МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ ЦИФРОВИХ ТЕХНОЛОГІЙ, ДИЗАЙНУ ТА ТРАНСПОРТУ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЄКТУВАННЯ ТА ДИЗАЙНУ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни «Технології захисту інформації»
для студентів усіх форм навчання
(частина 2)

Перший (бакалаврський) рівень вищої освіти
Спеціальність – 122 КОМП'ЮТЕРНІ НАУКИ
Освітня програма – КОМП'ЮТЕРНИЙ ДИЗАЙН

Одеса – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ ЦИФРОВИХ ТЕХНОЛОГІЙ, ДИЗАЙНУ ТА ТРАНСПОРТУ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЄКТУВАННЯ ТА ДИЗАЙНУ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни «Технології захисту інформації»
для студентів усіх форм навчання
(частина 2)

Перший (бакалаврський) рівень вищої освіти
Спеціальність – 122 КОМП'ЮТЕРНІ НАУКИ
Освітня програма – КОМП'ЮТЕРНИЙ ДИЗАЙН

Затверджено на засіданні
кафедри інформаційних технологій
проєктування та дизайну
Протокол № 3 від 29.10.2024

Одеса – 2024

Методичні вказівки до виконання лабораторних робіт з дисципліни «Технології захисту інформації» для студентів першого (бакалаврського) рівня за спеціальністю 122 Комп’ютерні науки, за освітньо-професійною програмою «Комп’ютерний дизайн» (частина 2) / Укл.: А. С. Коляда, О. С. Лопаков, В. В. Космачевський – Одеса : Національний університет «Одеська політехніка», 2024. – 55 с.

Укладачі:

Коляда А. С., к.т.н., доц. кафедри
Лопаков О. С., ст. викладач
Космачевський В. В., ст. викладач

Дані методичні вказівки розглядають основні аспекти криптографічного захисту інформаційних систем, які необхідно опрацювати під час виконання лабораторних робіт. Оскільки інформація є цінним ресурсом, виникають ризики несанкціонованого доступу та викривлення даних, що вимагає застосування ефективних методів захисту. Виконання лабораторних робіт спрямоване на практичне освоєння криптографічних методів захисту, зокрема забезпечення конфіденційності та цілісності даних у цифрових системах. Особливу увагу приділено впливу сучасних технологій на розвиток криптографії, методам контролю справжності інформації, а також загрозам, пов’язаним із безпаперовим документообігом, електронними платежами та цифровими архівами. Завдання лабораторних робіт передбачають вивчення та реалізацію криптографічних алгоритмів, оцінку їхньої ефективності, а також аналіз можливих атак на захищені системи. Це дозволить студентам отримати практичні навички розробки та використання сучасних засобів інформаційної безпеки.

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 8 ШИФРУВАННЯ З ВІДКРИТИМ КЛЮЧЕМ. АЛГОРИТМ RSA.

Мета роботи: Вивчити принцип роботи асиметричних алгоритмів шифрування на прикладі алгоритму RSA

8.1 Теоретичні відомості

Головна проблема використання одноключових (симетричних) криптосистем полягає в розподілі ключів. Для того, щоб був можливий обмін інформацією між двома сторонами, ключ повинен бути згенерований однією з них, а потім в конфіденційному порядку переданий іншій. Особливої гостроти ця проблема набула в наші дні, коли криптографія стала загальнодоступною, внаслідок чого кількість користувачів великих криптосистем може обчислюватися сотнями і тисячами.

Початок асиметричним шифрів було покладено в роботі «Нові напрямки в сучасній криптографії» Уитфілда Діффі і Мартіна Хеллмана, опублікованій в 1976 році. Перебуваючи під впливом роботи Ральфа Меркле (Ralph Merkle) про поширення відкритого ключа, вони запропонували метод отримання секретних ключів для симетричного шифрування, використовуючи відкритий канал. У 2002 році Хеллман запропонував називати даний алгоритм «Діффі - Хеллмана - Меркле», Визнаючи внесок Меркле в винахід криптографії з відкритим ключем. Проте робота Діффі-Хеллмана створила великий теоретичний доробок для відкритої криптографії, першою реальною криптосистемою з відкритим ключем вважають алгоритм RSA (названий по імені авторів - Рон Ривест (Ronald Linn Rivest), Аді Шамір (Adi Shamir) і Леонард Адлеман (Leonard Adleman) з Массачусетського Технологічного Інституту (MIT)).

Суть шифрування з відкритим ключем полягає в тому, що для шифрування даних використовується один ключ, а для розшифрування інший (тому такі системи часто називають асиметричними).

Основна передумова, яка привела до появи шифрування з відкритим ключем, полягало в тому, що відправник повідомлення (той, хто зашифровує повідомлення), не обов'язково повинен бути здатний його розшифровувати. Тобто навіть маючи вихідне повідомлення, ключ, за допомогою якого воно шифрувати, і знаючи алгоритм шифрування, він не може розшифрувати закриті повідомлення без знання ключа розшифрування.

Перший ключ, яким шифрується вихідне повідомлення, називається відкритим і може бути опублікований для використання всіма користувачами системи. Розшифрування за допомогою цього ключа неможливо. Другий ключ, за допомогою якого дешифрується

повідомлення, називається секретним (закритим) і повинен бути відомий тільки законному одержувачу закритого повідомлення.

8.2 Завдання на лабораторну роботу

Реалізувати шифрування та дешифрування тексту довільної довжини за допомогою асиметричного алгоритму RSA без використання зовнішніх бібліотек. Програма має включати:

- Генератор простих чисел (p і q);
- Функцію знаходження оберненого елементу по модулю за допомогою розширеного алгоритму Евкліда.

8.3 Опис способу обробки результатів

Алгоритми шифрування з відкритим ключем використовують так звані незворотні або односторонні функції. Ці функції мають наступну властивість: при заданому значенні аргументу x відносно просто обчислити значення функції (x), однак, якщо відомо значення функції y = f (x), то немає простого шляху для обчислення значення аргументу x. Наприклад, функція SIN. Знаючи x, легко знайти значення SIN (x) (наприклад, x =π, Тоді SIN (π) = 0). Однак, якщо SIN (x) = 0, однозначно визначити x не можна, тому що в цьому випадку x може бути будь-яким числом, що визначається за формулою і *π, Де і - ціле число.

Однак не всяка необоротна функція годиться для використання в реальних криптосистемах. У їх числі і функція SIN. Слід також зазначити, що в самому визначенні необоротності функції присутня невизначеність. Під необоротністю розуміється не теоретична необоротність, а практична неможливість обчислити зворотне значення, використовуючи сучасні обчислювальні засоби за доступний для огляду інтервал часу.

Тому щоб гарантувати надійний захист інформації, до криптосистем з відкритим ключем пред'являються два важливих і очевидних вимоги.

1. Перетворення вихідного тексту повинно бути умовно необоротним і виключати її відновлення на основі відкритого ключа.

2. Визначення закритого ключа на основі відкритого також повинно бути неможливим на сучасному технологічному рівні.

Усі пропоновані сьогодні криптосистеми з відкритим ключем спираються на один з наступних типів односторонніх перетворень.

1. Розкладання великих чисел на прості множники (алгоритм RSA).
2. Обчислення дискретного логарифма або дискретне зведення в ступінь (алгоритм Діффі-Хелмана-Меркле, схема Ель-Гамаля).
3. Питання про укладанні рюкзака (ранця) (автори Хелман і Меркле).
4. Обчислення коренів алгебраїчних рівнянь.
5. Використання кінцевих автоматів (автор Тао Ренжи).
6. Використання кодових конструкцій.

7. Використання властивостей еліптичних кривих.

8.4 Опис ходу експерименту

Алгоритм RSA. Стійкість RSA ґрунтується на великій обчислювальної складності відомих алгоритмів розкладання добутку простих чисел на множники. Наприклад, легко знайти добуток двох простих чисел 7 і 13 навіть подумки - 91. Спробуйте в розумі знайти два простих числа, добуток яких дорівнює 323 (числа 17 і 19). Звичайно, для сучасної обчислювальної техніки знайти два простих числа, добуток яких одно 323, не проблема. Тому для надійного шифрування алгоритмом RSA, як правило, вибираються прості числа, кількість двійкових розрядів яких дорівнює кільком сотням.

Алгоритм RSA складається з 4 етапів: генерації ключів, шифрування, розшифрування та розповсюдження ключів.

Безпека алгоритму RSA побудована на принципі складності факторизації цілих чисел. Алгоритм використовує два ключі — відкритий (public) і секретний (private), разом відкритий і відповідний йому секретний ключі утворюють пари ключів (keypair). Відкритий ключ не потрібно зберігати в таємниці, він використовується для шифрування даних. Якщо повідомлення було зашифровано відкритим ключем, то розшифрувати його можна тільки відповідним секретним ключем.

Генерація ключів

Для того, щоб згенерувати пари ключів виконуються такі дії:

1. Вибираються два великі прості числа p і q

2. Обчислюється їх добуток $n = pq$

3. Обчислюється функція Ейлера $\varphi(n) = (p-1)(q-1)$

4. Вибирається ціле число e таке, що $1 < e < \varphi(n)$ та e взаємно просте з $\varphi(n)$

5. За допомогою розширеного алгоритму Евкліда знаходиться число d таке, що $ed \equiv 1 \pmod{\varphi(n)}$

Число n називається модулем, а числа e і d — відкритою й секретною експонентами (англ. encryption and decryption exponents), відповідно. Пари чисел (n, e) є відкритою частиною ключа, а (n, d) — секретною. Числа p і q після генерації пари ключів можуть бути знищені, але в жодному разі не повинні бути розкриті.

Шифрування

Припустимо, що Боб хотів би відправити повідомлення M Алісі. Спочатку він перетворює M в ціле число m так, щоб $0 \leq m < n$ за допомогою узгодженого оборотного протоколу, відомого як схеми доповнення. Потім він обчислює зашифрований текст c, використовуючи відкритий ключ Аліси e, за допомогою рівняння:

$$c = m^e \bmod n$$

Це може бути зроблено досить швидко, навіть для 500-бітних чисел, з використанням модульного зведення в ступінь. Потім Боб передає c Алісі.

Розшифрування

Для розшифрування повідомлення Боба m Алісі потрібно обчислити таку рівність:

$$m = c^d \bmod n$$

№	Опис операції	Приклад
1	Вибирається два простих числа **p** і **q**	p=7, q=13
2	Розраховується добуток **n = p * q**	n=91
3	Розраховується функція Ейлера **ϕ(n)=(p-1)(q-1)** Результат дорівнює кількості позитивних чисел, які не перевищують **n** і взаємно прості з **n**	ϕ(n) = (7-1)(13-1) = 91-7-13+1
4	Вибирається довільне число **e** (0<e<n), взаємно просте з результатом функції Ейлера. (Відкритий ключ)	e=5
5	Розраховується секретний ключ **d** із відношення **(d*e) mod ϕ(n) = 1**	(d*5) mod 72 = 1, d = 29
6	Публікуються відкриті ключі **e** та **n** в спеціальному сховищі, де виключається можливість підміни	

Примітки. Просте число - натуральне число, більше одиниці і не має інших дільників, крім самого себе і одиниці. Взаємно прості числа - числа, які не мають спільних дільників, крім 1 (наприклад, 2 і 3 — взаємно прості, а 2 і 4 — ні, тому що діляться на 2).
Процедури шифрування і дешифрування виконуються за такими формулами

$C = T^e \bmod n$,

$T = C^d \bmod n$.

де T, C - числові еквіваленти символів відкритого і шифрованого повідомлення.

Слід зазначити, що p і q вибираються таким чином, щоб n було більше коду будь-якого символу відкритого повідомлення. В автоматизованих системах вихідне повідомлення переводитися в двійкове подання, після чого шифрування виконується над

блоками біт, рівної довжини. При цьому довжина блоку повинна бути менше, ніж довжина двійкового представлення n.

У висновку слід зазначити стійкість даного алгоритму. У 2003 р Аді Шамір і Еран Тромер розробили схему пристрою TWIRL, яке при вартості $ 10 000 може дешифрувати 512-бітний ключ за 10 хвилин, а при вартості $ 10 000 000 - 1024-бітний ключ менше, ніж за рік. В даний час Лабораторія RSA рекомендує використовувати ключі розміром 2048 бітів.

8.5 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

8.6 Контрольні запитання

- Що таке шифрування з відкритим ключем, і які основні його переваги в порівнянні з симетричним шифруванням?
- Які математичні принципи лежать в основі алгоритму RSA? Як використовується проблема факторизації великих чисел для забезпечення його безпеки?
- Як у RSA відбувається генерація пари ключів? Які етапи включає цей процес?
- Як працює алгоритм RSA у процесі шифрування та дешифрування даних? У чому полягає роль відкритого та закритого ключів?
- Які основні вразливості алгоритму RSA? Як можна забезпечити додаткову стійкість до атак?

8.7 Література

- Rivest R. L., Shamir A., Adleman L. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* // Communications of the ACM. – 1978. – Vol. 21, No. 2. – P. 120–126.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
- Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.
- Boneh D. *Twenty Years of Attacks on the RSA Cryptosystem* // Notices of the AMS. – 1999. – Vol. 46, No. 2. – P. 203–213.

ЛАБОРАТОРНА РОБОТА №9 Обмін ключами за схемою

Діффі-Хеллмана

Мета роботи: Освоїти обмін ключами за схемою Діффі-Хеллмана, вивчаючи проблему первісних коренів. Вивчити методи знаходження простих чисел.

9.1 Теоретичні відомості

Протокол Діффі-Хеллмана (Diffie-Hellman Key Exchange) — це криптографічний протокол, який дозволяє двом сторонам спільно генерувати секретний ключ через небезпечний канал зв'язку. Цей секретний ключ використовується для подальшого шифрування повідомлень. Протокол заснований на складності обчислення дискретного логарифму в великих простих полях. У 1976 році була опублікована робота американських математиків У. Діффі і М.Е. Хеллмана «Нові напрямки в криптографії», в ній вони запропонували конкретну конструкцію так званого "відкритого розподілу ключів". Мета алгоритму полягає в тому, щоб два учасники могли безпечно обмінятися ключем, який в подальшому може використовуватися в будь-якому алгоритмі симетричного шифрування. Сам алгоритм Діффі-Хеллмана може застосовуватися тільки для обміну ключами. Алгоритм заснований на труднощі обчислень дискретних логарифмів. Безпека обміну ключа в алгоритмі Діффі-Хеллмана випливає з того факту, що, хоча відносно легко обчислити експоненти по модулю простого числа, дуже важко обчислити дискретні логарифми. Для великих простих чисел завдання вважається нерозв'язною.

9.2 Завдання на лабораторну роботу

Реалізувати програму, яка демонструвала б, як проводиться обмін ключами за схемою Діффі-Хеллмана:

- Прості числа X_A, X_B і q отримувати випадковим чином за допомогою генератора простих чисел на основі тесту Рабіна-Міллера;
- Реалізувати та використати функцію знаходження числа Ейлера;
- Реалізувати та використати функцію знаходження первісного кореня

9.3 Опис способу обробки результатів

Прості числа

Простих чисел не так мало, як здається, наприклад, існує приблизно 10^{151} простих чисел довжиною від 1 біта до 512 включно. Для чисел близьких n, ймовірність того, що вибране число виявиться простим, дорівнює 1 / ln n. Тому повне число простих чисел, менших n одно n / ln n. Вважається, що ймовірність вибору двома людьми одного і того ж великого простого числа нехтує мала.

Існують різні імовірнісні перевірки на простоту чисел, що визначають чи є число простим з заданим ступенем достовірності. За умови, що ця ступінь достовірності

достатньо велика, такі способи досить гарні. Такі прості числа часто називають «промисловими простими», тобто вони прості з контрольованою можливістю помилки. Повсюдно використовуються є алгоритм, розроблений Майклом Рабіном по ідеям Гарі Міллера.

Тест Rabin-Miller

Виберіть для перевірки випадкове число p. Обчисліть b як найбільше число поділок p-1 на 2 тобто 2^b - найбільша ступінь числа 2, на яке ділиться p-1. Потім обчисліть m, таке, що $p = 1 + 2^b m$

1. Виберіть випадкове число a, менше p.
2. Встановіть j = 0 і $z = a^m \bmod p$
3. Якщо z = 1 або якщо z = p-1, то p проходить перевірку і може бути простим числом.
4. Якщо j> 0 і z = 1, то p не є простим числом.
5. Встановіть j = j + 1.

Якщо j <b і z <p-1, встановіть z = z2 mod p і поверніться на ④.

Якщо z = p-1, то p проходить перевірку і може бути простим числом.

6. Якщо j = b і z≠*p*-1, то p не є простим числом.

Повторити цю перевірку потрібно t раз.

Доведено, що в цьому тесті ймовірність проходження перевірки складовим числом убуває швидше, ніж в інших. Гарантується, що 75% можливих значень, a виявляться показниками того, що вибране число p складене. Це означає, що ймовірність прийняти складене число p за просте не перевищує величини (¼) ^{t}.

Псевдокод:

МІЛЛЕР-РАБІН(n, k)

1. **якщо** n парне **тоді**
2. **повернути** ХИБА
3. $m \leftarrow (n-1) \operatorname{div} 2; t \leftarrow 1$
4. **поки** m парне
5. $m \leftarrow m \operatorname{div} 2; t \leftarrow t+1$
6. **для** $i \leftarrow 1$ **до** k
7. $a \leftarrow Random() \mod n$
8. $u \leftarrow a^m \mod n$
9. **якщо** $u \neq 1$ **тоді**
10. $j \leftarrow 1$
11. **поки** $u \neq n-1$ і $j < t$
12. $u \leftarrow u^2 \mod n; j \leftarrow j+1$
13. **якщо** $u \neq n-1$ **тоді**
14. **повернути** ХИБА
15. **повернути** ІСТИНА

Генерація простого числа

1. Згенеруйте випадкове n-бітове число p.

2. Встановіть його старший і молодший біти рівними 1. Старший біт гарантуватиме необхідну довжину шуканого числа, а молодший біт забезпечує його непарність.

3. Переконайтеся, що p не ділиться на невеликі прості числа: 3, 5, 7, 11 і т.д. Найбільш ефективною є перевірка на подільність для всіх простих чисел, менших 2000.

4. Виконайте тест Rabin-Miller мінімум 5 разів.

Якщо p не пройшло хоча б одну перевірку з 3 або 4, Воно не є простим.

Перевірка, що випадкове непарне p не ділиться на 3, 5 і 7 відсікає 54% непарних чисел. Перевірка подільності на всі прості числа, менші 256 відсікає 80% складових непарних чисел.

Навіть, якщо складене число «просочилося» через цей алгоритм, це буде відразу ж помічено, тому що шифрування і дешифрування не працюватимуть.

Первісна (примітивний) корінь по модулю n

В силу теореми Ейлера, для будь-яких взаємно простих, a й n виконується співвідношення

$a^{\varphi(N)} \equiv 1 \bmod n,$ (9.1)

де $\varphi(n)$позначає функцію Ейлера, значення якої дорівнює числу позитивних цілих значень, менших n і взаємно простих з n. Для простого числа p

$\varphi(p) = p-1,$

Якщо припустити, що два числа p і q прості, тоді для $n = p^{\alpha} * q^{\beta}$ функція Ейлера матиме вигляд

$\varphi(n) = \varphi(p^{\alpha} * q^{\beta}) = \varphi(p^{\alpha}) * \varphi(q^{\beta}) = [(p-1)\, p^{\alpha-1}] * [(q-1)\, q^{\beta-1}].$ (9.2)

Розглянемо більш загальне співвідношення, ніж (9.1). Кажуть, що число a, взаємно просте з модулем n, належить показнику m, якщо m - таке найменше натуральне число, що виконується порівняння

$a^m \equiv 1 \bmod n.$ (9.3)

Якщо a і n є взаємно простими, то існує, принаймні, одне число $m = \varphi(n)$, Що задовольняє (9.3). Найменше з позитивних чисел m, для яких виконується (9.3), є довжиною періоду послідовності, що генерується ступенями a.

Справедливі наступні властивості.

Властивість 1. Числа $a^0, a^1, ..., a^{m-1}$ попарно непорівнянні по модулю n.

Дійсно, $a^l \equiv a^k \bmod n$, $L > k \Rightarrow a^{l-k} \equiv 1 \bmod n$, Де $l - k \in N$, $l - k < m$.

Властивість 2. $a^{\gamma} \equiv a^{\gamma'} \bmod n \Leftrightarrow \gamma \equiv \gamma' \bmod m$. розділимо$\gamma$, γ' на m із залишками $\gamma = Mq + r$, $\gamma' = Mq' + r'$. тоді $a^{\gamma} \equiv a^{\gamma'} \Leftrightarrow amq + r \equiv amq' + r' \Leftrightarrow ar \equiv ar' \Leftrightarrow r' = r$. Звідси випливає наступна властивість.

Властивість 3. m | *φ(n)*. Число, що належить показнику*φ(n)*, Називається первісним коренем за модулем *n*.

Властивість 4. За будь-якій простій модулю p існує первісний корінь. Первісні корені існують по модулях 2, 4, p^{α}, $2p^{\alpha}$, Де p - непарне просте, $\alpha \in N$.

Властивість 5. Нехай c = *φ(n)*і q1, q2, ..., q_k - різні прості дільники числа c. Число a, взаємно просте з модулем n, буде первісним коренем тоді і тільки тоді, коли не виконана жодна з наступних порівнянь:

$$a^{c/q_1} \equiv 1 \bmod n, a^{c/q_2} 1 \bmod n, \ldots, a^{c/q_k} 1 \bmod n$$

Необхідність випливає з того, що $a^{\varphi(n)} \equiv 1 \bmod n$ і порівняння не має місця при менших показниках ступеня. Назад, припустимо, що a не задовольняє жодному з порівнянь, і нехай a належить показнику m <c. Тоді m | c⇒*c = mn*. Позначимо через q простий дільник u. Тоді легко отримати протиріччя:

$a^{c/q} = a^{mu/q} = (am)^{u/q} \equiv 1 \bmod n$.

Якщо деяка послідовність має довжину *φ(n),*тоді ціле число a генерує своїми ступенями безліч всіх ненульових лишків за модулем *n*. Таке ціле число називають первісним коренем числа *a* по модулю *n*. Кількість їх одно для числа *n*

φ(N-1), (9.4)

де *φ* - функція Ейлера.

Приклад (Перевірка Властивості 5.)Нехай n = 41. Маємо c =φ(41) = 40 = 23 · 5. Отже, первісний корінь не повинен задовольняти двом порівнянь

$a^{8} \equiv 1 \pmod{41}$, $a^{20} \equiv 1 \pmod{41}$.

Спробуємо числа 2, 3, 4, …: $2^8 \equiv 10$, $2^{20} \equiv 1$, $3^8 \equiv 1$, $4^8 \equiv 18$, $4^{20} \equiv 1$, $5^8 \equiv 18$, $5^{20} \equiv 1$, $6^8 \equiv 10$, $6^{20} \equiv 40$. Звідси бачимо, що 6 є найменшим первісним коренем за модулем 41.

9.4 Опис ходу експерименту

Припустимо, що двом абонентам необхідно провести конфіденційне листування, а в їхньому розпорядженні немає спочатку обумовленого секретного ключа. Однак між ними існує канал, захищений від модифікації, тобто дані, що передаються по ньому, можуть бути прослухані, але не змінені (такі умови мають місце досить часто). У цьому випадку дві сторони можуть створити однаковий секретний ключ, ні разу не передавши його по мережі, за наступним алгоритмом.

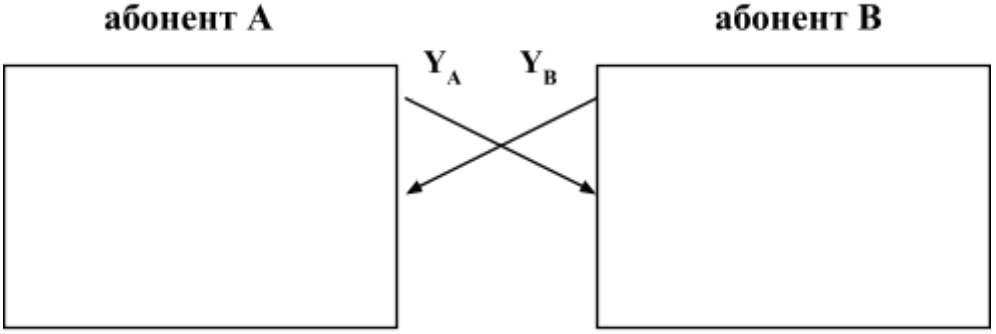


Рис. 1.Обмін ключами за схемою Діффі - Хеллмана.

Алгоритм полягає в наступному:

Глобальні відкриті елементи:

q - просте число

a - первісний корінь q

Обчислення ключа абонентом А:

вибирається секретне число X_A (X_A <q)

обчислення відкритого значення Y_A: $Y_A = a^{X_A} (\mathrm{mod}\, q)$

Обчислення ключа абонентом В:

вибирається секретне число X_B (X_B <q)

обчислення відкритого значення Y_B: $Y_B = a^{X_B} (\mathrm{mod}\, q)$

Обчислення секретного ключа абонентом А: $K = (Y_B)^{X_A} (\mathrm{mod}\, q)$

Обчислення секретного ключа абонентом В: $K = (Y_A)^{X_B} (\mathrm{mod}\, q)$

Необхідно ще раз відзначити, що алгоритм Діффі-Хеллмана працює тільки на лініях зв'язку, надійно захищених від модифікації. Якби він був застосуємо на будь-яких відкритих каналах, то давно зняв би проблему поширення ключів і, можливо, замінив собою всю асиметричну криптографію.

Приклад:

Нехай $q = 97$ і $a = 5$

Абонент А: згенерувати випадкове число $X_A = 36$

Абонент В: згенерувати випадкове число $X_B = 58$

Ці елементи вони тримають в секреті. Далі кожен з них обчислює новий елемент:

$Y_A = 5^{36} (\mathrm{mod}\, 97) = 50$, $Y_B = 5^{58} (\mathrm{mod}\, 97) = 44$

Потім вони обмінюються цими елементами по каналу зв'язку. Тепер абонент А, отримавши Y_B і знаючи свій секретний елемент X_A, обчислює загальний ключ: $K_A = 44^{36} (\mathrm{mod}\, 97) = 75$.

Аналогічно надходить абонент В: $K_B = 50^{58} (\mathrm{mod}\, 97) = 75$

9.5 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

9.6 Контрольні запитання

- У чому полягає основна ідея обміну ключами за схемою Діффі-Хеллмана?
- Які загальні параметри використовуються в схемі Діффі-Хеллмана, і як вони обираються?
- Як обчислюється спільний секретний ключ в учасників протоколу?
- Які основні вразливості має схема Діффі-Хеллмана? Як можна захиститися від атак "людина посередині"?
- У яких сучасних протоколах застосовується обмін ключами за схемою Діффі-Хеллмана?

9.7 Література

- Diffie W., Hellman M. *New Directions in Cryptography* // IEEE Transactions on Information Theory. – 1976. – Vol. 22, No. 6. – P. 644–654.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
- Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
- Koblitz N. *A Course in Number Theory and Cryptography*. – Springer, 1994. – 250 p.

ЛАБОРАТОРНА РОБОТА №10 АЛГОРИТМ ШИФРУВАННЯ ЕЛЬ-ГАМАЛЯ

Мета роботи: Дослідження структури алгоритму і методики практичної реалізації шифрування алгоритмом Ель-Гамаля.

10.1 Теоретичні відомості

Схема Ель-Гамаля (ElGamal) — криптосистема з відкритим ключем, яку засновано на складності обчислення дискретних логарифмів у скінченному полі. Криптосистема включає у себе алгоритм шифрування і алгоритм цифрового підпису. Схема Ель-Гамаля лежить в основі колишніх стандартів електронного цифрового підпису в США (DSA). Схему запропонував Тахер Ель-Гамаль 1985 року. Ель-Гамаль розробив один з варіантів алгоритму Діффі-Геллмана. Він удосконалив систему Діффі-Геллмана й отримав два алгоритми, які призначено для шифрування та для автентифікації. На відміну від RSA, алгоритм Ель-Гамаля не запатентовано, і тому він став дешевшою альтернативою, оскільки оплата внесків за ліцензію не потрібна. Вважається, що алгоритм потрапляє під дію патенту Діффі-Геллмана.

10.2 Завдання на лабораторну роботу

Реалізувати шифрування та дешифрування тексту довільної довжини за допомогою асиметричного алгоритму Ель Гамаля без використання зовнішніх бібліотек. Програма має включати:

- Генератор простих чисел (p);
- Функцію знаходження оберненого елементу по модулю за допомогою розширеного алгоритму Евкліда.

10.3 Опис способу обробки результатів

Асиметрична схема Ель Гамаль, запропонована автором (El Gamal), використовує операцію піднесення до степеня за модулем простого числа. При цьому важкою задачею для зловмисника є відшукання не числа, яке зведено в ступінь, а в який ступінь зведено відоме число. Це завдання носить назву проблеми дискретного логарифма.

Параметри:

p - велике просте число;

g - ціле число (примітивний корінь за модулем p)

C - випадкове число, 1 <C <(p-1);

$D = g^C mod p$;

x - відкритий текст;

якщо $|x| > |p|$, Тоді m розбивається на блоки і кожен блок шифрується окремо.

Відкритий ключ: D;

Секретний ключ: C.

Шифрування:

1. Вибрати k - випадкове число, $1 \leq k \leq p - 2$;

2. Обчислити $r = g^k mod p$;

3. Обчислити $e = x \cdot D^k mod p$;

4. Пара (r, e) - шифрований текст.

Розшифрування:

обчислити $x^{'} = e \cdot (r^C)^{-1} mod p$

10.4 Опис ходу експерименту

На етапі вироблення ключів:

1. Вибирається довільне (правда досить велике) просте число p.

2. Для цього простого числа визначається примітивний корінь (Англ. Primitive root) - таке число a, при багаторазовому зведенні якого в степеня за модулем p (a^1 mod p, a^2 mod p, ...), будуть перебиратися (в довільному порядку, але обов'язково по одному разу) всі числа від 1 до (p-1) включно.

3. Генерується довільне випадкове число x (0 <x <p) - це і є
закритий ключ.

4. Обчислюється значення b = a^x mod p - комбінація (a, p, b) являє собою відкритий ключ одержувача.

На етапі шифрування:

1. Відправник генерує довільне випадкове число y (0 <y <p).

2. поміщає на початку шифрограми число (a^y mod p).

3. Обчислює величину k = (b^y mod p) = ((a^x mod p) ^{y} mod p).

4. Використовуючи деяку, заздалегідь обумовлену в даній реалізації, частину k як симетричного ключа для будь-якого блочного шифру шифрує і відправляє повідомлення.

5. Надійно стирає числа y і k з оперативної пам'яті і інших місць, куди вони могли випадково потрапити.

На етапі дешифрування:

1. По приходу зашифрованого повідомлення одержувач відокремлює від пакета величину (a^y mod p) і обчислює на її основі ((a^y mod p) ^{x} mod p) - математика доводить, що отримане число буде дорівнює тому самому k, яке обчислив відправник, так як в цій формулі операнди x і y можна міняти місцями.

2. Виділивши з k ту саму частину, що і відправник, отримувач дешифрує залишок пакету симетричним алгоритмом.

Схема алгоритму Ель Гамаля приведена на рис. 1.

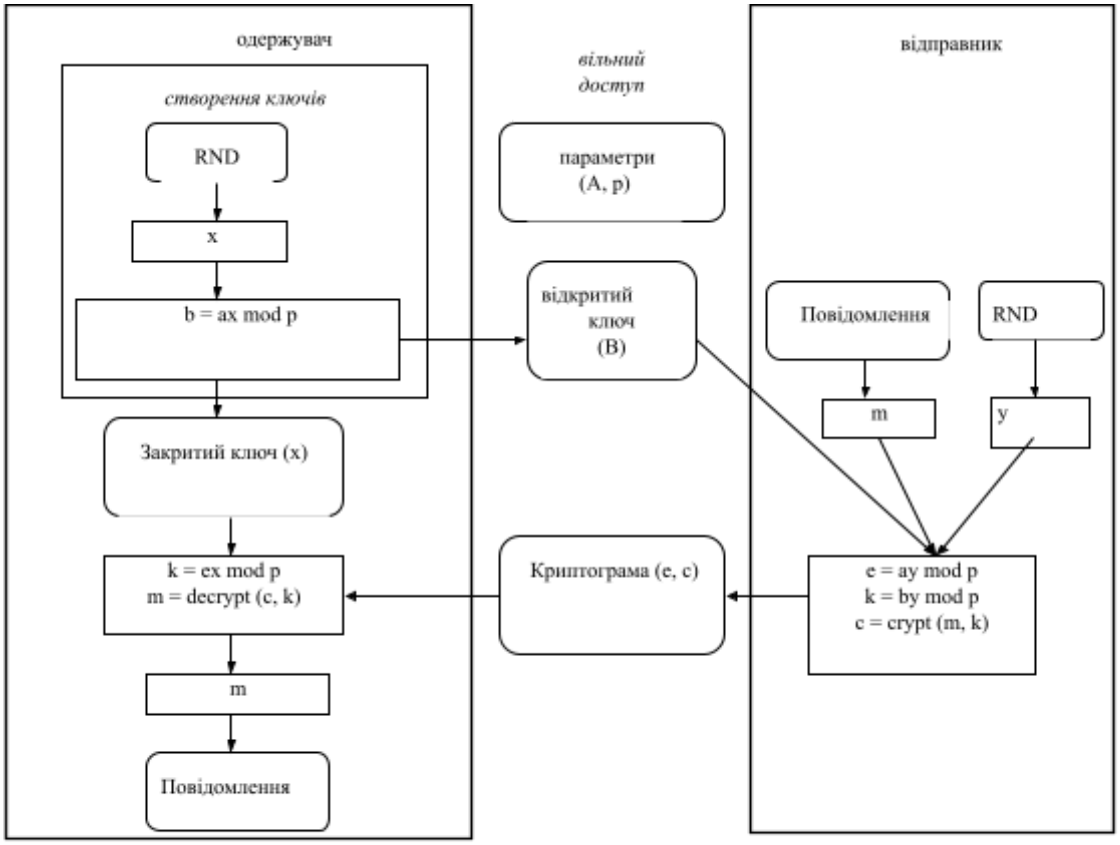


Рис. 1. Криптосистема Ель Гамаля

Проблема дискретного логарифма полягає в тому, що, знаючи основу ступеня і значення після зведення результату по модулю простого числа, неможливо за доступне для огляду час визначити, в яку саме ступінь було зведено основу. У схемі Ель Гамаль потенційний зловмисник може отримати значення a, p, (a^x mod p) і (a^y mod p). Однак через складність визначення чисел x і y "в чистому вигляді" у нього не виявляється можливо розрахувати значення k = (a^{xy} mod p), яке необхідно для розшифрування.

За криптостійкістю в схемі Ель Гамаль 512-бітове число p прирівнюється до 56-бітного симетричного ключа, розмір якого в даний час є недостатнім для надійного шифрування. Тому на практиці застосовуються p довжиною в 768, 1024 і 1536 біт.

Приклад. В якості простого числа, що породжує циклічну групу, виберемо p = 11, за який утворює елемент приймемо число a = 7 (при зведенні 7 у ступінь 1, 2, 3 і т. д. по модулю 11 послідовно проходять всі 10 значень [7, 5 , 2, 3, 10, 4, 6, 9, 8, 1]). Секретним ключем x виберемо 6, параметр b приймає значення b = (a^x mod p) = (7^6 mod 11) = 4. У цілому ключ приймає вид (a = 7, p = 11, b = 4).

Припустимо, що якийсь абонент хоче передати повідомлення. Він вибрав випадкове число, яке не перевищує p, наприклад, y = 9. На початок шифрограми поміщається число (a^y mod p) = 79 mod 11) = 8. Крім того, на основі y і відкритого ключа відправник обчислює

$k = b^y \bmod p = 49 \bmod 11 = 3$. Вибравши значення 3 або будь-які його біти в якості симетричного ключа, відправник шифрує дані, що передаються і стирає величини 9 і 3 зі своїх накопичувачів.

Одержувач по приходу пакету для обчислення $k = (a^y \bmod p) \times \bmod p$ зводить число 8 з заголовка шифрограми в ступінь секретного ключа і отримує $k = 8^6 \bmod 11 = 3$ - те ж саме значення, яке використовував відправник, шифруючи власне дані.

10.5 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

10.6 Контрольні запитання

- У чому полягає основна ідея алгоритму шифрування Ель Гамаля, і на якій математичній задачі базується його безпека?
- Як у алгоритмі Ель Гамаля відбувається генерація відкритого та закритого ключів? Які математичні обчислення для цього використовуються?
- Як працює алгоритм шифрування Ель Гамаля під час шифрування та дешифрування повідомлень?
- Які основні вразливості має алгоритм Ель Гамаля, і як їх можна мінімізувати?
- У яких криптографічних системах застосовується алгоритм Ель Гамаля? Які його переваги та недоліки в порівнянні з RSA?

10.7 Література

- ElGamal T. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms* // IEEE Transactions on Information Theory. – 1985. – Vol. 31, No. 4. – P. 469–472.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
- Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.
- Katz J., Lindell Y. *Introduction to Modern Cryptography*. 3rd ed. – CRC Press, 2020. – 644 p.

ЛАБОРАТОРНА РОБОТА № 11 КРИПТОГРАФІЧНІ ХЕШ-ФУНКЦІЇ.

СІМЕЙСТВО АЛГОРИТМІВ SHA.

Мета роботи: Вивчити існуючі алгоритми сімейства SHA обчислення дайджестів повідомлень і написати програму, що реалізує заданий алгоритм хешування.

11.1 Теоретичні відомості

Хешування(Іноді хешування, англ. Hashing) - перетворення вхідного масиву даних довільної довжини в вихідну бітову рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями або функціями згортки, а їх результати називають хешем, хеш-кодом або дайджестом повідомлення (англ. Message digest).

Хешування застосовується для порівняння даних: якщо у двох масивів хеш-коди різні, масиви гарантовано розрізняються; якщо однакові - масиви, швидше за все, однакові. У загальному випадку однозначної відповідності між вихідними даними і хеш-кодом немає в силу того, що кількість значень хеш-функцій менше, ніж варіантів вхідного масиву. Також існує безліч масивів, що дають однакові хеш-коди - так звані колізії. Імовірність виникнення колізій відіграє важливу роль в оцінці якості хеш-функцій.

Існує безліч алгоритмів хешування з різними характеристиками (розрядність, обчислювальна складність, криптостойкость і т.п.). Вибір тієї чи іншої хеш-функції визначається специфікою розв'язуваної задачі. Найпростішими прикладами хеш-функцій можуть служити контрольні суми.

контрольні суми- це нескладні, вкрай швидкі і легко реалізовані апаратні алгоритми, використовувані для захисту від ненавмисних спотворень, в тому числі помилок апаратури. За швидкістю обчислення в десятки і сотні разів швидше, ніж криптографічні хеш-функції, і значно простіше в апаратної реалізації.

Платою за таку високу швидкість є відсутність криптостойкости - легка можливість підігнати повідомлення під заздалегідь відому суму. Розрядність контрольних сум (зазвичай 32 біта) нижче, ніж криптографічних хеш (типові значення - 128, 160 і 256 біт), що означає можливість виникнення ненавмисних колізій.

Найпростішим випадком такого алгоритму є розподіл повідомлення на 32- або 16-бітові слова і їх підсумовування, що застосовується, наприклад, в TCP / IP.

Як правило, до такого алгоритму пред'являються вимоги відстеження типових апаратних помилок, таких, як кілька поспіль помилкових біт до заданої довжини. До таких алгоритмам відноситься, наприклад, CRC32, застосовуваний в апаратурі Ethernet і в форматі упакованих файлів ZIP.

Серед безлічі існуючих хеш-функцій виділяють криптографически стійкі, що застосовуються в криптографії. Для того, щоб хеш-функція H вважалася

криптографически стійкою, вона повинна задовольняти трьом основним вимогам, на яких засновано більшість застосувань хеш-функцій в криптографії:

- ***незворотність***: Для заданого значення хеш-функції m повинно бути обчислювально нездійсненно знайти блок даних X, для якого H (X) = m.
- ***стійкість до колізій першого роду***: Для заданого повідомлення M має бути обчислювально нездійсненно підібрати інше повідомлення N, для якого H (N) = H (M).
- ***стійкість до колізій другого роду***: Має бути обчислювально нездійсненно підібрати пару повідомлень (M, M '), що мають однаковий хеш.

Слід зазначити, що не доведене існування незворотних хеш-функцій, для яких обчислення будь-якого прообразу заданого значення хеш-функції теоретично неможливо. Зазвичай знаходження зворотного значення є лише обчислювально складним завданням.

Для криптографічних хеш-функцій також важливо, щоб при щонайменшій зміні аргументу значення функції сильно змінювалося (це властивість називається лавинних ефектом). Зокрема, значення хеша не повинно давати витоку інформації навіть про окремі біти аргументу. Ця вимога є запорукою криптостойкости алгоритмів, хешірующіх пароль користувача для отримання ключа.

11.2 Завдання на лабораторну роботу

Реалізувати програму, що дозволяє обчислювати значення хеш-функції, заданої у варіанті:

- текст повідомлення повинен зчитуватися з файлу;
- отримане значення хеш-функції має представлятися в шістнадцятковому вигляді і зберігатися в файл;

11.3 Опис способу обробки результатів

Сімейство алгоритмів SHA (Secure hash standard) включає в себе 5 алгоритмів обчислення хеш-функції: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512. Чотири останні хеш-функції об'єднуються в підродина SHA-2. Алгоритм SHA-1 був розроблений Агентством національної безпеки США (NSA) в 1995 році. Алгоритми підродини SHA-2 також були розроблені Агентством національної безпеки США і опубліковані Національним інститутом стандартів і технологій у федеральному стандарті обробки інформації FIPS PUB 180-2 в серпні 2002 року. Ці алгоритми використовуються в SSL, SSH, S / MIME, DNSSEC, X.509, PGP, IPSec, при передачі файлів по мережі (BitTorrent).

Між собою алгоритми відрізняються криптостійкості, яка забезпечується для хешіруемих даних, а також розмірами блоків і слів даних, що використовуються при хешування. Основні відмінності алгоритмів можна представити у вигляді таблиці 1:

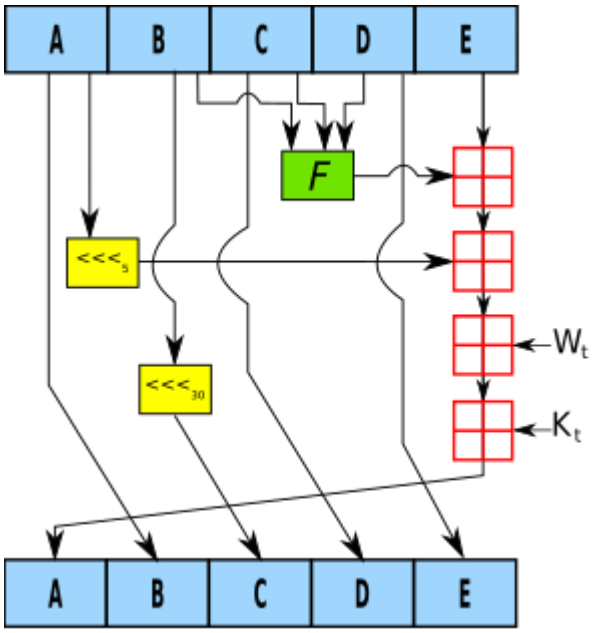
Таблиця 1 - Властивості алгоритмів SHA

Алгоритм	Довжина дайджесту повідомлення (біт)	Довжина внутрішнього стану (біт)	Довжина блоку (біт)	Довжина повідомлення (біт)	Довжина слова (біт)	Кількість ітерацій в циклі
SHA-1	160	160	512	$<2^{64}-1$	32	80
SHA-224	224	256	512	$<2^{64}-1$	32	64
SHA-256	256	256	512	$<2^{64}-1$	32	64
SHA-384	384	512	1024	$<2^{128}-1$	64	80
SHA-512	512	512	1024	$<2^{128}-1$	64	80

11.4 Опис ходу експерименту

Алгоритм SHA-1

Secure Hash Algorithm 1 — алгоритм криптографічного хешування. Описано в RFC 3174. Для вхідного повідомлення довільної довжини $2^{64}-1$ біт. Алгоритм генерує 160-бітове хеш-значення, відоме також дайджестом повідомлення.



Одна ітерація алгоритму SHA-1

SHA-1 реалізує хеш-функцію, побудовану на ідеї функції стиснення. Входом функції стиснення є блок повідомлення довжиною 512 біт і вихід попереднього блоку повідомлення. Виходом є значення всіх хеш-блоків до цього моменту. Іншими словами хеш блоку M_i дорівнює $h_i = f(M_i, h_{i-1})$. Хеш-значенням всього повідомлення є виходом останнього блоку.

Вхідне повідомлення розбивається на блоки по 512 біт у кожному. Останній блок доповнюється до довжини кратної 512 біт. Спочатку додається 1, а потім нулі, щоб довжина блоку стала рівною (512-64=448) біт. В останні 64 біта записується довжина вихідного повідомлення у бітах (в big-endian форматі). Якщо останній блок має довжину понад 448, але менше 512 біт, то додаток виконується в такий спосіб: спочатку додається 1, потім нулі аж до кінця 512-бітного блоку; після цього створюється ще один 512-бітний блок, який заповнюється аж до 448 біта нулями, після чого в 64 біта, що залишилися, записується довжина вихідного повідомлення в бітах (в big-endian форматі). Доповнення останнього блоку здійснюється завжди, навіть якщо повідомлення вже має потрібну довжину.

Ініціалізуються п'ять 32-бітових змінних:

A = a = 0x67452301
B = b = 0xEFCDAB89
C = c = 0x98BADCFE
D = d = 0x10325476
E = e = 0xC3D2E1F0

Визначаються чотири нелінійні операції і чотири константи:

$F_t(m,l,k) = (m \wedge l) \vee (\neg m \wedge k)$	K_t = 0x5A827999	0≤t≤19
$F_t(m,l,k) = m \oplus l \oplus k$	K_t = 0x6ED9EBA1	20≤t≤39
$F_t(m,l,k) = (m \wedge l) \vee (m \wedge k) \vee (l \wedge k)$	K_t = 0x8F1BBCDC	40≤t≤59
$F_t(m,l,k) = m \oplus l \oplus k$	K_t = 0xCA62C1D6	60≤t≤79

Головний цикл ітеративно обробляє кожен 512-бітний блок. Ітерація складається з чотирьох етапів по двадцять операцій у кожному. Блок повідомлення перетворюється з 16 32-бітових слів M_i у 80 32-бітових слів W_j за наступним правилом:

$$W_t = M_t \qquad \text{при } 0 \le t \le 15$$
$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \ll 1 \qquad \text{при } 16 \le t \le 79$$

тут << — це циклічний зсув вліво

```
для t від 0 до 79
        temp = (a << 5) + F_t(b,c,d) + e + W_t + K_t
        e = d
        d = c
        c = b << 30
        b = a
        a = temp
```

Після цього a, b, c, d, e додаються до A, B, C, D, E відповідно. Починається наступна ітерація. Підсумковим значенням буде об'єднання п'яти 32-бітних слів в одне 160-бітове хеш-значення.

Приклад значення хеш-функції:

sha-1(“Привіт”): 4b35ac662f1d1604689c02d22e6c1e3fc39b7dc8

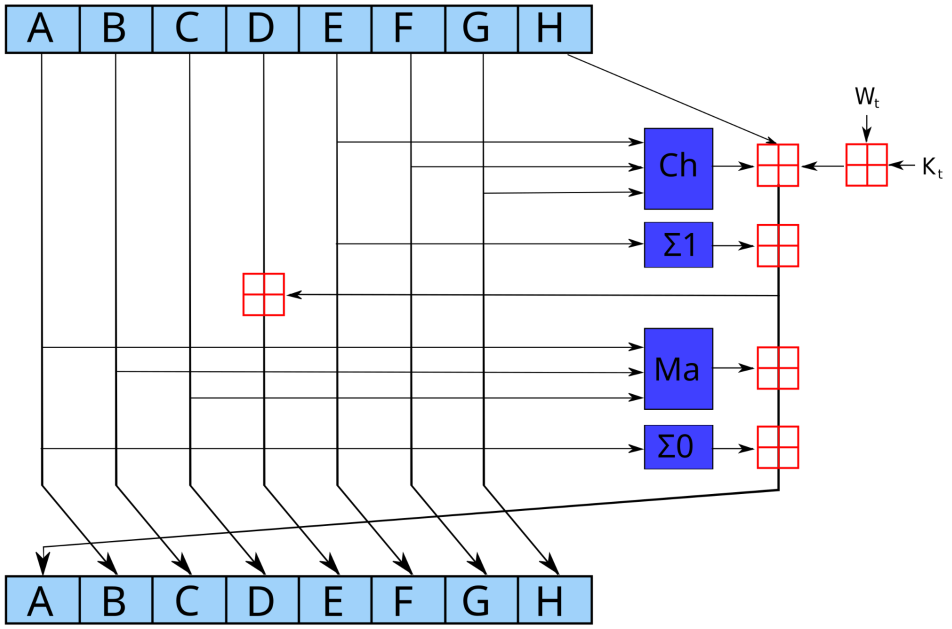
Алгоритми SHA-2

SHA-2 (Secure Hash Algorithm Version 2) — збірна назва односторонніх хеш-функцій SHA-224, SHA-256, SHA-384 і SHA-512. Хеш-функції призначені для створення «відбитків» або «дайджестів» повідомлень довільної бітової довжини. Застосовуються в різних додатках або компонентах, пов'язаних із захистом інформації. Хеш-функції сімейства *SHA-2* побудовані на основі структури Меркла-Демґарда.

Початкове повідомлення після доповнення розбивається на блоки, кожен блок — на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64-ма чи 80-ма ітераціями (раундами). На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням геш-функції.

Алгоритм використовує такі бітові операції:

8. ‖ — Конкатенація,
9. + — Додавання,
10. *And* — Побітове «І»,
11. *Or* — Побітове «АБО»,
12. *Xor* — Виключне «АБО»,
13. *Shr* (Shift Right) — Логічний зсув вправо,
14. *Rotr* (Rotate Right) — Циклічний зсув вправо.



Одна ітерація у функції стиснення сімейства SHA-2.
Сині компоненти виконують наступні операції:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

При побітовому обертанні використовуються інші константи для SHA-512. Наведені числа для SHA-256. Червоним кольором позначено додавання за модулем 2^{32} для SHA-256, або 2^{64} для SHA-512.

Приклади значення хеш-функції слова “Привіт”:

SHA-224:

6601e07f05c8b47cfcb6c4c9a219a1c0bd06ab8c148cd6f679d3b473

SHA-256: 626d11f11dd0073c4c78d9fd8c5b3034683219ac3d556f9038b646bc22b48c43

SHA-384:
d2c36d6465f44cd018de87a7210d40b19bcb17467f569a8a53616e87a520f7f9460ed0f601053319c
d3050f1fa9c7a2f

SHA-512:

6d1e9c051a3da97cf511343f4d46d5e506fe9f31e545027d004b21245a92d6e98627cbd0b1e3496b54f7da39a0c2e8752b4d7d59eace70f196de18c4e2dc247d

11.5 Варіанти завдань

№	Алгоритм
1	SHA-384
2	SHA-224
3	SHA-1
4	SHA-224
5	SHA-384
6	SHA-512
7	SHA-256
8	SHA-512
9	SHA-384
10	SHA-1
11	SHA-224
12	SHA-256
13	SHA-384
14	SHA-1
15	SHA-512
16	SHA-256
17	SHA-512
18	SHA-224
19	SHA-1
20	SHA-384
21	SHA-224
22	SHA-512
23	SHA-1
24	SHA-384

11.6 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,

- результати роботи програми,
- аналіз результатів
- висновки.

11.7 Контрольні запитання

- Що таке криптографічна хеш-функція? Які її основні властивості необхідні для забезпечення безпеки?
- Для яких цілей використовуються криптографічні хеш-функції в інформаційній безпеці?
- Які основні алгоритми входять до сімейства SHA (SHA-1, SHA-2, SHA-3)? У чому полягають їхні відмінності?
- Які атаки загрожують алгоритмам SHA (наприклад, колізії)? Чому алгоритм SHA-1 більше не рекомендується для використання?
- У яких сучасних протоколах та технологіях використовуються алгоритми сімейства SHA?

11.8 Література

- National Institute of Standards and Technology (NIST). *Secure Hash Standard (SHS)* // FIPS PUB 180-4. – Gaithersburg, MD: NIST, 2015.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
- Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.
- Bertoni G., Daemen J., Peeters M., Assche G. *Keccak* // Proceedings of the NIST SHA-3 Competition. – 2012.

ЛАБОРАТОРНА РОБОТА № 12 КРИПТОГРАФІЧНІ ХЕШ-ФУНКЦІЇ.

АЛГОРИТМ MD5. СІМЕЙСТВО АЛГОРИТМІВ RIPEMD.

Мета роботи: Вивчити існуючі алгоритми сімейства RIPEMD обчислення дайджестів повідомлень і написати програму, що реалізує заданий алгоритм хешування.

12.1 Теоретичні відомості

Хешування(Іноді хешування, англ. Hashing) - перетворення вхідного масиву даних довільної довжини в вихідну бітову рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями або функціями згортки, а їх результати називають хешем, хеш-кодом або дайджестом повідомлення (англ. Message digest).

Хешування застосовується для порівняння даних: якщо у двох масивів хеш-коди різні, масиви гарантовано розрізняються; якщо однакові - масиви, швидше за все, однакові. У загальному випадку однозначної відповідності між вихідними даними і хеш-кодом немає в силу того, що кількість значень хеш-функцій менше, ніж варіантів вхідного масиву. Також існує безліч масивів, що дають однакові хеш-коди - так звані колізії. Імовірність виникнення колізій відіграє важливу роль в оцінці якості хеш-функцій.

Існує безліч алгоритмів хешування з різними характеристиками (розрядність, обчислювальна складність, криптостойкость і т.п.). Вибір тієї чи іншої хеш-функції визначається специфікою розв'язуваної задачі. Найпростішими прикладами хеш-функцій можуть служити контрольні суми.

контрольні суми- це нескладні, вкрай швидкі і легко реалізовані апаратні алгоритми, використовувані для захисту від ненавмисних спотворень, в тому числі помилок апаратури. За швидкістю обчислення в десятки і сотні разів швидше, ніж криптографічні хеш-функції, і значно простіше в апаратної реалізації.

Платою за таку високу швидкість є відсутність криптостойкости - легка можливість підігнати повідомлення під заздалегідь відому суму. Розрядність контрольних сум (зазвичай 32 біта) нижче, ніж криптографічних хеш (типові значення - 128, 160 і 256 біт), що означає можливість виникнення ненавмисних колізій.

Найпростішим випадком такого алгоритму є розподіл повідомлення на 32- або 16-бітові слова і їх підсумовування, що застосовується, наприклад, в TCP / IP.

Як правило, до такого алгоритму пред'являються вимоги відстеження типових апаратних помилок, таких, як кілька поспіль помилкових біт до заданої довжини. До таких алгоритмам відноситься, наприклад, CRC32, застосовуваний в апаратурі Ethernet і в форматі упакованих файлів ZIP.

Серед безлічі існуючих хеш-функцій виділяють криптографически стійкі, що застосовуються в криптографії. Для того, щоб хеш-функція H вважалася

криптографически стійкою, вона повинна задовольняти трьом основним вимогам, на яких засновано більшість застосувань хеш-функцій в криптографії:

- ***незворотність***: Для заданого значення хеш-функції m повинно бути обчислювально нездійсненно знайти блок даних X, для якого H (X) = m.
- ***стійкість до колізій першого роду***: Для заданого повідомлення M має бути обчислювально нездійсненно підібрати інше повідомлення N, для якого H (N) = H (M).
- ***стійкість до колізій другого роду***: Має бути обчислювально нездійсненно підібрати пару повідомлень (M, M '), що мають однаковий хеш.

Слід зазначити, що не доведене існування незворотних хеш-функцій, для яких обчислення будь-якого прообразу заданого значення хеш-функції теоретично неможливо. Зазвичай знаходження зворотного значення є лише обчислювально складним завданням.

Для криптографічних хеш-функцій також важливо, щоб при щонайменшій зміні аргументу значення функції сильно змінювалося (це властивість називається лавинних ефектом). Зокрема, значення хеша не повинно давати витоку інформації навіть про окремі біти аргументу. Ця вимога є запорукою криптостойкости алгоритмів, хешірующіх пароль користувача для отримання ключа.

12.2 Завдання на лабораторну роботу

Реалізувати програму, що дозволяє обчислювати значення хеш-функції, заданої у варіанті:

- текст повідомлення повинен зчитуватися з файлу;
- отримане значення хеш-функції має представлятися в шістнадцятковому вигляді і зберігатися в файл;

12.3 Опис способу обробки результатів

Алгоритм MD5 (англ. Message Digest 5) - 128-бітний алгоритм хешування, розроблений Р. Ривестом з Массачусетського технологічного інституту (MIT) в 1991 році. Призначений для створення «відбитків» або «дайджестів» повідомлень довільної довжини. Є покращеною в плані безпеки версією алгоритму MD4. Використовується для перевірки автентичності опублікованих повідомлень шляхом порівняння дайджесту повідомлення.

Алгоритм MD5 був розроблений таким чином, щоб бути досить швидким для виконання на 32-розрядному процесорі. Алгоритм не вимагає великих таблиць підстановок і може бути закодований дуже компактно.

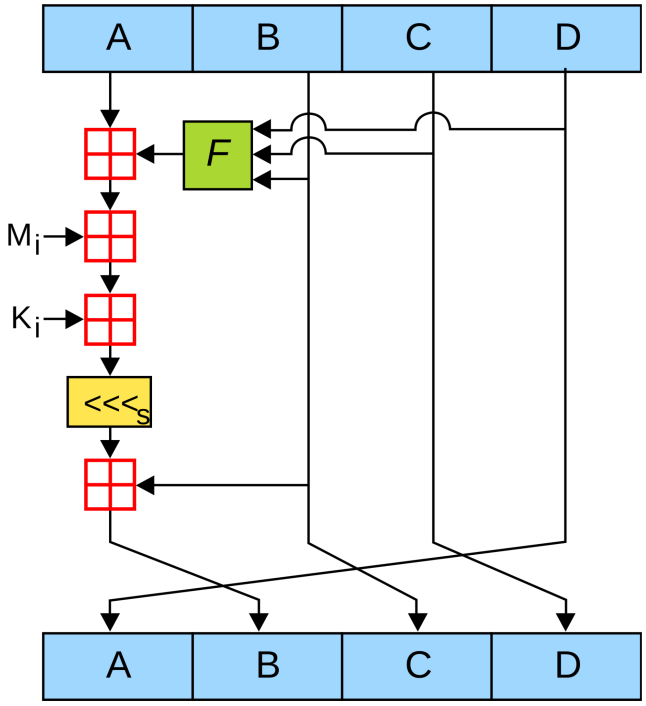


Рис. 1 Одна ітерація алгоритму MD5

Алгоритми сімейства RIPEMD (RIPE Message Digest) - це сімейство криптографічних хеш-функцій, розроблене в 1992 (оригінальний RIPEMD) і 1996 (інші варіанти) роках. У сімействі п'ять функцій: RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256 і RIPEMD-320, з яких RIPEMD-160 є найбільш поширеною. Оригінальний RIPEMD, як і RIPEMD-128, не вважається безпечним, оскільки 128-бітний результат занадто малий, а також (для оригінального RIPEMD) через недоліки конструкції. 256- і 320-розрядні версії RIPEMD забезпечують той же рівень безпеки, що і RIPEMD-128 і RIPEMD-160 відповідно; вони призначені для додатків, де рівень безпеки достатній, але необхідний довший результат хешування. Хоча функції RIPEMD менш популярні, ніж SHA-1 і SHA-2, вони використовуються, зокрема, в Bitcoin та інших криптовалютах, заснованих на Bitcoin.

RIPEMD-160

RIPEMD-160 була розроблена у відкритій академічній спільноті, на відміну від SHA-1 і SHA-2, які були створені NSA. З іншого боку, RIPEMD-160 на практиці використовується не так часто, як SHA-1. Використання RIPEMD-160 не обмежене жодними патентами.

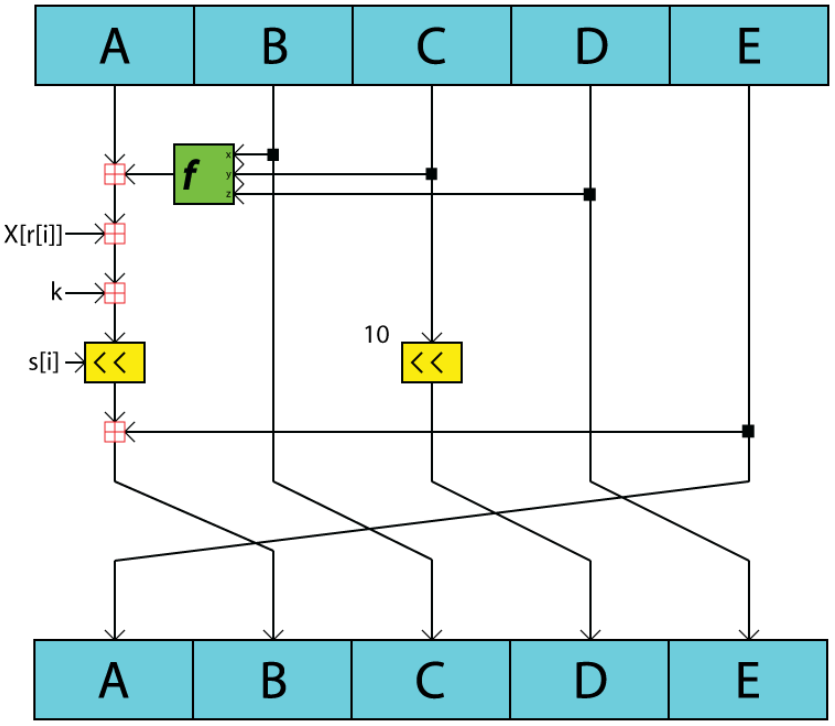


Рис. 2 Підблок з функції стиснення хеш-алгоритму RIPEMD-160

12.4 Опис ходу експерименту

Алгоритм MD5

Початковий етап підготовки

Вхідні дані вирівнюються так, щоб їхній розмір можна було порівняти з 448 по модулю з 512. Спочатку дописують одиничний біт (навіть якщо довжина порівняна з 448), далі необхідна кількість нульових бітів.

Дописування 64-бітного представлення довжини даних по вирівнюванню. Якщо довжина перевищує 2^{64}-1, то дописують молодші біти

Допоміжні таблиці та функції

Ініціалізуються 4 змінних розміром по 32 біта:

8.8 A = 01 23 45 67;

8.9 B = 89 AB CD EF;

8.10 C = FE DC BA 98;

8.11 D = 76 54 32 10.

Вирівнювані дані розбиваються на блоки по 32 біта, і кожен проходить 4 раунди з 16 операторів. Всі оператори однотипні і мають вигляд: [abcd k s i], визначений як

$$a = b + ((a + Fun(b, c, d) + X[k] + T < i >) <<< s),$$

де X - блок даних, а T[1..64] - 64-елементна таблиця, побудована наступним чином:

$$T[i] = int(4294967296 * |sin(i)|),$$

s - циклічний зсув вліво на s біт отриманого 32-бітного аргументу.

- В першому раунді Fun F(X, Y, Z) = XY v (not X)Z
- В другому раунді Fun G(X, Y, Z) = XZ v (not Z)Y.

- В третьому раунді Fun H(X, Y, Z) = X xor Y xor Z.
- В четвертому раунді Fun I(X, Y, Z) = Y xor (X v (not Z)).

Циклічна процедура обчислення

Саме обчислення проходить наступним чином:
Зберігаються значення A, B, C і D, що залишились після операцій з попередніми блоками(або їх початкові значення якщо блок перший).
AA = A
BB = B
CC = C
DD = D

Раунд 1

/*[abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]
[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]
[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

Раунд 2

/*[abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]
[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]
[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]
[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

Раунд 3

/*[abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]
[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]
[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]
[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

Раунд 4

/*[abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]
[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]
[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]
[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Проміжний результат

Виконати наступні операції

A = AA + A
B = BB + B
C = CC + C
D = DD + D

Після цього перевірити, чи є ще блоки, якщо є, то повторюють циклічну процедуру обчислення для наступного 32-х бітового блоку.

Результат

Після обчислення для всіх блоків даних, отримуємо кінцевий хеш у регістрах A B C D. Якщо вивести слова у зворотному порядку DCBA, то отримаємо MD5 хеш.

Псевдокод алгоритму MD5:

```
// : All variables are unsigned 32 bit and wrap modulo 2^32 when calculating
var int s[64], K[64]
var int i

// s specifies the per-round shift amounts
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22 }
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20 }
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23 }
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21 }

// Use binary integer part of the sines of integers (Radians) as constants:
for i from 0 to 63 do
    K[i] := floor(2^32 × abs(sin(i + 1)))
end for
// (Or just use the following precomputed table):
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

// Initialize variables:
var int a0 := 0x67452301   // A
var int b0 := 0xefcdab89   // B
var int c0 := 0x98badcfe   // C
var int d0 := 0x10325476   // D

// Pre-processing: adding a single 1 bit
append "1" bit to message<
 // Notice: the input bytes are considered as bit strings,
 //  where the first bit is the most significant bit of the byte.[53]

// Pre-processing: padding with zeros
append "0" bit until message length in bits ≡ 448 (mod 512)

// Notice: the two padding steps above are implemented in a simpler way
 //  in implementations that only work with complete bytes: append 0x80
 //  and pad with 0x00 bytes so that the message length in bytes ≡ 56 (mod 64).

append original length in bits mod 2^64 to message

// Process the message in successive 512-bit chunks:
for each 512-bit chunk of padded message do
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15
    // Initialize hash value for this chunk:
```

```
    var int A := a0
    var int B := b0
    var int C := c0
    var int D := d0
    // Main loop:
    for i from 0 to 63 do
        var int F, g
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31 then
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47 then
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63 then
            F := C xor (B or (not D))
            g := (7×i) mod 16
        // Be wary of the below definitions of a,b,c,d
        F := F + A + K[i] + M[g] // M[g] must be a 32-bit block
        A := D
        D := C
        C := B
        B := B + leftrotate(F, s[i])
    end for
    // Add this chunk's hash to result so far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 // (Output is in little-endian)
```

12.5 Варіанти завдань

№	Алгоритм
1	MD5
2	RIPEMD-160
3	MD5
4	RIPEMD-160
5	RIPEMD-256
6	MD5
7	RIPEMD-256
8	RIPEMD-512
9	RIPEMD-160
10	MD5
11	RIPEMD-256
12	MD5
13	RIPEMD-256
14	RIPEMD-256

15	MD5
16	RIPEMD-256
17	RIPEMD-512
18	MD5
19	RIPEMD-512
20	MD5
21	RIPEMD-256
22	RIPEMD-160
23	RIPEMD-512
24	MD5

12.6 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

12.7 Контрольні запитання

- Що таке криптографічна хеш-функція, і які основні її властивості?
- Чому MD5 більше не вважається безпечним для криптографічних застосувань?
- У яких випадках MD5 може бути доцільним, незважаючи на його вразливості?
- Що таке алгоритми сімейства RIPEMD, і які їхні основні особливості?
- У чому полягають ключові відмінності між RIPEMD-160 та MD5 з точки зору безпеки?

12.8 Література

- Rivest R. *The MD5 Message-Digest Algorithm* // RFC 1321. – 1992.
- Dobbertin H. *The Status of MD5 After a Recent Attack* // CryptoBytes. – 1996. – Vol. 2, No. 2. – P. 1–6.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
- Dobbertin H., Bosselaers A., Preneel B. *RIPEMD-160: A Strengthened Version of RIPEMD* // Fast Software Encryption. – Springer, 1996. – P. 71–82.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.

ЛАБОРАТОРНА РОБОТА №13 ПРАКТИЧНЕ ВИКОРИСТАННЯ ПРОГРАМИ PGP ДЛЯ ШИФРУВАННЯ ДАНИХ.

Мета роботи: Знайомство з основними принципами передачі і прийому зашифрованих повідомлень електронної пошти. Набуття практичних навичок роботи з програмою PGP.

13.1 Теоретичні відомості

Програма PGP (Pretty Good Privacy - досить хороша секретність) розроблена Філіпом Циммерманом і розповсюджується вільно програмою. Незважаючи на це, на відміну від деяких комерційних програм, програма PGP забезпечує дуже високий ступінь захисту даних за рахунок використання алгоритмів шифрування з відкритим ключем і застосування ключів з довжиною 1024 біт і більше. Після установки програми в операційній системі Windows, в системному треї з'являється іконка  , А в списку програм - вкладка PGP (рис. 4).

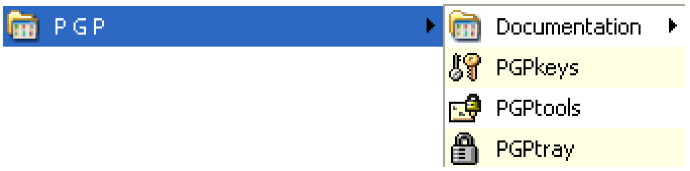


Рис. 4. Вкладка програми PGP в меню «Пуск / Всі програми».

Управляти програмою можна і через меню, і через клацання по іконці. Робота в програмі ділиться на три етапи:

1. Створення пари ключів і обмін відкритими ключами з кореспондентами;
2. Шифрування повідомлень і документів;
3. Дешифрування повідомлень і документів

13.2 Завдання на лабораторну роботу

Робота виконується групами по дві-три людини. Кожен член групи повинен:

1. Створити пару ключів для використання шифрування з відкритим ключем;
2. Виконати експорт свого відкритого ключа та передачу його всім членам групи;
3. Виконати шифрування довільного повідомлення для кожного члена групи і відправку зашифрованого повідомлення адресатам;
4. Кожен член групи виконує дешифрування повідомлень, отриманих від своїх кореспондентів.
5. Кожен член групи демонструє викладачеві:
   - Створену ним пару ключів шифрування;
   - Отримані від членів групи відкриті ключі;
   - Вихідні повідомлення, що передаються членам групи після шифрування;

- Зашифровані повідомлення, що передаються членам групи;
- Отримані від членів групи зашифровані повідомлення;
- Розшифровані повідомлення.

13.3 Опис способу обробки результатів

Створення пари ключів і обмін відкритими ключами

Для початку роботи необхідно створити пару ключів: відкритий (public) і закритий (privat). Вибираючи пункт PGPKeys, ми відкриваємо вікно програми для роботи з ключами шифрування (рис. 5).

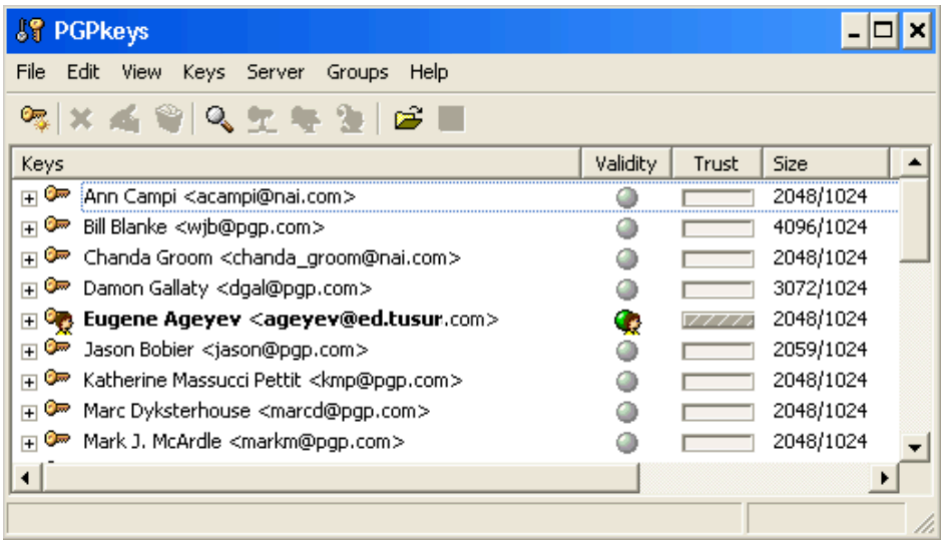


Рис. 5. Вікно програми PGP для створення і управління параметрами ключів шифрування.

Вибір в цьому вікні пункту меню Keys / New Key ... (рис. 6) запускає майстер, що дозволяє в кілька кроків створити пару ключів на основі індивідуальної інформації (імені, адреси електронної пошти) і символьного пароля.

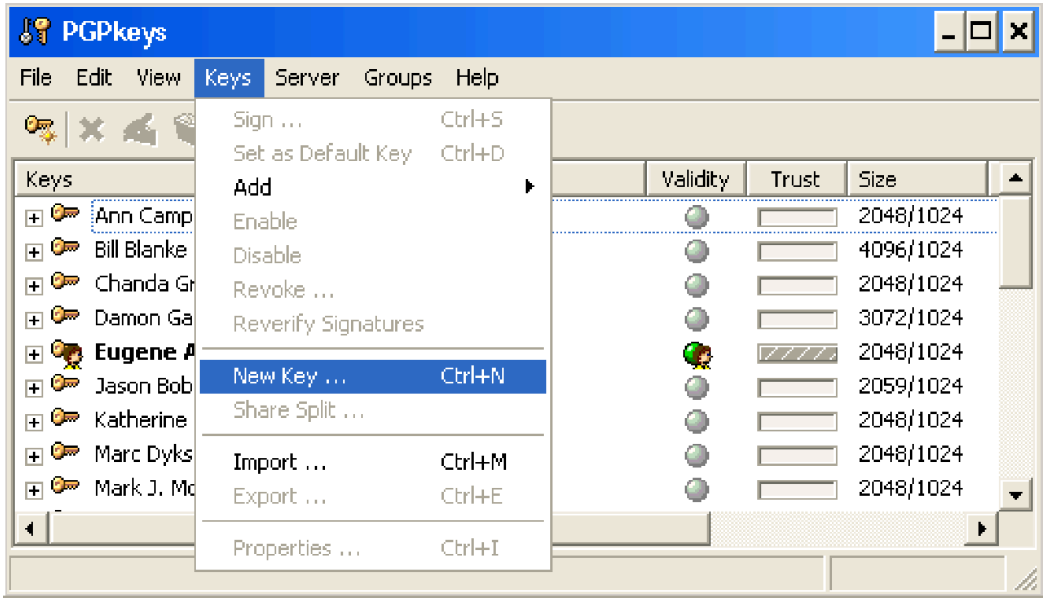


Рис. 6. Запуск майстра створення нової пари ключів.

Обмін ключами можна здійснювати за допомогою спеціального сервера, надійність якого не викликає сумнівів (пункт Server в меню вікна PGPKeys) або шляхом прямої

передачі ключа, яку можна здійснити, виконавши експорт ключа у вигляді текстового файлу, і передавши його своїм кореспондентам будь-яким шляхом: електронною поштою , копіюванням і т.п. Експорт відкритого (public) ключа можна виконати через меню Keys / Export ... або контекстне меню (контекстне меню Export ... викликається правою клавішею мишки, встановленої на рядок вікна програми з новоствореною парою ключів). Отримавши будь-яким із способів відкритий ключ, кореспондент повинен імпортувати його в програму (меню Keys / Import), після чого він зможе його використовувати для шифрування повідомлень для того адресата, який передав йому цей ключ.

13.4 Опис ходу експерименту

Шифрування повідомлень і документів

Процес шифрування нескладний і може бути виконаний двома способами.

Спосіб 1. Вибором команди PGPTools при натисканні мишкою по іконці PGP в системному треї або в вікні меню програми (рис. 4). При цьому відкривається вікно PGPTools програми PGP (рис. 7). Клацання мишкою на другий зліва піктограмі в цьому вікні відкриває вікно вибору файлу (або групи файлів) для шифрування.

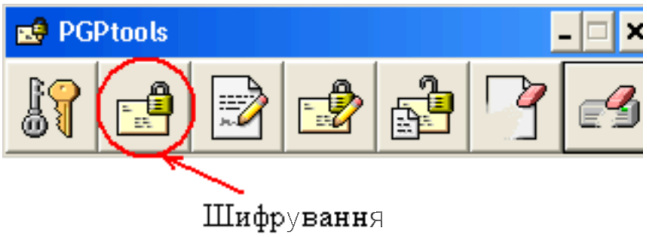


Рис. 7. Створення зашифрованого повідомлення (файлу) за допомогою панелі PGPTools.

Після завершення вибору файлу відкривається вікно вибору кореспондента (рис. 8). Цей вибір виконується подвійним клацанням по рядку з обраним користувачем або перетягуванням цього рядка у вікно Recipients. Відповідний відкритий ключ для вказаного користувача буде використаний автоматично в процесі шифрування. Зашифрований файл має розширення «pgp». У цьому ж вікні можна задати традиційний (Conventional), більш про стій, спосіб шифрування з використанням парольної фрази, встановивши прапорці в рядках Conventional Encryption або Self Decrypting Archive. Алгоритм шифрування з відкритим ключем в цьому випадку не використовується і для розшифровки повідомлення його одержувач повинен буде дізнатися парольний фразу, яка використовувалася при шифруванні. Прапорець Wipe Original дозволяє виконати шифрування з одночасним видаленням оригіналу файлу,

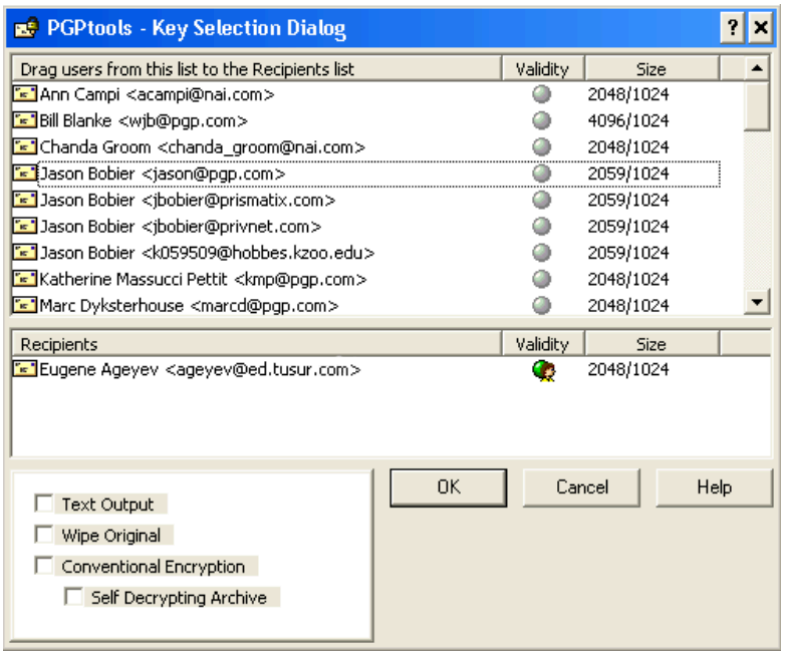


Рис. 8. Вікно вибору кореспондента і відкритого ключа для шифрування.

Другий спосіб шифрування файлу - за допомогою контекстного меню, що з'являється при виборі файлу у вікні «Провідника» і натисканні правої клавіші мишки (рис. 9).

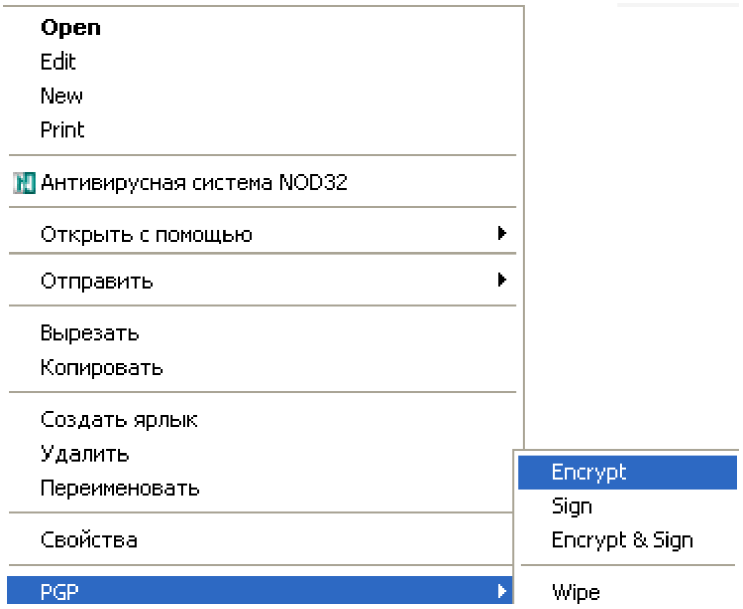


Рис. 9. Шифрування файлу (групи файлів) за допомогою контекстного меню.

Оскільки вибір файлу в цьому випадку вже виконано, відразу відбувається перехід до вікна вибору ключа шифрування (рис. 8).

Дешифрування повідомлення або файлу

Для розшифрування файлу, його одержувач просто виконує подвійне клацання лівою кнопкою мишки на файлі. Ця дія викликає вікно введення пароля, використаного

при створенні пари з відкритого і закритого ключів (рис. 10).

Рис. 10. Вікно введення пароля при розшифровці файлу.

Введення коректного пароля запускає процес дешифрування, в результаті якого виходить розшифрований вихідний файл.

13.5 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

13.6 Контрольні запитання

- Що таке відкритий ключ, поясніть механізм роботи шифрування з відкритим ключем.
- Що таке цифровий підпис.
- Ким була створена програма PGP.
- Чому програма PGP забезпечує дуже високу ступінь захисту зашифрованих файлів.
- Що таке електронний сертифікат, чи може електронний сертифікат створюватися за допомогою програми PGP.

13.7 Література

- Zimmermann P. *PGP User's Guide*. – MIT Press, 1991.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
- Callas J., Schneier B., Ferguson N., et al. *OpenPGP Message Format* // RFC 4880. – 2007.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
- Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.

ЛАБОРАТОРНА РОБОТА №14 КОМБІНОВАНИЙ МЕТОД ШИФРУВАННЯ (ГІБРИДНІ КРИПТОСИСТЕМИ)

Мета роботи: Вивчення принципів побудови та функціонування гібридних криптосистем, які поєднують симетричні та асиметричні методи шифрування для забезпечення високого рівня безпеки даних.

14.1 Теоретичні відомості

Загальні поняття про гібридні криптосистеми

Гібридні криптосистеми поєднують два криптографічні підходи: симетричне шифрування та асиметричне шифрування. Основна ідея полягає в комбінуванні переваг кожного з них:

- Симетричне шифрування забезпечує швидкість обробки великих обсягів даних.
- Асиметричне шифрування забезпечує безпечну передачу ключів для симетричного алгоритму.
- Гібридний підхід дозволяє використовувати симетричні алгоритми для ефективного шифрування великих файлів, а асиметричні — для захищеного обміну ключами.

Симетричне шифрування

Симетричне шифрування використовує один секретний ключ для шифрування та дешифрування даних. Приклади алгоритмів: AES, DES, ChaCha20.

Переваги:

- Висока швидкість шифрування/дешифрування.
- Простота реалізації.

Недоліки:

- Потреба у безпечному способі передачі ключа.

Принцип роботи AES (Advanced Encryption Standard):

AES є блоковим шифром із розміром блоку 128 біт та довжинами ключів 128, 192 або 256 біт. Він складається з кількох раундів, у кожному з яких виконуються операції: SubBytes, ShiftRows, MixColumns та AddRoundKey. AES широко використовується у сучасних гібридних криптосистемах.

Асиметричне шифрування

Асиметричне шифрування використовує пару ключів: відкритий ключ (для шифрування) і закритий ключ (для дешифрування). Приклади алгоритмів: RSA, ECC, ElGamal.

Переваги:

- Не потрібно передавати секретний ключ через небезпечний канал.

Недоліки:

- Низька швидкість обробки великих обсягів даних.

Принцип роботи RSA (Rivest–Shamir–Adleman):

RSA базується на складності факторизації великих чисел. Відкритий ключ використовується для шифрування симетричного ключа, а закритий — для його дешифрування.

Основи гібридної криптосистеми

У гібридних криптосистемах шифрування даних виконується у два етапи:

- Генерація та шифрування симетричного ключа:

    - Генерується сеансовий ключ для симетричного шифрування.
    - Сеансовий ключ шифрується відкритим ключем одержувача (RSA).
- Шифрування даних: Дані шифруються симетричним алгоритмом (AES) з використанням сеансового ключа.
- Розшифрування: Одержувач дешифрує сеансовий ключ за допомогою свого закритого ключа (RSA). Потім використовує цей сеансовий ключ для дешифрування даних.

Переваги гібридних криптосистем

- Висока швидкість обробки великих даних завдяки симетричному шифруванню.
- Безпечна передача ключів завдяки асиметричному шифруванню.
- Підвищена криптографічна стійкість через комбінування двох підходів.

Реальні приклади використання гібридних криптосистем

- TLS/SSL:

Використовується для захищеного передавання даних у мережі Інтернет.
Симетричне шифрування забезпечує швидкість, а асиметричне — безпеку передачі ключів.

- PGP (Pretty Good Privacy):

Забезпечує захищене шифрування електронної пошти.

- SSH (Secure Shell):

Використовується для захищеного доступу до віддалених серверів.

- Протоколи IPsec:

Забезпечують захист мережевого трафіку.

14.2 Завдання на лабораторну роботу

- Реалізувати алгоритм комбінованого шифрування з використанням симетричного (AES) та асиметричного (RSA) методів шифрування.
- Здійснити шифрування заданого текстового повідомлення.
- Реалізувати дешифрування повідомлення за умови, що є доступ до закритого ключа.
- Оцінити час шифрування та дешифрування для різних обсягів даних.

14.3 Опис способу обробки результатів

Обробка результатів лабораторної роботи спрямована на:

- Визначення ефективності комбінованого методу шифрування.
- Оцінку швидкості роботи симетричного та асиметричного алгоритмів окремо й у гібридній системі.
- Перевірку коректності шифрування та дешифрування даних.

Етапи обробки результатів:

1. Аналіз коректності шифрування/дешифрування
    1. Порівняння вхідних і вихідних даних:
        - Дешифроване повідомлення має збігатися з оригінальним текстом.
        - Якщо є відхилення, це може свідчити про помилки в реалізації алгоритмів.
    2. Перевірка сеансового ключа:
        - Сеансовий ключ після дешифрування закритим ключем повинен збігатися з початково згенерованим.
2. Вимірювання продуктивності
    1. Оцінка часу виконання шифрування та дешифрування:
        - Виміряти час, необхідний для:
            - Шифрування тексту симетричним алгоритмом (AES).
            - Шифрування сеансового ключа асиметричним алгоритмом (RSA).
            - Дешифрування сеансового ключа RSA.
            - Дешифрування тексту AES.

2. Порівняння часу роботи:
    - Порівняти швидкість роботи симетричного алгоритму (AES) з асиметричним (RSA).
    - Визначити, наскільки ефективнішим є гібридний метод у порівнянні з використанням лише RSA для шифрування всіх даних.

3. Аналіз споживання ресурсів
    1. Використання пам'яті:
        - Оцінити обсяг пам'яті, необхідний для роботи алгоритмів.
        - Перевірити, чи відрізняється споживання ресурсів у гібридній системі від окремих методів.
    2. Складність обчислень:
        - Визначити, як обсяг даних впливає на швидкість роботи алгоритмів.
4. Ефективність для великих обсягів даних
    1. Тестування на різних обсягах даних:
        - Виконати шифрування текстових файлів розміром 100 КБ, 1 МБ і 10 МБ.
        - Зробити висновки про придатність гібридної системи для великих обсягів даних.
    2. Побудова графіків:
        - Побудувати графіки залежності часу шифрування/дешифрування від розміру даних для симетричного, асиметричного та гібридного методів.

14.4 Опис ходу експерименту

Покроковий опис алгоритму:

1. Генерація симетричного ключа (AES):
    - Згенеруйте випадковий 256-бітний ключ AES.
    - Використовуйте його для шифрування текстових даних.
2. Генерація пари ключів для RSA:
    - Згенеруйте пару ключів (відкритий і закритий ключі) для RSA.
    - Використовуйте відкритий ключ для шифрування симетричного ключа.
3. Шифрування даних:
    - Шифруйте текстові дані за допомогою AES.
    - Зашифруйте симетричний ключ за допомогою відкритого ключа RSA.
4. Дешифрування даних:
    - Розшифруйте симетричний ключ за допомогою закритого ключа RSA.
    - Використовуйте отриманий симетричний ключ для дешифрування текстових даних.

Псевдокод для комбінованого шифрування з використанням AES і RSA:

```
PROCEDURE main():
  // 1. Дані для шифрування
  plaintext = "Data to be encrypted"

  // 2. Генерація AES-ключа
  aesKey = generateAESKey()

  // 3. Генерація RSA-пари ключів
  privateKey, publicKey = generateRSAKeys()

  // 4. Шифрування даних за допомогою AES
  iv, ciphertext = aesEncrypt(plaintext, aesKey)

  // 5. Шифрування AES-ключа відкритим ключем RSA
  encryptedAESKey = rsaEncryptKey(aesKey, publicKey)

  // 6. Дешифрування AES-ключа закритим ключем RSA
  decryptedAESKey = rsaDecryptKey(encryptedAESKey, privateKey)

  // 7. Дешифрування даних за допомогою розшифрованого AES-ключа
  decryptedPlaintext = aesDecrypt(iv, ciphertext, decryptedAESKey)
```

```
// 8. Перевірка коректності шифрування і дешифрування
IF decryptedPlaintext == plaintext:
  PRINT("Encryption and decryption successful")
ELSE:
  PRINT("Decryption failed")
```

Пояснення роботи алгоритму:

1. Генерація AES-ключа:
    - Симетричний ключ використовується для шифрування великих даних.
2. Генерація RSA-ключів:
    - RSA-пара ключів використовується для безпечного обміну AES-ключем.
3. Шифрування даних (AES):
    - Дані шифруються симетричним алгоритмом AES з випадковим IV.
4. Шифрування AES-ключа (RSA):
    - AES-ключ шифрується відкритим ключем сервера для передачі.
5. Дешифрування AES-ключа (RSA):
    - Клієнт використовує закритий ключ RSA для отримання AES-ключа.
6. Дешифрування даних (AES):
    - Розшифровані дані відновлюються симетричним алгоритмом AES.

14.5 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

14.6 Контрольні запитання

- У чому полягає перевага гібридних криптосистем над окремими симетричними чи асиметричними методами?
- Як забезпечується безпека передачі симетричного ключа у гібридних криптосистемах?
- Чому алгоритми симетричного шифрування швидші за асиметричні?
- Які особливості використання алгоритму AES для шифрування великих обсягів даних?
- Як можна підвищити стійкість гібридної криптосистеми до атак?

14.7 Література

- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020.
- Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996.
- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020.
- Katz J., Lindell Y. *Introduction to Modern Cryptography*. 3rd ed. – CRC Press, 2020.
- RFC 5246. *The Transport Layer Security (TLS) Protocol*. – 2008.

ЛАБОРАТОРНА РОБОТА №15 КРИПТОГРАФІЧНІ ПЕРЕТВОРЕННЯ НА ЕЛІПТИЧНИХ КРИВИХ

Мета роботи: Ознайомитися з алгоритмами криптографічних перетворень на еліптичних кривих, їх застосуванням у криптографічних системах і оцінити ефективність таких перетворень.

15.1 Теоретичні відомості

Криптографія на еліптичних кривих (Elliptic Curve Cryptography, ECC) — це сучасний метод захисту даних, що базується на математичних властивостях еліптичних кривих над скінченними полями. ECC використовується для забезпечення конфіденційності, автентифікації, а також для створення цифрових підписів і обміну

ключами. Її основною перевагою є висока криптостійкість при значно меншому розмірі ключа у порівнянні з іншими методами, такими як RSA.

Основи еліптичних кривих

Еліптична крива визначається рівнянням виду:

$y^2 = x^3 + ax + bmodp$

де:

- a і b — коефіцієнти, що визначають форму кривої.
- p — просте число, що задає модуль операцій.

Еліптичні криві повинні відповідати умові:

$4a^3 + 27b^2 \neq 0modp$,

щоб уникнути вироджених випадків.

Точки на еліптичній кривій — це набір координат (x,y)(x, y), які задовольняють це рівняння. Крім того, існує спеціальна точка, яка називається нескінченною, і використовується як нейтральний елемент у групових операціях.

Операції на еліптичних кривих

1. Додавання точок

   Для двох точок $P = (x_1, y_1)$ і $Q = (x_2, y_2)$ на еліптичній кривій виконується:

   - Якщо P≠Q:

$\lambda = \frac{y_2 - y_1}{x_2 - x_1} modp$, $x_3 = \lambda^2 - x_1 - x_2 modp$, $y_3 = \lambda(x_1 - x_3) - y_1 modp$.

   - Якщо P=Q (подвоєння точки):

$\lambda = \frac{3x_1^2 + a}{2y_1} modp$, $x_3 = \lambda^2 - 2x_1 modp$, $y_3 = \lambda(x_1 - x_3) - y_1 modp$.

2. Множення точки на скаляр

   Множення точки P на ціле число k (kP) досягається через повторне додавання точки. Для ефективності використовують метод подвоєння та додавання.

Криптографічні проблеми на еліптичних кривих

1. Проблема дискретного логарифму на еліптичних кривих (ECDLP):

   Задача: для заданої точки P і результату Q=kP знайти k.

   Вирішення цієї задачі вважається обчислювально складним, що забезпечує стійкість криптографічних алгоритмів на ECC.
2. ECDLP у порівнянні з класичним дискретним логарифмом:

   Еліптичні криві дозволяють досягти такого ж рівня безпеки, як і RSA, але при значно меншому розмірі ключів. Наприклад:

   - ECC з ключем 256 біт еквівалентний RSA з ключем 3072 біти.

Основні криптографічні алгоритми на еліптичних кривих

1. ECDH (Elliptic Curve Diffie-Hellman):

   Протокол обміну ключами, який дозволяє двом сторонам узгодити спільний секретний ключ через незахищений канал.

   - Етапи роботи:
     - Сторона A генерує пару ключів (d_A, Q_A), де d_A — приватний ключ, $Q_A = d_A P$ — публічний ключ.
     - Сторона B генерує аналогічну пару (d_B, Q_B).
     - Секретний ключ обчислюється як $K = d_A Q_B = d_B Q_A$.

2. ECDSA (Elliptic Curve Digital Signature Algorithm):
   Алгоритм цифрового підпису, який забезпечує автентифікацію повідомлень.
   - Етапи роботи:
     - Підпис: Генерація випадкового числа k, обчислення точки $R = kP$ та створення підпису (r, s).
     - Перевірка: Використання публічного ключа підписувача для перевірки підпису.
3. ECIES (Elliptic Curve Integrated Encryption Scheme):
   Гібридний метод, що використовує ECC для захищеної передачі ключів, а симетричний алгоритм (наприклад, AES) — для шифрування даних.

Переваги криптографії на еліптичних кривих

1. Менший розмір ключів:
   - Ключі ECC значно коротші за RSA, що знижує вимоги до пам'яті та мережевого трафіку.
2. Швидкість:
   - Операції ECC швидші, ніж RSA, особливо при перевірці підписів.
3. Стійкість:
   - ECC забезпечує високу криптостійкість, що робить її придатною для мобільних пристроїв та вбудованих систем.

Застосування криптографії на еліптичних кривих

1. TLS/SSL:
   - Протоколи захищеного веб-трафіку широко використовують ECC для обміну ключами.
2. Блокчейн:
   - Більшість криптовалют, включаючи Bitcoin, використовують ECC для цифрових підписів (ECDSA).
3. Мобільні пристрої та IoT:
   - Завдяки ефективності ECC вона є ідеальною для енергообмежених пристроїв.
4. Шифрування електронної пошти:
   - PGP і S/MIME підтримують ECC для захисту повідомлень.
5. Військові та урядові системи:
   - ECC широко використовується в стандартах NIST і NSA для захисту секретної інформації.

15.2 Завдання на лабораторну роботу

- Реалізувати операції додавання і множення точок на еліптичній кривій.
- Реалізувати алгоритм ECDH для обміну ключами.
- Реалізувати цифровий підпис та перевірку за допомогою алгоритму ECDSA.
- Здійснити оцінку ефективності операцій на еліптичних кривих у порівнянні з алгоритмами RSA.

15.3 Опис способу обробки результатів

Обробка результатів лабораторної роботи спрямована на:

1. Перевірку правильності реалізації криптографічних операцій на еліптичних кривих.
2. Оцінку ефективності алгоритмів з використанням різних параметрів еліптичних кривих.

3. Порівняння обчислювальної складності криптографічних операцій із традиційними алгоритмами (наприклад, RSA).
4. Аналіз безпеки та стійкості обраних криптографічних методів.

Основні етапи обробки результатів

1. Перевірка правильності роботи алгоритмів
    1. Операції на еліптичних кривих:
        - Обчислення суми точок і множення точки на скаляр:
            - Результат обчислення перевіряється на відповідність правилам роботи з еліптичними кривими.
            - Наприклад, P+O=P (нейтральний елемент).
    2. ECDH (Elliptic Curve Diffie-Hellman):
        - Перевіряється, чи однакові спільні секретні ключі обчислюються обома сторонами: $KA=dA \cdot QB, KB=dB \cdot QA, KA=KB$.
    3. ECDSA (Elliptic Curve Digital Signature Algorithm):
        - Перевірка автентичності підпису:
            - Цифровий підпис має бути визнаний коректним, якщо повідомлення, закритий ключ підписувача та підпис збігаються.
2. Вимірювання продуктивності
    1. Оцінка часу виконання операцій:
        - Виміряйте час обчислення суми точок та множення точки на скаляр.
        - Для ECDH:
            - Час генерації публічного ключа.
            - Час обчислення спільного секретного ключа.
        - Для ECDSA:
            - Час генерації цифрового підпису.
            - Час перевірки цифрового підпису.
    2. Аналіз залежності продуктивності від параметрів кривої:
        - Змінюйте розмір ключа (наприклад, 192, 256, 384 біт) та оцінюйте вплив на час виконання операцій.
    3. Порівняння з RSA:
        - Виконайте аналогічні операції для RSA (генерація ключів, шифрування, підпис) з еквівалентним рівнем безпеки (наприклад, ECC-256 ≈ RSA-3072).

15.4 Опис ходу експерименту

Встановлення програмного забезпечення

1. Середовище розробки:
    - Встановіть Python, Java, або інше програмне середовище для роботи з криптографією.
    - Для Python рекомендується бібліотека ecdsa для роботи з еліптичними кривими.
    - Для Java — бібліотека BouncyCastle.
    - Для C++ — Crypto++.
2. Налаштування середовища:
    - Переконайтеся, що бібліотеки для криптографії встановлені коректно.

- Створіть новий проект для збереження коду, даних і результатів експерименту.

Підготовка параметрів еліптичної кривої

1. Виберіть стандартну еліптичну криву, наприклад:
   - P-192, P-256, P-384 (NIST стандарти).
   - Curve25519 (для сучасних застосувань).
2. Налаштуйте параметри кривої:
   - Координати (a, b).
   - Просте число p, яке задає скінченне поле.
   - Генераторну точку G.

Підготовка даних для шифрування

1. Створіть текстові файли різного розміру (наприклад, 100 байт, 1 КБ, 10 КБ).
2. Використовуйте випадковий текст або заздалегідь підготовлені повідомлення.

Реалізація операцій на еліптичних кривих

1. Операція додавання точок:
   - Реалізуйте алгоритм для обчислення суми двох точок $P + Q$ на еліптичній кривій.
   - Перевірте результати:
     - $P + O = P$.
     - Якщо $P = Q$, обчисліть подвоєння точки.
2. Операція множення точки на скаляр:
   - Реалізуйте алгоритм для обчислення kP через метод подвоєння та додавання.
   - Перевірте правильність реалізації:
     - $2P = P + P$.
     - $nP = O$ (для порядку n).

Реалізація алгоритму ECDH (Elliptic Curve Diffie-Hellman)

1. Генерація ключів:
   - Для кожної сторони (наприклад, Аліса і Боб) створіть:
     - Закритий ключ d_A, d_B (випадкове число).
     - Відкритий ключ $Q_A = d_A G$, $Q_B = d_B G$.
2. Обчислення спільного ключа:
   - Аліса обчислює $K = d_A Q_B$.
   - Боб обчислює $K = d_B Q_A$.
   - Перевірте, чи збігаються отримані ключі.

Реалізація алгоритму ECDSA (Elliptic Curve Digital Signature Algorithm)

1. Підпис повідомлення:
   - Згенеруйте випадкове число k.
   - Обчисліть точку $R = kG$.
   - Визначте підпис (r, s), де: $r = x_R mod n$, $s = k^{-1}(H(m) + d \cdot r) mod n$.
2. Перевірка підпису:
   - Перевірте підпис, використовуючи відкритий ключ підписувача.

Демонстраційний код із реалізацією алгоритму ECDSA (цифровий підпис та його

перевірка) на Python:

```
from random import randint

# Параметри еліптичної кривої (y^2 = x^3 + ax + b mod p)
a, b = 2, 3
p = 97  # Модуль
G = (3, 6)  # Генераторна точка
n = 5  # Порядок точки G (спрощено для демонстрації)

# Додавання точок на еліптичній кривій
def point_add(P, Q):
    if P == (None, None):
        return Q
    if Q == (None, None):
        return P
    if P[0] == Q[0] and P[1] == -Q[1] % p:
        return (None, None)  # Протилежні точки

    if P != Q:
        lam = (Q[1] - P[1]) * pow(Q[0] - P[0], -1, p) % p
    else:
        lam = (3 * P[0]**2 + a) * pow(2 * P[1], -1, p) % p

    x_r = (lam**2 - P[0] - Q[0]) % p
    y_r = (lam * (P[0] - x_r) - P[1]) % p
    return (x_r, y_r)

# Множення точки на скаляр
def scalar_mult(k, P):
    R = (None, None)  # Нейтральний елемент
    while k:
        if k & 1:
            R = point_add(R, P)
        P = point_add(P, P)
        k >>= 1
    return R

# ECDSA: Генерація підпису
def ecdsa_sign(message_hash, d):
    while True:
        k = randint(1, n - 1)  # Випадкове число k
        R = scalar_mult(k, G)  # Обчислення точки R
        r = R[0] % n  # r = x_R mod n
        if r == 0:
            continue
        k_inv = pow(k, -1, n)  # Обчислення оберненого k mod n
        s = (k_inv * (message_hash + d * r)) % n
        if s != 0:
            break
    return (r, s)

# ECDSA: Перевірка підпису
def ecdsa_verify(message_hash, signature, Q):
    r, s = signature
    if not (1 <= r < n and 1 <= s < n):
        return False

    w = pow(s, -1, n)  # w = s^-1 mod n
    u1 = (message_hash * w) % n
    u2 = (r * w) % n
    P = point_add(scalar_mult(u1, G), scalar_mult(u2, Q))

    if P == (None, None):
        return False

    return P[0] % n == r

# Тестування алгоритму ECDSA
if __name__ == "__main__":
    # Генерація ключів
    d = randint(1, n - 1)  # Закритий ключ
    Q = scalar_mult(d, G)  # Відкритий ключ

    print("Закритий ключ:", d)
    print("Відкритий ключ:", Q)
```

```
# Підпис повідомлення
message_hash = 3  # Хеш повідомлення (для простоти задано)
signature = ecdsa_sign(message_hash, d)
print("Підпис:", signature)

# Перевірка підпису
is_valid = ecdsa_verify(message_hash, signature, Q)
print("Перевірка підпису:", "Успішно" if is_valid else "Помилка")
```

Опис роботи коду

1. Операції на еліптичній кривій:
    - point_add() реалізує додавання точок на кривій.
    - scalar_mult() дозволяє обчислювати скалярне множення точки PP на число kk.
2. ECDSA Підпис:
    - Використовується випадкове число kk і закритий ключ dd для створення підпису (r,s)(r, s).
3. ECDSA Перевірка:
    - Використовується хеш повідомлення, підпис (r,s)(r, s) та відкритий ключ QQ для перевірки правильності підпису.
4. Тестування:
    - Задані параметри кривої та фіксований хеш повідомлення.
    - Алгоритм генерує підпис і перевіряє його правильність.

Цей код є демонстраційним і використовує спрощені параметри кривої. Для реального застосування рекомендується використовувати стандартні криві, наприклад, secp256k1.

15.5 Рекомендації щодо оформлення звіту з роботи

- опис використовуваного методу,
- опис вихідних даних,
- алгоритм роботи програми,
- текст програми,
- результати роботи програми,
- аналіз результатів
- висновки.

15.6 Контрольні запитання

- Які властивості мають еліптичні криві, що роблять їх придатними для криптографії?
- У чому полягає суть проблеми дискретного логарифму на еліптичних кривих (ECDLP)?
- Як працює алгоритм ECDH для обміну ключами?
- У чому переваги ECC у порівнянні з RSA?
- Які сучасні протоколи використовують криптографію на еліптичних кривих?

15.7 Література

- Hankerson D., Vanstone S., Menezes A. *Guide to Elliptic Curve Cryptography*. – Springer, 2004.
- Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. – Wiley, 2020.

- Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Pearson, 2020.
- RFC 4492. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*.
- NIST FIPS PUB 186-4. *Digital Signature Standard (DSS)*.