

ННК «ІПСА»

Кафедра системного проектування

(повна назва кафедри, циклової комісії)

ЗВІТ

про виконання самостійної роботи

з дисципліни

"Безпека інформаційних систем"

(назва дисципліни)

на тему: «Аналіз атак на асиметричні системи та їх реалізація»

Студента 4 курсу

Групи ДА-82 Муравльова А.Д.

Керівник доц., к.т.н. Капшук О.О.

Київ - 2021

Зміст

ЗВІТ.....	1
1. Вступ.....	3
1.1 Модель криптографічної системи.....	3
1.2. Принцип Керкхоффа.....	5
1.3. Поняття криптографічної стійкості.....	5
1.4. Застосування атак методом «грубої сили».....	7
1.5. Поняття абсолютної криптостійкості шифрів.....	8
2. Модель асиметричної криптосистеми.....	10
2.1. Передумови виникнення асиметричних систем.....	10
2.2. Модель криптосистеми з публічними ключами.....	11
3. Основні алгоритми асиметричного шифрування.....	13
3.1. Поняття про алгоритм Діффі-Хелмана.....	13
3.2. Поняття про алгоритм RSA.....	15
4. Атаки на алгоритми асиметричного шифрування.....	16
4.1. Алгоритм Гельфорда-Шенкса(Baby-Step Giant-Step Algorithm).....	17
4.1.1. Визначення алгоритму.....	17
4.1.2. Оцінка складності алгоритму.....	18
4.1.3. Реалізація алгоритму.....	18
4.2 Р-Алгоритм Полларда.....	20
4.2.1. Опис алгоритму.....	20
4.2.2. Оцінка складності алгоритму.....	21
4.2.3. Реалізація алгоритму.....	21
4.3 Атака Вінера.....	24
4.3.1. Опис алгоритму.....	24
4.3.2. Складність алгоритму.....	24
5.Висновки.....	27
6. Література.....	28

1. Вступ

1.1 Модель криптографічної системи

Криптографічна система (криптосистема) – система секретного зв'язку, в якій зміст інформації, що передається, утаємничується за допомогою криптографічних перетворень; при цьому сам факт передачі інформації не приховується. Криптографічні перетворення визначаються певним параметром, який називається ключ. Зазвичай ключ є буквеною або числовою послідовністю. Кожне криптографічне перетворення однозначно визначається ключем і описується певним криптографічним алгоритмом. Криптографічними перетвореннями являються наступні операції:

- **Шифрування** - процес перетворення вихідного тексту (P) в зашифрований текст (C) за допомогою шифруючої функції (E) з секретним ключем шифрування (K_e) у відповідності з обраним алгоритмом шифрування: $C = E_{K_e}(P)$.
- **Розшифрування** - обернений шифруванню процес перетворення зашифрованого тексту (C) в вихідний текст (P) за допомогою функції розшифрування (D) з секретним ключем розшифрування (K_d) у відповідності з обраним алгоритмом шифрування: $P = D_{K_d}(C)$.

Сімейство обернених перетворень зашифрування і розшифрування називають **шифром**.

Алгоритми шифрування і розшифрування можуть відрізнятись, відповідно можуть розрізнятись і ключі шифрування і розшифрування.

В загальному випадку криптосистема має наступну структуру:

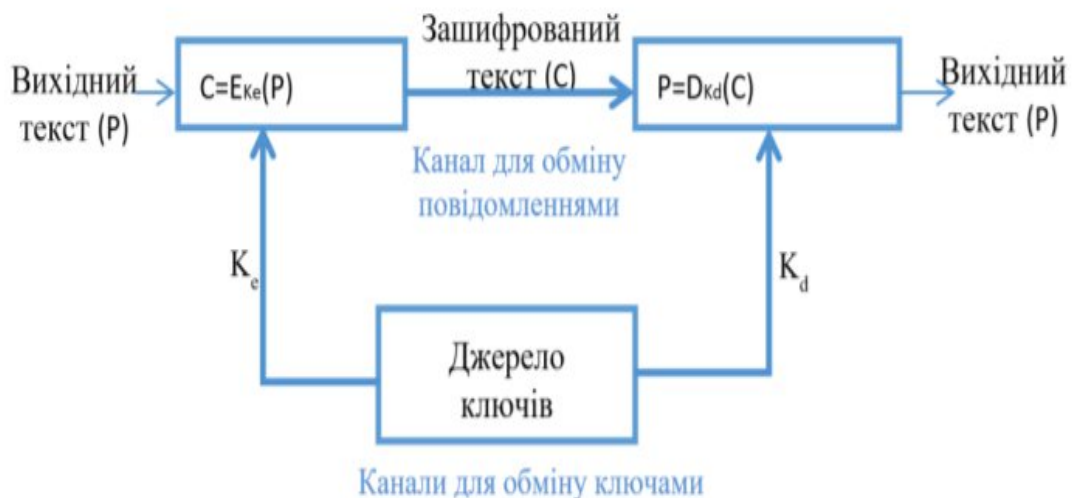


Рис. 1.1. Загальна схема криптосистеми

Робота криптосистеми може бути описана наступним чином:

1. З джерела ключів вибирається ключі (шифрування K_e і розшифрування K_d) і відправляються по надійним каналам передаючій і приймаючій стороні.
2. До вихідного (або відкритого) повідомлення p , що призначене для передачі, застосовується алгоритм шифрування E_{K_e} , внаслідок чого отримується зашифроване повідомлення $C = E_{K_e}(p)$.
3. Зашифроване повідомлення пересилається по каналу для обміну повідомленнями, який не вважається надійним (тобто зашифроване повідомлення може бути перехоплене порушником), приймаючій стороні.
4. На приймаючій стороні до зашифрованого повідомлення C застосовується обернене перетворення для отримання вихідного повідомлення $p = D_{K_d}(C)$.

В класичній криптографії для шифрування і розшифрування використовувався один і той самий ключ: $K_e = K_d = K$. Такі криптосистеми отримали назву криптосистем з секретним ключом (secret key cryptosystems); сьогодні їх також називають симетричним криптосистемами (symmetric cryptosystems). Зауважимо, що алгоритми шифрування і розшифрування E і D відкриті, і секретність вихідного тексту p в даному шифротекста C залежить від таємності ключа K .

Для сучасної криптографії революційним стало усвідомлення того факту, що ключі шифрування і розшифрування можуть не співпадати. Такі криптосистеми отримали назву асиметричних криптосистем (asymmetric cryptosystems).

Оскільки зашифроване повідомлення передається через канал, доступний противнику, можливе його перехоплення і «прочитання» особою, яка не має ключа. В цьому випадку говорять про **дешифрування** повідомлення.

Таким чином, терміни «розшифрування» і «дешифрування» слід розрізняти: при розшифровуванні ключ вважається відомим, тоді як при дешифруванні ключ невідомий.

Розшифрування здійснюється так само просто, як і шифрування. Дешифрування є значно складнішою задачею. Рівень складності цієї задачі і визначає здатність протистояти спробам противника заволодіти інформацією, яка захищається. В зв'язку з цим говорять про криптографічну стійкість шифру, розрізняючи більш і менш стійкі.

1.2. Принцип Керкхоффа

Основне правило криптографії - використовувати відкриті й опубліковані алгоритми та протоколи.

Вперше цей головний принцип був сформульований у 1883 році Агустом Керкхоффсом (A.Kerckhoffs): в криптографічній системі єдиним секретом має залишатися ключ, сам же алгоритм не повинен бути засекречений.

Сучасні криптологи повністю прийняли цей принцип, називаючи все, що йому не відповідає, "безпекою через неясність". Будь-яка система, що тримає в цілях безпеки свої алгоритми в секреті, просто ігнорується співтовариством і обзивається "ханаанським бальзамом".

Висновок з принципу Керкхоффа полягає в тому, що чим менше секретів містить система, тим вище її безпека. Якщо втрата будь-якого із секретів призводить до руйнування системи, то система з меншим числом секретів безумовно буде надійнішою. Чим більше секретів містить система, тим вона більш крихка. Менше секретів - вище міцність.

Обґрунтування принципу Керкхоффа полягає у такому твердженні: якщо для забезпечення безпеки системи криптоалгоритм повинен залишатися в таємниці, то система буде більш крихкою просто тому, що в ній виявиться більше секретів, які для забезпечення її безпеки потрібно зберігати.

1.3. Поняття криптографічної стійкості

Криптографічна стійкість (або криптостійкість) - здатність криптографічного алгоритму протистояти можливим атакам на нього.

Стійким вважається алгоритм, який для успішної атаки вимагає від противника:

- недосяжних обчислювальних ресурсів,
- недосяжного обсягу перехоплених відкритих і зашифрованих повідомлень чи
- такого часу розкриття, що по його закінченню захищена інформація буде вже не актуальна, і т. д.

Стійкість не можна підтвердити, її можна тільки спростувати зломом.

Для оцінки стійкості шифру використовують оцінку обчислювальної складності алгоритму успішної атаки на криптосистему.

Рівень криптостійкості - показник криптостійкості алгоритму, пов'язаний з обчислювальною складністю виконання успішної атаки на криптосистему. Зазвичай рівень криптостійкості вимірюється в бітах. N-бітний рівень криптостійкості криптосистеми означає, що для її злому потрібно виконати n

обчислювальних операцій. Наприклад, якщо симетрична криптосистема зламуються не швидше, ніж за повний перебір значень n -бітного ключа, то кажуть, що рівень криптостійкості дорівнює n .

Складність алгоритмів зазвичай оцінюють за часом виконання, але не менш важливі й інші показники - вимоги до обсягу пам'яті, вільного місця на диску. В теорії алгоритмів **обчислювальна складність алгоритму** - це функція, яка визначає залежність обсягу роботи, виконуваної деяким алгоритмом, від розміру вхідних даних.

У загальному випадку складність алгоритму можна оцінити по порядку величини. Алгоритм має складність $O(f(n))$, якщо при збільшенні розмірності n вхідних даних, час виконання алгоритму зростає з тією ж швидкістю, що і функція $f(n)$.

Використання великої літери O (або так звана O -нотація) прийшло з математики, де її застосовують для порівняння асимптотичної поведінки функцій. Формально $O(f(n))$ означає, що час роботи алгоритму (або обсяг займаної пам'яті) росте в залежності від обсягу вхідних даних не швидше, ніж деяка константа, помножена на $f(n)$.

Типові приклади використання O -нотації:

- $O(n)$ - лінійна складність. Таку складність має, наприклад, алгоритм пошуку найбільшого елемента в невідсортованому масиві. Нам доведеться пройти по всіх n елементах масиву, щоб зрозуміти, який з них максимальний.
- $O(\log n)$ - логарифмічна складність. Найпростіший приклад - бінарний пошук. Якщо масив відсортований, ми можемо перевірити, чи є в ньому якесь конкретне значення, шляхом поділу навпіл. Перевіримо середній елемент, якщо він більше шуканого, то відкинемо другу половину масиву - там його точно немає. Якщо ж менше, то навпаки - відкинемо початкову половину. І так будемо продовжувати ділити навпіл, в результаті перевіримо $\log n$ елементів.
- $O(n^2)$ - квадратична складність. Таку складність має, наприклад, алгоритм сортування вставками. У канонічній реалізації він вдає із себе два вкладених циклу: один, щоб проходити по всьому масиву, а другий, щоб знаходити місце чергового елемента уже відсортованої частини. Таким чином, кількість операцій буде залежати від розміру масиву як $n*n$.

Множина задач, для вирішення яких існують алгоритми, схожі за обчислювальною складністю, утворює клас складності.

Розрізняють такі основні класи складності:

- **Клас P** - вміщує всі ті проблеми, вирішення яких вважається «швидким», тобто поліноміально залежать від розміру входу (наприклад, сортування, пошук, з'ясування зв'язності графів і т.п.).
- **Клас NP** - містить задачі, які не в змозі вирішити за поліноміальну кількість часу (наприклад, факторизація, дискретне логарифмування та інші).

1.4. Застосування атак методом «грубої сили»

Найбільш загальний вид атаки на шифр – метод повного перебору (або метод «грубої сили»).

Повний перебір (англ. brute force) - загальний метод вирішення завдань шляхом перебору всіх можливих потенційних рішень.

У криптографії на обчислювальній складності повного перебору ґрунтується оцінка криптостійкості шифрів. Зокрема, шифр вважається крипостійким, якщо не існує методу «злому» істотно більш швидкого, ніж повний перебір всіх ключів. Криптографічні атаки, засновані на методі повного перебору, є найбільш універсальними, але і найдовшими.

Стійкість до brute-force атаки визначає використовуваний в криптосистемі ключ шифрування. Так, зі збільшенням довжини ключа складність злому цим методом зростає експоненціально. У найпростішому випадку шифр довжиною в n бітів зламуються, в найгіршому випадку, за час $O(2^n)$. Взагалі будь-яка задача з класу NP може бути вирішена повним перебором, але це вимагатиме експоненціального часу роботи.

Існують способи підвищення стійкості шифру до «brute force», наприклад заплутування (обфускація) шифрованих даних, що робить нетривіальним відмінність зашифрованих даних від незашифрованих.

Виходячи з фізичних міркувань можна показати, що існує можливість вибрати таку довжину ключа, що таку атаку методом «грубої сили» неможливо буде виконати в принципі, безвідносно до потужностей наявної обчислювальної техніки. Наведемо таке доведення:

За законами термодинаміки поглинання енергії при здійсненні незворотного перетворення має порядок $\sim kT$, де $k=1.4 \cdot 10^{-23}$ Дж/К, T – температура зовнішнього середовища.

Потужність сонячного випромінювання $P_s \sim 3.86 \cdot 10^{26}$ Вт. Час існування Всесвіту $t_s \sim 14$ млрд. років $\sim 14 \cdot 10^9$ років $\sim 4 \cdot 10^{19}$ с. Температура Сонця $T_s \sim 10^6$ К.

Тоді витрати на одну обчислювальну операцію $\sim kT = 1.4 \cdot 10^{-23} \cdot 10^6 \sim 1.4 \cdot 10^{-17}$ (Дж).

Вся енергія виділена Сонцем за час його існування Всесвіту $E = P_{\text{с}} \cdot t_{\text{с}} \sim 3.86 \cdot 10^{26} \cdot 4 \cdot 10^{19} \sim 1.6 \cdot 10^{46}$ (Дж). Тому можлива кількість виконаних операцій $N = E / kT \sim 1.6 \cdot 10^{46} / 1.4 \cdot 10^{-17} \sim 10^{63}$.

А це означає, що *при довжині ключа шифрування $L \geq 63$ перебір всіх варіантів не можна здійснити за час існування Всесвіту.*

1.5. Поняття абсолютної криптостійкості шифрів

Абсолютна криптостійкість означає, що:

- результатом розшифрування може бути будь-який текст,
- не існує ніякого способу перевірки, що розшифрування виконане правильно.

Доведення існування абсолютно стійких алгоритмів шифрування було виконано Клодом Шенноном та опубліковано в роботі «Теорія зв'язку в секретних системах» (1946 рік). Там же визначені вимоги до такого роду систем:

1. Ключ генерується для кожного повідомлення (кожен ключ використовується один раз).
2. Ключ статистично надійний (тобто ймовірності появи кожного з можливих символів однакові, символи в ключовій послідовності незалежні і випадкові).
3. Довжина ключа дорівнює або більше довжини повідомлення. Стійкість цих систем не залежить від того, якими обчислювальними можливостями володіє криптоаналітик.

Майже всі використовувані на практиці шифри характеризуються як умовно надійні, оскільки вони можуть бути розкриті в принципі при наявності необмежених обчислювальних можливостей. Абсолютно надійні шифри не можна зруйнувати навіть при наявності необмежених обчислювальних можливостей.

Єдиним відомим шифром, який задовольняє вимогам абсолютної криптостійкості, є шифр Вернама. Шифр Вернама (інша назва: англ. One-time pad - схема одноразових блокнотів) названий на честь телеграфіста АТ & Т Гільберта Вернама, який в 1917 році побудував телеграфний апарат, що виконував цю операцію автоматично - треба було тільки подати на нього стрічку з ключем.

Для шифрування відкритий текст «сумується» з ключем (який називається одноразовим блокнотом або шифроблокнотом) аналогічно розширеному шифру Цезаря.

Але при цьому ключ повинен задовольняти трьом критично важливими умовам:

1. Бути істинно випадковим.
2. Співпадати за розміром з заданим відкритим текстом.
3. Застосовуватись тільки один раз.

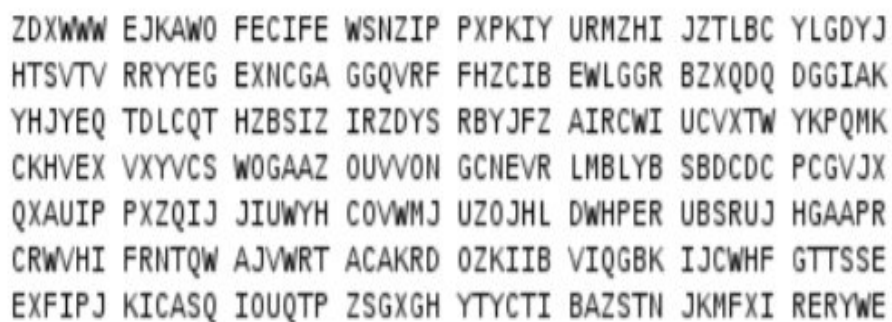
Будемо вважати, що число символів розширеного алфавіту, тобто сукупності літер, а також знаків пунктуації та пропусків між словами, дорівнює N . Занумерувавши всі символи розширеного алфавіту числами від 0 до N , текст, що передається, можна розглядати як послідовність $\{a_n\}$ чисел множини $A = \{0, 1, 2, \dots, N\}$.

Маючи випадкову послідовність $\{k_n\}$ з чисел множини A тієї ж довжини що і ключ, складаємо по модулю N число a_n переданого тексту з відповідним числом k_n ключа $a_n + k_n \equiv c_n \pmod{N}$, $0 \leq c_n \leq N$, одержимо послідовність $\{c_n\}$ знаків шифрованого тексту.

Щоб отримати переданий текст, можна скористатися тим ж ключем: $a_n \equiv c_n k_n \pmod{N}$, $0 \leq a_n \leq N$.

У двох абонентів, що знаходяться в секретному листуванні, є два однакових блокнота, складених з відривних сторінок, на кожній з яких надрукована таблиця з випадковими числами або буквами, тобто випадкова послідовність чисел з множини A . Таблиця має тільки дві копії: одна використовується відправником, інша - одержувачем.

Приблизна сторінка шифроблокноту показана на рис. 1.2.



ZDXWWW EJKAWO FECIFE WSNZIP PXPKIY URMZHI JZTLBC YLGDYJ
HTSVTV RRYEYG EXNCGA GGQVRF FHZCIB EWLGG R BZXQDQ DGGIAK
YHJYEQ TDLQCT HZBSIZ IRZDYS RBYJFZ AIRCWI UCVXTW YKPQMK
CKHVEX VXVCS WOGAAZ OUVVON GCNEVR LMBLYB SBDCDC PCGVJX
QXAUIP PXZQIJ JIUWYH COVWMJ UZOJHL DWHPER UBSRUJ HGAAPR
CRWVHI FRNTQW AJVWRT ACAKRD OZKIIB VIQGBK IJCWHF GTTSSE
EXFIPJ KICASQ IOUQTP ZSGXGH YTYCTI BAZSTN JKMFXI RERYWE

Рис.1.2. Приклад сторінки шифроблокноту

Відправник свій текст шифрує зазначеним вище способом за допомогою першої сторінки блокнота. Зашифрувавши повідомлення, він знищує використану сторінку і відправляє її іншому абоненту. Одержувач шифрованого тексту розшифровує його і також знищує використаний лист блокнота.

Неважко бачити, що одноразовий шифр не можна розкрити в принципі, так як символ у тексті може бути замінений будь-яким іншим символом і цей вибір абсолютно випадковий.

2. Модель асиметричної криптосистеми

2.1. Передумови виникнення асиметричних систем

Традиційні криптографічні системи мають два суттєвих недоліки:

- **Проблема розподілення ключів в управління ними.** Система з n учасниками вимагає використання $n/2$ ключів і стільки ж безпечних каналів для їх розповсюдження. При зміні ключа одним з учасників доводиться генерувати і розподіляти $(n-1)$ ключ. А додавання нового учасника вимагає генерування і розподілення n нових ключів.
- **Проблема автентифікації.** Традиційні криптосистеми не забезпечують потребу користувачів у використанні електронного еквіваленту підпису: будь-яке повідомлення, що відправлене одним з них, може бути відправлене і іншим.

Намагання усунути ці недоліки спонукало дослідників до пошуку криптографічних систем нового типу. Ідея такої системи була опублікована в 1976 р. у піонерській роботі У. Діффі і Д.Хеллмана «Нові напрями в криптографії».

Криптографічні системи нового типу отримали назву **криптосистем з публічними ключами** або **асиметричних криптосистем**.

Наведемо приклади проблем, які вирішуються за допомогою асиметричних систем.

- **Проблема зберігання паролів на комп'ютері.** Якщо зберігати паролі на магнітному диску, то зловмисник може прочитати їх і використати для несанкціонованого доступу. Тому необхідно організувати зберігання паролів на диску так, щоб унеможливити таке зчитування.
- **Проблема радіолокації в ППО.** При перетині літаком кордону у нього запитується пароль. Зловмисник може перехопити пароль і використати його для нелегального перетину кордону.
- **Проблема віддаленої взаємодії.** Так, при взаємодії банку з клієнтом на початку сеансу банк запитує клієнта ім'я і секретний пароль, який при використанні відкритого каналу зв'язку також може бути перехоплений.

2.2. Модель криптосистеми з публічними ключами

Ідея У. Діффі і Д.Хеллмана полягала у такому. Кожний користувач U криптосистеми створює (або отримує з надійного джерела) пару узгоджених алгоритмів P_u і S_u . Алгоритм P_u оголошується публічним, а алгоритм S_u

утримується в секреті. Ці алгоритми в залежності від застосування мають задовольняти окремим з наступних умов:

1. Алгоритми P_u і S_u ефективні, тобто не вимагають занадто великого часу обчислень і великих обсягів пам'яті.
2. $S_u(P_u(m))=m$ для будь-якого користувача U і будь-якого повідомлення m .
3. Неможливо виходячи з алгоритму P_u знайти такий алгоритм S_u^* , що $S_u^*(P_u(m))=m$ для всіх m .
4. $P_u(S_u(m))=m$ для будь-якого користувача U і будь-якого повідомлення m .
5. Неможливо виходячи з алгоритму P_u знайти такий алгоритм S_u^* , що $P_u(S_u^*(m))=m$ для всіх m .

Варто зазначити, що умови 3 та 5 неточні та їх зміст має уточнюватись в залежності від області застосування криптосистеми.

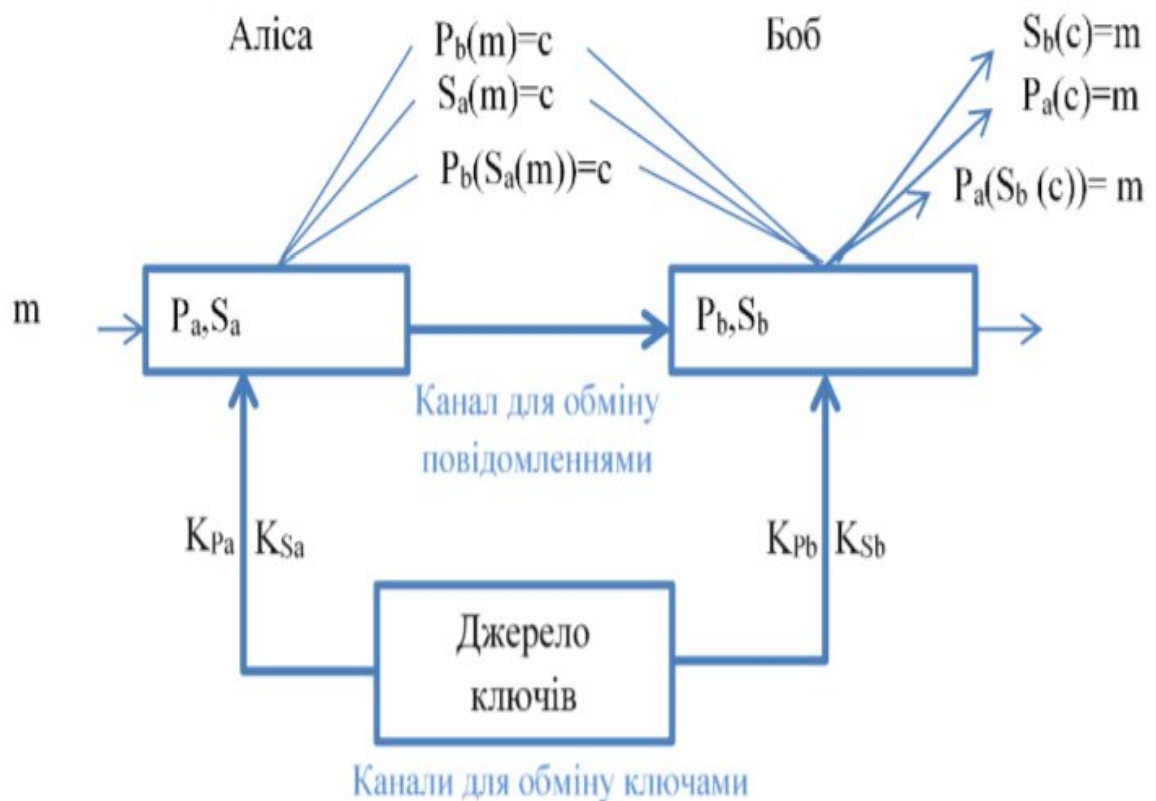


Рис. 2.1. Модель асиметричної криптосистеми

Розглянемо окремі випадки застосування цих умов.

1. Нехай виконуються умови 1-3.

В цьому випадку забезпечується можливість шифрування повідомлень. Дійсно, якщо Аліса хоче надіслати Бобу зашифроване повідомлення m , то вона спочатку знаходить публічний ключ Боба P_b і за його допомогою шифрує m : $c = P_b(m)$. Боб може відновити m ,

скористувавшись своїм секретним алгоритмом S_B , оскільки за умовою 2 $S_B(c) = S_B(P_B(m)) = m$.

Виконання умови 1 забезпечує практичність системи. А виконання умови 3 дозволяє публікування алгоритму утворення відкритого ключа.

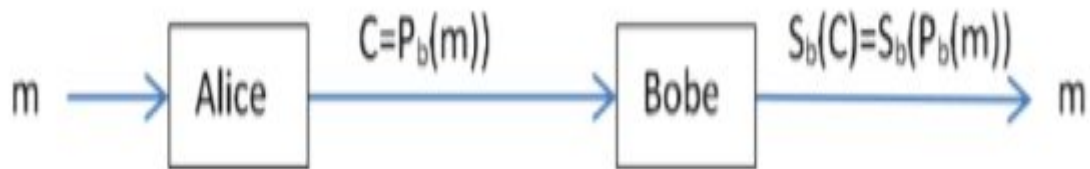


Рис. 2.2. Шифрування в асиметричній криптосистемі

2. Нехай виконуються умови 1, 4 та 5

В цьому випадку забезпечується можливість використання цифрового підпису.

Дійсно, якщо Аліса хоче надіслати Бобу підписане повідомлення m , то вона може спочатку застосувати до нього секретний алгоритм S_A і відправити $c = S_A(m)$. Боб може відновити m , скористувавшись публічним алгоритмом P_A , оскільки за умовою 4 $P_A(c) = P_A(S_A(m)) = m$. Виконання умови 1 забезпечує практичність системи. А виконання умови 5 дозволяє публікування алгоритму утворення цифрового підпису.

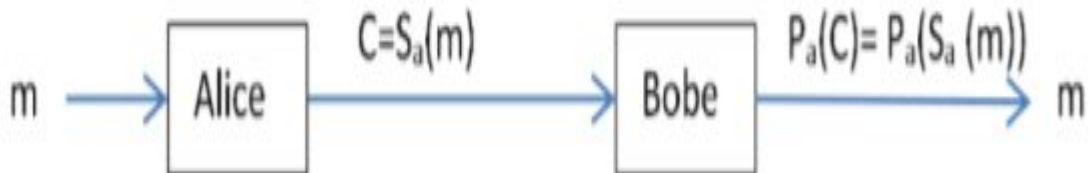


Рис. 2.3. Цифровий підпис в асиметричній криптосистемі

3. Нехай виконуються всі умови 1-5

В цьому випадку забезпечується можливість шифрування підписаних повідомлень.

Дійсно, якщо Аліса хоче надіслати Бобу повідомлення m в зашифрованому вигляді зі своїм підписом, то вона може спочатку підписати його за допомогою власного секретного алгоритму S_A , потім зашифрувати за допомогою публічного алгоритму Боба P_B і відправити $c = P_B(S_A(m))$. Боб може відновити m , скористувавшись публічним алгоритмом P_A і своїм секретним S_B , оскільки за умовою 2,4 $P_A(S_B(c)) = P_A(S_B(P_B(S_A(m)))) = P_A(S_A(m)) = m$. Виконання умови 1 забезпечує практичність системи. А виконання умови 3,5 дозволяє публікування алгоритми шифрування і утворення цифрового підпису.



Рис. 2.4. Шифрування підписаного повідомлення в асиметричній криптосистемі

3. Основні алгоритми асиметричного шифрування

3.1. Поняття про алгоритм Діффі-Хелмана

Проблема розподілу ключів була вирішена в тій же основній роботі Діффі і Хеллмана, в якій була розроблена схема шифрування з відкритим ключем. Цей протокол розподілу ключів, названий протоколом Діффі-Хеллмана обміну ключами, дозволяє двом сторонам узгодити секретний ключ по відкритому каналу зв'язку. Стійкість протоколу ґрунтується на складності проблеми дискретного логарифмування в кінцевій абелевій групі.

В алгоритмі Діффі-Хеллмана симетричний сеансовий ключ не генерується і не розподіляється між учасниками. Алгоритм забезпечує формування одного й того ж секрету двома сторонами, який можна використовувати для побудови сеансового ключа в симетричному алгоритмі. Ця процедура називається погодженням ключа: сторони погоджують ключ, який будуть використовувати. Вона полягає у такому: кожна із сторін отримує секретне і відкрите значення (пара ключів Діффі-Хеллмана); об'єднання секретного значення однієї із сторін з відкритим значенням іншої забезпечує створення одного й того ж самого секретного значення.

Протокол Діффі-Хеллмана спрощується, якщо зв'язок здійснюється тільки між двома абонентами. В цьому випадку:

1. Спочатку генеруються два великих простих числа p і q . Ці два числа не обов'язково зберігати в секреті.
2. Один з партнерів A генерує випадкове число x і посилає іншому учаснику B значення $L = Q^x \pmod{N}$.
3. Після отримання L партнер B генерує випадкове число y і посилає A обчислене значення $M = Q^y \pmod{N}$.
4. Партнер A , отримавши M , обчислює $K_x = M^x \pmod{N}$, а партнер B обчислює $K_y = L^y \pmod{N}$.

Алгоритм гарантує, що $K_y = K_x$ і можуть бути використані в якості секретного ключа для шифрування. Адже навіть перехопивши числа L і M , важко обчислити K_x або K_y .

Основні повідомлення в протоколі Діффі-Хеллмана представляються наступною діаграмою :

$A \leftrightarrow B: N, Q;$

$A: x; A \rightarrow B: L = Q^x \pmod{N};$

$B: y; B \rightarrow A: M = Q^y \pmod{N};$

$A: K_x = M^x \pmod{N};$

$$B: K_y = L_y \pmod{N};$$

Розглянута система має суттєвий недолік: у Аліси немає впевненості, що вона листується саме з Бобом. Це може привести до наступної атаки, яка умовно називається «людина посередині»:

- Аліса домовляється про ключ з Євою, думаючи, що переписується з Бобом;
- Боб веде переговори про ключ з Євою, вважаючи, що переписується з Алісою;
- Єва може вивчати повідомлення, т.я. вони проходять через неї як через комутатор. Оскільки вона не вносить змін в відкритий текст, її дії не можуть бути виявлені.

Отже, можна зробити висновок про те, що самого по собі протоколу Діффі-Хеллмана не достатньо для забезпечення секретності розподілення ключів. Але на основі протоколу Діффі-Хеллмана можна створити справжню криптосистему.

3.2. Поняття про алгоритм RSA

Система RSA, названа на честь його розробників Ріверса (Ron Rivers), Шаміра (Adi Shamir) і Адлемана (Leonard Adleman).

Система ґрунтується на використанні іншої односторонньої функції. В цій системі використовуються наступні факти з теорії чисел:

- Задача перевірки числа на простоту є порівняно простою.
- Задача розкладання числа $n=p*q$, де p і q – прості числа, на множники є дуже складною задачею, якщо ми знаємо тільки n , а p і q – великі числа (задача факторизації).

В групі абонентів A, B, C, \dots кожний абонент:

1. Вибирає випадково два великих простих числа P і Q і обчислює $N = PQ$.
2. Обчислює число $f = (P-1)(Q-1)$.
3. Вибирає деяке число $d < f$, взаємно просте з f , і за розширеним алгоритмом Евкліда знаходить таке c , що $cd \pmod{f} = 1$

В результаті отримуємо таблицю ключів:

Абонент	Секретний ключ	Відкритий ключ
A	P_A, Q_A, c_A	N_A, d_A
B	P_B, Q_B, c_B	N_B, d_B

C	P_C, Q_C, e_C	N_C, d_C
...

Нехай А хоче передати В повідомлення $m < N_B$. Протокол передачі складається з таких кроків:

1. Абонент А шифрує повідомлення, використовуючи відкриті параметри абонента В: $e = m^{(d_B)} \bmod N_B$
2. Абонент В, отримавши зашифроване повідомлення, обчислює $m' = e^{(e_B)} \bmod N_B$.

Відмітимо, що для RSA важливо, щоб кожний абонент вибирав власну пару простих чисел P і Q , тобто всі N мають різнитись, інакше один абонент міг би читати повідомлення, призначені для іншого. Проте інший відкритий параметр d може бути однаковим у всіх абонентів. Часто рекомендується $d=3$. Тоді шифрування виконується максимально швидко – всього за дві операції множення.

4. Атаки на алгоритми асиметричного шифрування

З визначення алгоритму Діффі-Хелмана очевидно, що зломисник, який має доступ до каналу зв'язку при спробі втрутитись в обмін даними може себе видати. Таким чином, перед ним постає обчислювально складна задача визначення чисел x та y маючи доступ лише до значень L та M , іншими словами – проблема дискретного логарифмування. Така ж задача стоїть перед зломисником, який намагається зламати інші асиметричні криптосистеми – такі, як RSA. Ця задача належить до класу складності NP та не може бути вирішена шляхом простого перебору через свою обчислювальну складність за умови, що числа p , q , x та y є досить великими. В цьому підрозділі опишемо та наведемо реалізації алгоритмів, які дають змогу вирішити цю проблему.

4.1. Алгоритм Гельфорда-Шенкса(Baby-Step Giant-Step Algorithm)

4.1.1. Визначення алгоритму

Очевидно, найпростішим методом вирішення рівняння виду $a^x = b \pmod N$ є просте перемноження a до тих пір, поки не задовольнимо умову рівняння:



Рис. 4.1. Найпростіший спосіб вирішення проблеми дискретного логарифму

З рисунку 4.1. очевидно, що в такому випадку складність операції буде оцінюватись порядком $O(n)$. Для пришвидшення рішення даної задачі був введений алгоритм Гельфорда-Шенкса.

В рамках алгоритму спочатку для деякого k визначаються значення a^k , a^{2k} , a^{3k} і т.д. Це так звані «великі кроки»:

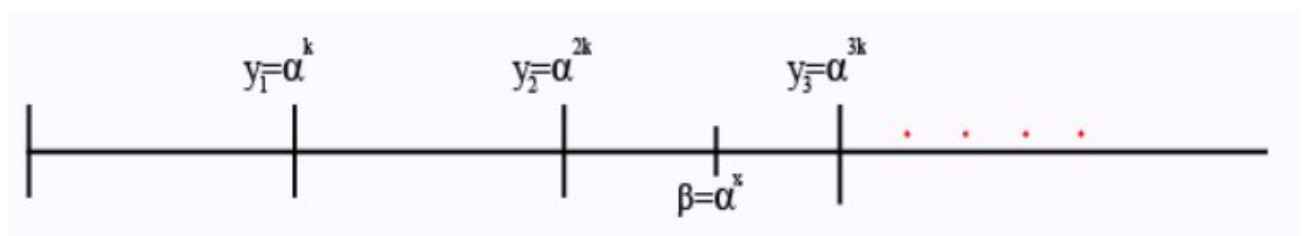


Рис. 4.2. «Великі кроки» алгоритму Гельфорда-Шенкса

Потім, на другій стадії алгоритму, визначаються добутки $b * a$, $b * a^2$, $b * a^3$ і т.д. Це так звані «малі кроки»:

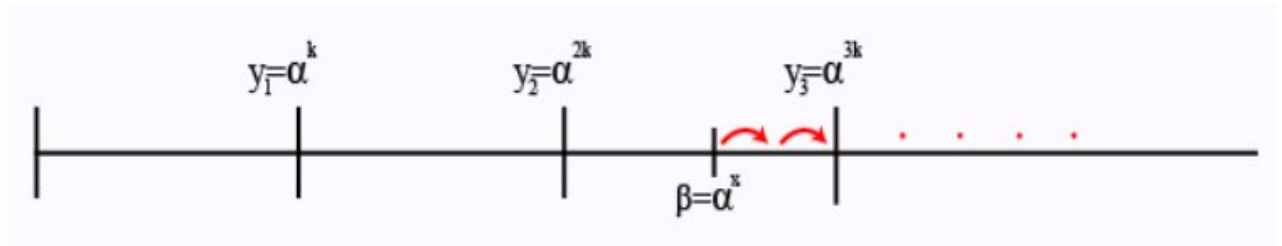


Рис. 4.3. «Малі кроки» алгоритму Гельфорда-Шенкса

З ілюстрації очевидно, що в певний момент добуток $b * a^t$ стане рівним якомусь з y_m , які ми обчислили на попередньому кроці. Таким чином, маємо рівняння $a^x * a^t = y_m = a^m * k$, звідки при логарифмуванні отримуємо $x = mk - t$.

4.1.2. Оцінка складності алгоритму

З визначення алгоритму можемо зробити висновок, що кількість необхідних кроків являє собою функцію від обраної величини поділу k виду

$$f = k + \frac{N}{k}$$

Знайдемо мінімум даної функції:

$$f' = 1 - \frac{N}{k^2}$$

$$1 - \frac{N}{k^2} = 0 \implies N = k^2 \implies k = \sqrt{N}$$

Якщо представити N у вигляді експоненти e^n , то складність алгоритму можемо визначити $L \sim O(e^{n/2})$ як, де $n = \ln N$. Варто зазначити, що у простого перебору (метод грубої сили) складність буде становити $L \sim O(e^n)$.

4.1.3. Реалізація алгоритму

```
from math import ceil, sqrt

def bsgs(g, y, p, verify: bool = False):
    """
    Алгоритм Гельфорда-Шенкса
    :param g: g з рівняння g^x = y (mod n). g > 0.
    :param y: y з рівняння g^x = y (mod n). Не може бути від'ємним.
    :param p: p з рівняння g^x = y (mod p). Просте число.
    :returns: x з рівняння g^x = y (mod n). Якщо такого числа не знайдено,
    повертає 1.
    """
```

```

if verify:
    assert is_prime(p), "p має бути простим числом"
else:
    print("Алгоритм виконується з припущенням що p - просте число.")
m = ceil(sqrt(p))

# В даній хешмані зберігаємо обчислені  $p^j$ 
table = dict()

# Заповнюємо хешману та визначаємо всі  $g^j$ 
g_raised_to_j = 1
for j in range(0, m):
    table[g_raised_to_j] = j
    g_raised_to_j = (g_raised_to_j * g) % p

# Маємо визначити  $g^{-im}$ 
# для цього спочатку визначаємо  $g^{-m}$  та підносимо його до степеню в циклі
g_raised_to_minus_m = pow(g, p-(m+1), p)
# змінна зберігає  $g^x * g^{-im}$ 
temp = y
for i in range(0, m):
    if temp in table:
        return (i * m) + table[temp]
    temp = (temp * g_raised_to_minus_m) % p

return -1

def is_prime(n: int) -> bool:
    """
    Перевіряє, чи є n простим числом
    :param n: ціле число > 1.
    :returns: true якщо n просте число, інакше false
    """
    assert n > 1, "Вхідний параметр повинен бути > 1"

    if n in [2, 3, 5, 7]:
        # n - просте число
        return True
    if n % 2 == 0 or n % 3 == 0:
        # має дільником 2 або 3
        return False
    # sqrt(n) є верхньою межею для дільника
    upper_bound = ceil(n ** 0.5)
    divisor = 5
    # кожне просте число окрім 2 та 3
    # має вигляд 6k +- 1
    # тому починаємо з 5 та збільшуємо дільник на 6
    while (divisor <= upper_bound):
        if n % divisor == 0 or n % (divisor + 2) == 0:
            return False
        divisor += 6

```

```
return True
```

4.2 Р-Алгоритм Полларда

4.2.1. Опис алгоритму

р-алгоритм Джона Полларда, запропонований ним в 1975 році, служить для факторизації цілих чисел. Він ґрунтується на алгоритмі Флойда пошуку довжини циклу в послідовності і деяких наслідках з парадоксу днів народжень. Алгоритм найбільш ефективний при факторизації складених чисел з досить малими множниками в розкладанні. Складність алгоритму оцінюється як $O(N^{1/4})$.

У всіх р-методах Полларда будується числова послідовність, елементи якої утворюють цикл, починаючи з деякого номера n , що може бути проілюстровано, розташуванням чисел у вигляді грецької букви ρ . Це і послужило назвою для сімейства методів.

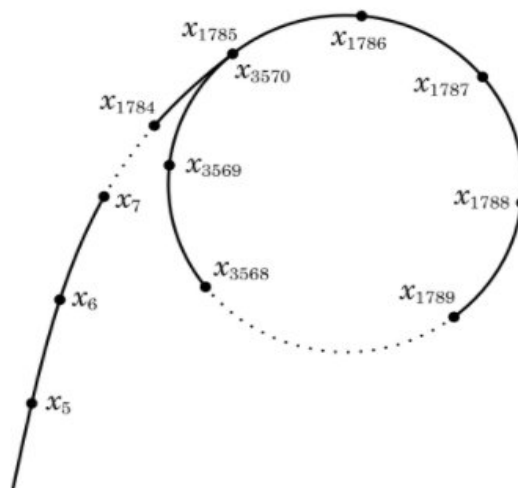


Рис. 4.4. Числова послідовність в алгоритмі Полларда.

Нехай N – складене ціле додатне число, яке потрібно розкласти на множники.

Алгоритм виглядає таким чином:

1. Обираємо невелике число x_0 та будуємо послідовність $\{x_n\}$, $n = 0, 1, 2, \dots$, визначаючи кожне наступне як $x_{n+1} = F(x_n) \pmod{N}$.
2. Одночасно на кожному i -му кроці обчислюємо $d = \text{НОД}(N, |x_i - x_j|)$ для будь-яких i, j таких, що $j < i$.
3. Якщо виявили, що $d > 1$, то обчислення закінчується і знайдене на попередньому кроці d є дільником N . Якщо N/d не є простим числом, то процедуру пошуку дільників можемо продовжити, взявши як N число $N^* = N / d$.

На практиці функція $F(x)$ не повинна бути надто складною для обчислення, але в той же час не повинна бути лінійним многочленом, а також не повинна

породжувати взаємо однозначне відображення. Зазвичай за функцію $F(x)$ беруть функцію $F(x) = x^2 + 1 \pmod{N}$.

4.2.2. Оцінка складності алгоритму

Щоб оцінити складність алгоритму, можна розглядати послідовність, що будується в процесі обчислень, як випадкову (звісно, ні про яку строгість при цьому говорити не можна). Щоб повністю факторизувати число N довжиною b біт, достатньо знайти всі його дільники, що не переважають його квадратного кореню, що вимагає максимум порядку $2^{b/4}b^2$ бітових операцій. Тому складність алгоритму оцінюється як $O(N^{1/4})$, однак при цій оцінці не враховуються витрати по обчисленню найбільшого спільного дільника. Отримана складність алгоритму, хоча і не є точною, досить точно узгоджується з практикою.

4.2.3. Реалізація алгоритму

```
import random

from math import gcd

def __compute_next(x: int, g: int, y: int, a: int, b: int, p: int, order: int):
    if x % 3 == 0:
        x = (x * x) % p
        a = (2 * a) % order
        b = (2 * b) % order
    elif x % 3 == 1:
        x = (x * y) % p
        b = (b + 1) % order
    else:
        x = (x * g) % p
        a = (a + 1) % order
    return x, a, b

def pollard_rho(g: int, y: int, p: int, order: int = 0, tries: int = 10, verify: bool = False) -> int:
    if verify:
        assert is_prime(p), "Pollard-rho works for prime p"
    else:
        print("WARNING: Pollard-rho running assuming passed n is prime. If not, answer may be wrong.")

    if order == 0 or order is None:
        print("WARNING: Order has not been provided to Pollard Rho, answer might not be the smallest one.")
        order = p-1

    if y == 1:
        return 0
```

```

def __pollard_rho(g: int, y: int, p: int, order: int, a0: int, b0: int):

    x0 = (pow(g, a0, p) * pow(y, b0, p)) % p
    xi, ai, bi = x0, a0, b0
    x2i, a2i, b2i = x0, a0, b0

    for i in range(1, p):
        xi, ai, bi = __compute_next(xi, g, y, ai, bi, p, order)
        x2i, a2i, b2i = __compute_next(x2i, g, y, a2i, b2i, p, order)
        x2i, a2i, b2i = __compute_next(x2i, g, y, a2i, b2i, p, order)

        if (xi == x2i):
            beta = (b2i - bi) % order
            alpha = (ai - a2i) % order

            if beta == 0:
                return -1

            xs = solve_linear_congurence(beta, alpha, order)
            for x in xs:
                if pow(g, x, p) == y:
                    return x

    return -1

a0, b0 = 0, 0
x = __pollard_rho(g, y, p, order, a0, b0)
if x >= 0:
    return x

for _ in range(tries):
    a0, b0 = random.randint(1, order), random.randint(1, order)
    x = __pollard_rho(g, y, p, order, a0, b0)
    if x >= 0:
        return x

return x

def mod_inverse(g: int, n: int) -> int:

    assert g > 0 and n > 1, "Inappropriate values to compute inverse"

    g = g % n

    _, v, g = extended_euclidean_algorithm(n, g)

    if g != 1:
        print("Inverse doesn't exist")
        exit()
    else:

```

```

        v = v % n
        return v

def solve_linear_congurence(a: int, b: int, m: int):

    g = gcd(a, m)
    if b % g != 0:
        print("Recurrence can not be solved")
        return -1

    a, b, m = a//g, b//g, m//g

    a_inverse = mod_inverse(a, m)

    x = (b * a_inverse) % m

    xs = [x + (i*m) for i in range(0, g)]
    return xs

def is_prime(n: int) -> bool:
    assert n > 1, "Вхідний параметр повинен бути > 1"

    if n in [2, 3, 5, 7]:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    upper_bound = ceil(n ** 0.5)
    divisor = 5
    while (divisor <= upper_bound):
        if n % divisor == 0 or n % (divisor + 2) == 0:
            return False
        divisor += 6
    return True

def extended_euclidean_algorithm(a: int, b: int) -> tuple[int, int, int]:
    s, old_s = 0, 1
    t, old_t = 1, 0

    r, old_r = b, a

    while (r != 0):

        q = old_r // r

        old_r, r = r, old_r % r

        old_s, s = s, old_s - q * s
        old_t, t = t, old_t - q * t

    return old_s, old_t, old_r

```

4.3 Атака Вінера

4.3.1. Опис алгоритму

Атака Вінера, названа так в честь криптолога Майкла Дж. Вінера являє собою тип криптографічної атаки на алгоритм RSA. Алгоритм використовує метод неперервного дробу для злому системи при малому значенні експоненти d .

Маючи відкритий ключ RSA (e, N) необхідно визначити закриту експоненту d . Якщо відомо, що $d < 1/3N^{1/4}$, то цього можливо досягти за допомогою даного алгоритму:

1. Розкласти дріб e / N в неперервний дріб $[a_1, a_2, \dots]$
2. Для неперервного дробу $[a_1, a_2, \dots]$ знайти множину всіх можливих наближених дробів k_n / d_n
3. Дослідити наближений дріб k_n / d_n :
 - a. Визначити можливе значення $f_i(N)$, обчисливши $f_n = (ed_n - 1) / k_n$
 - b. Вирішити рівняння $x^2 - ((N - f_n) + 1)x + N = 0$, отримати пару коренів (p_n, q_n) .
4. Якщо для пари коренів (p_n, q_n) виконується рівність $N = p_n * q_n$, то закрита експонента(приватний ключ) знайдена: $d = d_n$.
5. Якщо умова не виконується або пару коренів знайти не вдалось, то необхідно дослідити наступний наближений дріб k_{n+1} / d_{n+1} , повернувшись на крок 3.

4.3.2. Складність алгоритму

Складність алгоритму визначається кількістю наближених дробів для неперервного дробу числа e / N , яка являє собою величину порядку $O(\log(n))$, тобто приватний ключ d визначається за лінійний час від довжини ключа.

4.3.3 Реалізація алгоритму

```
from typing import Tuple, Iterator, Iterable, Optional

def isqrt(n: int) -> int:
    if n == 0:
        return 0

    x = 2 ** ((n.bit_length() + 1) // 2)
    while True:
```

```

    y = (x + n // x) // 2
    if y >= x:
        return x
    x = y

def is_perfect_square(n: int) -> bool:
    sq_mod256 =
(1,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,
0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1
,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0
,0,0,0,0,0,0,1,0,0,0,0,0,0)
    if sq_mod256[n & 0xff] == 0:
        return False

    mt = (
        (9, (1,1,0,0,1,0,0,1,0)),
        (5, (1,1,0,0,1)),
        (7, (1,1,1,0,1,0,0)),
        (13, (1,1,0,1,1,0,0,0,0,1,1,0,1)),
        (17, (1,1,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1))
    )
    a = n % (9 * 5 * 7 * 13 * 17)
    if any(t[a % m] == 0 for m, t in mt):
        return False

    return isqrt(n) ** 2 == n

def rational_to_contfrac(x: int, y: int) -> Iterator[int]:
    while y:
        a = x // y
        yield a
        x, y = y, x - a * y

def contfrac_to_rational_iter(contfrac: Iterable[int]) -> Iterator[Tuple[int,
int]]:
    n0, d0 = 0, 1
    n1, d1 = 1, 0
    for q in contfrac:
        n = q * n1 + n0
        d = q * d1 + d0
        yield n, d
        n0, d0 = n1, d1
        n1, d1 = n, d

```



```

def convergents_from_contfrac(contfrac: Iterable[int]) -> Iterator[Tuple[int,
int]]:
    n_, d_ = 1, 0
    for i, (n, d) in enumerate(contfrac_to_rational_iter(contfrac)):
        if i % 2 == 0:
            yield n + n_, d + d_
        else:
            yield n, d
        n_, d_ = n, d

def attack(e: int, n: int) -> Optional[int]:
    f_ = rational_to_contfrac(e, n)
    for k, dg in convergents_from_contfrac(f_):
        edg = e * dg
        phi = edg // k

        x = n - phi + 1
        if x % 2 == 0 and is_perfect_square((x // 2) ** 2 - n):
            g = edg - phi * k
            return dg // g
    return None

```

5.Висновки

В розділі 1 дано визначення асиметричної криптосистеми, проаналізовано проблеми, які покликана вирішувати така система на відміну від класичних симетричних систем, зокрема проблему автентифікації, описано принципи побудови та функціонування криптографічної системи.

Також в першому розділі приведено поняття криптостійкості алгоритму, абсолютної криптостійкості та криптографічної функції та описано основні принципи роботи асиметричних криптосистем.

В розділі 2 побудовано узагальнену модель криптографічної системи, визначено умови для її надійного функціонування та цілі її застосування, приведено та проаналізовано випадки, в яких можуть не виконуватись деякі з визначених раніше умов та застосовування таких криптосистем.

В розділі 3 описано основні алгоритми асиметричного шифрування, на яких будуються сучасні асиметричні криптосистеми, а саме алгоритм Діффі-Хелмана та RSA. В цьому розділі також визначено деталі функціонування алгоритмів, які можуть становити найбільшу цікавість для злоумисника, а отже можуть являти собою основні вектори атаки на алгоритми. Розділ 3 також описує проблему дискретної логарифмізації, яка являє собою основним математичним фактом, який робить задачу злomu таких алгоритмів надзвичайно складною обчислювальною задачею та відносить її до NP-повних задач.

В розділі 4 описано основні принципи побудови алгоритмів для злomu асиметричних криптосистем та слабкі місця алгоритмів, надано текстовий опис найбільш поширених та відомих атак. Також в даному розділі проаналізовано обчислювальну складність приведених алгоритмів атак, що є одним із головних критеріїв до роботи такого виду, оскільки обчислювальна складність є однією з основних запорук криптостійкості таких алгоритмів.

Всі приведені алгоритми атак були реалізовані на мові програмування Python, а їх працездатність перевірена.

6. Література

1. https://en.wikipedia.org/wiki/Wiener%27s_attack
2. [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
3. https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%93%D0%B5%D0%BB%D1%8C%D1%84%D0%BE%D0%BD%D0%B4%D0%B0_%E2%80%94%D0%A8%D0%B5%D0%BD%D0%BA%D1%81%D0%B0
4. https://ru.wikipedia.org/wiki/%D0%90%D1%82%D0%B0%D0%BA%D0%B0_%D0%92%D0%B8%D0%BD%D0%B5%D1%80%D0%B0
5. https://uk.wikipedia.org/wiki/P-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D0%BE%D0%BB%D0%B0%D1%80%D0%B4%D0%B0#%D0%9E%D0%BF%D0%B8%D1%81_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%83
6. <https://cyberleninka.ru/article/n/realizatsiya-metoda-faktorizatsii-pollarda-na-yazyke-c>
7. https://er.nau.edu.ua/bitstream/NAU/22464/2/diser_ua_2.0.pdf