що при використанні бібліотек cryptography та pycryptodone для еліптичного та класичного алгоритмів відповідно швидкість роботи при однакових ключах є більшою у звичайного шифрування Ель-Гамаля. При збільшенні довжини ключа в класичного алгоритму для досягнення того рівня безпеки, що і у еліптичного, тенденція залишається незмінною, але для файлів менше 1 мб швидше працює другий.

Алгоритм Ель-Гамаля в еліптичній криптографії (EC ElGamal) є більш ефективним з точки зору розміру ключів та обчислювальної складності

порівняно зі звичайним Ель-Гамалем. Використання еліптичних кривих дозволяє досягти того самого рівня безпеки при коротших ключах, що зменшує обчислювальні витрати та вимоги до пам'яті. По цій ж причині даний алгоритм є більш стійким до зламу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Новітні криптографічні методи захисту інформації [Електронний ресурс]. – Режим доступу: URL: - http://eprints.zu.edu.ua/13902/1/Mingaleva3.pdf
2. Забезпечення криптографічного захисту державних інформаційних ресурсів[Електронний ресурс]. – Режим доступу: URL: - https://eforum.lntu.edu.ua/
3. The cryptographic data transcoding operations matrix models synthesis method [Електронний ресурс – Режим доступу: URL: - https://jrnl.nau.edu.ua/index.php/ZI/article/view/3360
4. Сучасні стеганографічні методи захисту інформації [Електронний ресурс]. – Режим доступу: URL: - https://jrnl.nau.edu.ua/index.php/ZI/article/view/1994
5. What is a VPN? [Електронний ресурс] – Режим доступу: URL: - https://cpham.perso.univ-pau.fr/ENSEIGNEMENT/COMMUN/vpn_ferguson.pdf
6. Електронний (безпаперовий) документообіг. електронний цифровий підпис [Електронний ресурс] – Режим доступу: URL: - http://eui.zu.edu.ua/article/view/ISSN2410-3748-2019-1(24)-7
7. Research on Diffie-Hellman key exchange protocol [Електронний ресурс] – Режим доступу: URL: - https://ieeexplore.ieee.org/abstract/document/5485276
8. Empirical research of the distribution function of synchronization time of neural networks in the key exchange protocol [Електронний ресурс] – Режим доступу: URL: - http://journals.uran.ua/tarp/article/view/26288