Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
за спеціальністю 122 «Комп'ютерні науки»

на тему:

Криптографічні алгоритми з використанням випадкових чисел

Студента 4-го курсу
Шаргородського Арсенія
Олексійовича

_____
(підпис)

Науковий керівник:
доктор фізико-математичних
наук, професор
Пашко Анатолій Олексійович

_____
(підпис)

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

Студент _____
(підпис)

Робота заслухана на засіданні кафедри теоретичної кібернетики та рекомендована до захисту в ЕК, протокол № …… від ..... 2021р.

Завідувач кафедри Крак Ю.В.

Київ – 2021

РЕФЕРАТ

Ключові слова: КРИПТОГРАФІЧНІ АЛГОРИТМИ, ГЕНЕРАТОРИ ВИПАДКОВИХ ЧИСЕЛ, АЛГОРИТМ XOR.

Об’єктом дослідження є алгоритми криптографічного захисту інформації, що використовують випадкові числа.

В якості засобу розроблення програми було обрано мову програмування Python.

Метою роботи є підвищення ефективності алгоритмів шифрування, що використовують випадкові числа.

Методи розроблення: Генерація випадкових чисел за допомогою генераторів випадкових чисел, їх порівняльна характеристика, використання послідовності для алгоритму XOR, результати роботи у вигляді зашифрованого тексту.

Інструменти розроблення: мова програмування Python.

Результати роботи: Виконано аналіз роботи алгоритму XOR, проведено порівняльну характеристику ГВЧ різних бібліотек Python за статичними тестами NIST, реалізовано програму для шифрування, застосовуючи XOR з ключем у вигляді випадкової двійкової послідовності.

Отримані результати дають можливість порівнювати вже існуючи криптографічні алгоритми з використанням випадкових чисел з даною програмною реалізацію. Дослідницька частину можна використати в якості перевірки інших ГВЧ на якість випадковості генерованої послідовності.

Новизна полягає в шифруванні відомого алгоритму XOR з ключем достатньо крипто стійким для злому.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ГВЧ – генератор випадкових чисел

ГПВЧ – генератор псевдовипадкових чисел

ЗМІСТ

ВСТУП

В 21 столітті інформація стала одним із важливішим і дорогоцінішим ресурсом для людей. Так і захист інформації, можна сказати, володіє світом.

Перше, що ми робимо зранку – перевіряємо смартфон, шукаємо інформацію, заходимо в соціальні мережі, робимо онлайн покупки та безліч інших функцій в Інтернеті. Тому ви повинні бути впевнені, що захищені від несанкціонованого доступу та порушень безпеки. Якщо казати про компанії та організації, то вони теж потребують захисту від кібератак та загроз безпеки. Шкідливе програмне забезпечення та кіберзлочинність є постійною загрозою для всіх, хто кожен день використовує Інтернет. За допомогою інформаційної безпеки організація може захистити інформацію та технології, реагуючи, запобігаючи та виявляючи внутрішні та зовнішні загрози.

Кажучи про захист інформації, слід розповісти про криптографію – науку о математичних методах забезпечення конфіденційності, цілісності і автентичності інформації. Але це сучасне визначення криптографії, бо Найперша відоме використання криптографії знайдено в нестандартних ієрогліфах, вирізаних на стіні гробниці з Стародавнього царства Єгипту близько 1900 року до нашої ери. А після, в часи Стародавнього Риму криптографія знадобилась в шифруванні повідомлень в спілкуванні з генералами під час військових компаній. Сам шифр використовувався Юлієм Цезарем, в честь нього і назвали цей шифр.

На протязі декількох тисячоліть ми знаємо про існування такої науки як криптографія і саме в наші дні вона є дуже актуальною. Майже кожного року з’являються новини про кібератаки великих корпорацій і компаній, останніх з яких була зв’язано з блокчейном.

Метою роботи є розробка криптографічного алгоритму з використанням випадкових чисел.

Для виконання поставленої мети слід виконати такі завдання:

1. Проаналізувати літературу по темі криптографічні алгоритми
2. Проаналізувати поняття криптографічний алгоритм, його ознаки і методи його створення
3. Обрати один із типів криптографічних алгоритмів для створення і описати.

РОЗДІЛ 1. ОСНОВНІ ПОНЯТТЯ. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ.

1.1. Криптографія

Криптографія - це наука про шифри. Криптографія оперує такими поняттями як:

• Відкритий текст, або текст який ми плануємо зашифрувати

• Шифротекст - це текст, який ми отримаємо після шифрування

• Ключ - це секретна інформація, що використовується для шифрування тексту. Саме ключ ми намагаємося тримати в таємниці, щоб тільки одержувач, яким ми хочемо відправити повідомлення, міг перевести зашифрований текст за допомогою цього ключа.

Криптографія пов'язана з різними галузями математики (алгебра, теорія чисел, теорія імовірності, теорія складності, обчислювальна математика та ін.), з теорією зв'язку, а також з численними технічними дисциплінами, що створюють фундамент для побудови апаратури та програмного забезпечення захисту даних і злому шифрів.

Криптографічний захист даних забезпечується їх перетворенням, яке може бути представлене функцією з множини входів в множину виходів. Ця функція може бути як оборотною, так і необоротною. Крім того, вона може залежати від змінного параметра – ключа, який зазвичай буває секретним. Така функція називається криптографічним примітивом. [1]

Тому з поняття про криптографічний примітив випливає, що при спробах злому цього самого примітива можно визначати його стійкість.

В інформатиці – криптографія є набором з методів захисту інформації, створених під математичною базою і також алгоритмами, що дозволяють різними способами зашифрувати інформацію. Всі ці алгоритми використовуються в багатьох сферах нашого життя для захисту нашої конфіденційної інформації.

1.2. Поняття криптографічного алгоритму

Криптографічний алгоритм – це набір математичних правил, які виконуються в деякій послідовності для шифрування і дешифрування тексту.

Тому з вище сказаного вкажемо на які діляться на типи криптографічні алгоритми:

• Криптографія симетричного ключа: Це система шифрування, коли відправник та одержувач повідомлення використовують єдиний загальний ключ для шифрування та дешифрування повідомлень. Системи симетричних ключів швидші та простіші, але проблема полягає в тому, що відправник і одержувач повинні якимось чином обмінюватися ключами в безпечному порядку. Найпопулярнішою системою криптографічного симетричного ключа є система шифрування даних (DES).

• Хеш-функції: У цьому алгоритмі не використовується жоден ключ. Хеш-значення з фіксованою довжиною обчислюється відповідно до звичайного тексту, що унеможливлює відновлення вмісту звичайного тексту. Багато операційних систем використовують хеш-функції для шифрування паролів.

• Криптографія асиметричного ключа: За цією системою пара ключів використовується для шифрування та дешифрування інформації. Відкритий ключ використовується для шифрування, а приватний - для дешифрування. Відкритий та приватний ключі відрізняються. Навіть якщо відкритий ключ відомий кожному, призначений приймач може його розшифрувати лише тому, що він один знає приватний ключ.

Також вкажемо ознаки криптографічного алгоритму:

- Конфіденційність:

Доступ до інформації може мати лише особа, для якої вона призначена, і жодна інша особа, крім неї, не може отримати до неї доступ.

- Цілісність:

Інформацію не можна змінювати при зберіганні або переході між відправником та передбачуваним одержувачем без виявлення будь-якого доповнення до інформації.

- Безвідмовність:

Творець / відправник інформації не може заперечувати свій намір надіслати інформацію пізніше.

- Автентифікація:

Особи відправника та одержувача підтверджуються. А також підтверджується пункт призначення / походження інформації.

1.3. Різновиди криптографічних алгоритм з використанням випадкових чисел

Далі слід розібратись в криптографічних алгоритмах, які використовують випадкові числа, щоб краще розуміти позначення та терміни, пов’язані з моєї програмною реалізацією.

Спочатку наведемо основні поняття щодо генерації випадкових чисел:

Випадкове число – число, яка представляю собою реалізацією випадковою величини.

Детермінований алгоритм – алгоритм, який повертає те ж саме вихідне значення при тих же самих вхідних значеннях.

Псевдовипадкове число – число, отримане детермінованим алгоритмом, використане в якості випадкового числа. [2]

Якщо алгоритм здатен створити послідовність, за якою можна побачити закономірність з законом розподілу, то такий алгоритм можно назвати генератором випадкових бітів або генератор випадкових чисел.

На основі попереднього твердження, генератор псевдовипадкових чисел – це алгоритм, який генерує двійкову послідовність що має відношення одиниць та нулів майже таке ж, як і послідовність випадкових чисел.

Перелік криптографічних алгоритмів, які можуть використовуватися як генератор псевдовипадкових чисел:

1. Блочний шифр

Шифр, який шифрує дані в блоки, використовуючи детермінований алгоритм та симетричний ключ. [3]

Але в якості генератора псевдовипадкових чисел, блочний шифр нас цікавить у випадку його запуску в режимі лічильника. До цього типу шифрів відносяться “AES”, “TwoFish”, “Serpent”, “Camellia”

2. Потоковий шифр

Шифр, який шифрує один біт або байт відкритого тексту за раз.

Отримати псевдовипадкову двійкову послідовність за допомогою потокового шифру в разі його ініціалізації на послідовності натуральних чисел.

До цих типу відносяться “Salsa20” та “ChaCha”.

3. Хеш-функції

В цьому випадку достатньо щоб вихідне значення лічильника не було відоме.

І тоді в якості генератора псевдовипадкових чисел, підійде алгоритм

“SH-256”, “POCO”. [4]

1.4. Статистичні тести NIST

Якщо для шифрування використовуються криптографічний алгоритм з використанням випадкових чисел, то випадкові числа генеруються за допомогою генератора псевдовипадкових чисел. Але виникає питання наскільки такий генератор створює “випадкові” числа? Тому ми маємо перевірити їх на крипто стійкість, і в цьому нам допоможе статичні тести NIST.

Статистичні тести NIST – це набір із 15 статистичних тестів, розроблені для перевірки випадковості двійкової послідовності, утворені завдяки псевдовипадкового генератора випадкових чисел. Також можуть створені за допомогою апаратного генератора чисел. [3]

Перелік тестів NIST:

1. Частотний побітовий тест;

2. Частотний блоковий тест;

3. Тест на однакові біти, що йдуть поряд;

4. Тест на найдовшу послідовність з одиниць в блоці;

5. Тест рангів бінарних матриць;

6. Спектральний тест;

7. Тест на відповідність шаблону, що не перекривається;

8. Тест на відповідність шаблону накладання;

9. "Універсальний статистичний" тест Маурера;

10. Тест на лінійну складність;

11. Серійний тест;

12. Приблизний тест на ентропію;

13. Тест кумулятивних сумм;

14.Тест на випадкові відхилення;

15. Різновидність тесту на випадкове відхилення;

Так як всі ці тести описані на сайті “National Institute of Standards and Technology U.S. Department of Commerce”, то я опишу шість класичних тестів NIST нижче.

1. Частотний побітовий тест

Якщо наша двійкова послідовність випадкова, то кількість одиниць в цій послідовності має бути близько ½, маючи на увазі що і нулів теж близько ½. Тому тест аналізує, наскільки це відношення близьке до одиниці.

Кожен “0” приймається як -1 і кожен “1” як +1, а далі беремо суму всієї двійкової послідовності.

Сума записується таким чином:

$$S = X_1 + X_2 + ... + X_n, X_i = 2x_i - 1$$

Алгоритм перевірки частого побітового тесту:

1. Рахуємо статистику за формулою:
   $S_{obs} = \frac{|S|}{\sqrt{n}}$, де n – це довжина двійковій послідовності, яку ми перевіряємо.
2. Знаходимо за допомогою додаткової функції помилок P-значення:
   $P_{value} = erfc(\frac{S_{obs}}{\sqrt{2}})$
   Якщо результат матиме значення більше ніж 0.01, тоді можна підтвердити, що тест пройдено успішно.

2. Частотний блоковий тест

Цей тест спирається на попередній тест і тепер пропорція одиниці до нуля кожного блока перевіряється за допомогою Хі-квадрат.

Алгоритм перевірки частотного блокового тесту:

1. Вираховуємо статистику, використовуючи метод Хі-квадрат зі N степенями свободи.

$$X_2^{obs} = 4 \cdot M \cdot \sum_{i=1}^{N} (\pi_i - 1/2)^2$$

N – кількість блоків.

π – пропорція одиниць і нулів в блоці, тобто у випадку якщо блок з 3 цифр – 010, то їх відношення дорівнюватиме π = 1/3

M – довжина кожного блоку.

2. Використовуючи спеціальну функцію Q, знаходимо P-значення:

$$P_{value} = Q(\frac{N}{2}, \frac{x_{obs}^2}{2})$$

Q визначається як неповна верхня гамма-функція:

$Q(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt$, де Г(a) – гамма функція.

Висновки щодо двійкової послідовності виконується на результатах P-значення, як і в попередньому тесті.

3. Тест на однакові біти, що йдуть поряд

Цей тест знаходить в двійковій послідовності однакові біти, які звіряє з кількістю та розміром еталонної випадкової послідовності. Тобто якщо в послідовності рідко відбувається черговість з 1 на 0 або 0 на 1, то така послідовність тест підтвердить як не випадковою.

Алгоритм перевірки:

1. Вираховуємо відношення кількості одиниць для всієї послідовності:

$$\pi = \frac{\sum_j X_j}{n}$$

2. Перевірка умови:

$$\left| \pi - \frac{1}{2} \right| < \frac{2}{\sqrt{n}}$$

Якщо умова не виконується, тоді перевірка зупиняється і тест вважається неуспішним. З цього випливає, що не має сенсу перевіряти цей послідовність на випадковість, якщо не було пройдено перший тест – частотний побітовий тест.

3. Знаходимо загальну кількість чисел зі зміною знака:

$$V_n = \sum_{k=1}^{n-1} r(k) + 1$$

$r(k) = 0$, якщо $X_i = X_{i+1}$, або r(k) = 1 в іншому випадку.

4. Рахуємо P-значення за допомогою функції помилок:

$$P_{value} = erfc(\frac{|V_n - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)})$$

Результат має бути більше 0.01, тоді послідовність є випадковою.

4. Тест на найдовшу послідовність з одиниць в блоці

В цьому випадку знову звіряється з еталонною випадковою послідовність, але перед цим вхідну послідовність розділяють на n бітів, які в свою чергу розбиваються на N блоків, де в кожному по M біт, і вже в кожному блоці знаходять найдовшу послідовність одиниць.

Алгоритм перевірки:

1. Спочатку розбивають дану послідовність на блоки, після чого рахується найбільша послідовність з одиниць до кожного блока.

Якщо виникає питання яку ж взяти довжину блок, то NIST пропонує такі значення для поділу:

Загальна довжина, n	**Довжина блока, M**
128	8
6272	128
750000	10000

2. Розраховується статистика по всім довжинам за допомогою таблиці нижче:

vi	M=8	M=128	M=10000
v0	≤1	≤4	≤10
v1	2	5	11
v2	3	6	12
v3	≥4	7	13
v4		8	14
v5		≥9	15
v6			≥16

3. Знаходимо Хі-квадрат:

$$X^2 = \sum_{i=0}^{K} \frac{(v_i - R\pi_i)}{R\pi_i}$$

Підібрати потрібне значення K і R по цій таблиці:

M	K	R
8	3	16
128	5	49
10000	6	75

Знаходимо P-значення:

$$P_{value} = igamc(\frac{K}{2}, \frac{X^2_{obs}}{2})$$

Якщо P-значення більше 0.01, послідовність є випадковою.

5. Тест рангів бінарних матриць

Якщо побудувати матриці з вхідної послідовності, а потім розрахувати ранг непересічних підматриць, аналізувати послідовність на випадковість за формулою.

$$P_R = 2^{R(Q+M-R)-MQ} \prod_{i=0}^{R-1} \frac{(1-2^{i-Q})(1-2^{i-m})}{1-2^{i-R}}$$

Алгоритм перевірки тесту рангів бінарних матриць:

1. Спочатку ділимо послідовність на блоки за такою формулою:

$$N = \left[\frac{n}{MQ}\right]$$

n – довжина біта.

M – кількість рядків в кожній матриці.

Q – кількість стовпців в кожній матриці.

2. Розкладемо послідовність з блоку на матриці, а з них будемо вже вираховувати ранги матриць.

3. Знаходимо Хі-квадрат:

$$X^2 = \sum \frac{(F_i - NP_i)^2}{NP_i}$$

4. Знаходимо P-значення:

$$P_{value} = igamc(1, \frac{X^2_{(obs)}}{2})$$

На результатах P-значення, якщо більше 0.01, то послідовність є випадковою.

6. Спектральний тест

Розглянута послідовність представляється у вигляді дискретного сигналу і на основі цього робимо спектральний розклад для того, щоб знайти частотні піки. Тест перевіряє, чи ці піки переходять межу в 95% і після цього перевіряє, чи кількість цих піків не більше ніж 5%.

Алгоритм перевірки спектрального тесту:

1. Створюємо суму періодичних складових за допомогою формули Фур'є з послідовності:

$$X_j = \sum_{k=0}^{n-1} x_k e^{\frac{2\pi_i k_j}{n}}$$

2. Дізнаємось граничне значення формулою:

$$T = \sqrt{log(\frac{1}{0.05} \cdot n)}$$

3. Знаходимо кількість піків, що є граничною, які є менші ніж T:

$$N_0 = \frac{0.95n}{2}$$

4. Аналізуємо різницю між досліджуваною послідовністю та очікуваним числом частоти компонентів, які знаходяться за межею 95%

$$d = \frac{N_1 - N_0}{\sqrt{n \cdot 0.95 \cdot 0.05/4}}$$

5. Знаходимо P-значення:

$$P_{value} = erfc(\frac{|d|}{\sqrt{2}})$$

На основі P-значення, якщо воно більше ніж 0.01, то послідовність є випадковою.

1.5. Висновки до розділу 1-4

Розібравшись що із себе представляє криптографія для людства в наші часи, стає зрозуміло наскільки серйозно ми маємо ставитись до захисту нашої особистої інформації, які ми залишаємо в Інтернеті, соціальних мережах і т.д.

Криптографічні алгоритми також ще можуть використовуватися для цифрового підпису і частіше всього як метод захисту приватності для “IoT” домену.

За різновидом криптографічних алгоритмів є очевидним, в якій випадках потрібен той чи іншій алгоритм, і за яким принципом він працює. А вже ознаками криптографічного алгоритму визначається яким основним пріоритетам має служити алгоритм і без яких він вже не є повноцінним.

Досліджуючи криптографічні алгоритми з використанням випадкових чисел, генератори випадкових та псевдовипадкових чисел стали проривом в розробці криптографічних алгоритмів, тому зараз велика увага приділена випадковості як засіб поліпшення крипто стійкості існуючих алгоритмів або створення нових.

Звісно у разі використання ГВЧ або ГПВЧ, потрібно дізнатися наскільки ці генератори є випадковими. Для перевірки генераторів створили статичні тести, що дозволяють на основі створеною генератором послідовності робити аналіз. Існують різні статичні тести такі як:

- DIEHARD
- TEST-U01
- CRYPT-X
- NIST

Ці тести працюють за принципом визначення P-значення за гіпотезою, що еталонний генератор створить послідовність менш випадкову ніж вхідна. Якщо P-значення більше критичного значення, то послідовність не є випадковою. В роботі описано класичні тести NIST і далі вони будуть використовуватися для перевірки на випадковість двійкової послідовності.

РОЗДІЛ 2. Методи та алгоритми.

2.1. Опис алгоритму XOR

XOR є однією з найбільш базових операцій в комп'ютері через двійковій (бінарної) логіки. На її основі працюють більш комплексні і стійкі алгоритми шифрування, які іспользуються в інтернеті, месенджери і т.д.

XOR - це операція з двійковій логіки. Двійкова логіка - це логіка, яка працює над двійковими числами в двійковій системі числення. Для того щоб розібратися як працює XOR, потрібно в першу чергу розібрати операції OR і AND.

Таблиця операції OR:

a	b	a∨b
0	0	0
0	1	1
1	0	1
1	1	1

Візьмемо числа 4 і 5, ці числа з десятиричное системи обчислення. Переводимо їх в двійкову систему: 4 = 100, 5 = 101. Операція OR дуже схожа на додавання. Додавання відбувається порозрядно, справа наліво і переповнює значення не переноситься в наступний розряд (тобто 1 + 1 = 1).

Таблиця операція AND:

a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1

Дуже схоже на множення. Числа просто перемножаются.

Таблиця операції XOR:

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

В останньому прикладі, можна уявити:

• 100 - слово

• 101 - ключ

• 001 - результат шифрування

У комп'ютері текст, відео, аудіо, будь-яка інформація представлена як послідовність чисел, тому шифрування справедливо для будь-якої інформації.

Перед шифруванням, інформація переводитися в двійкову систему або вона безпосередньо є двійковій системою і застосовується алгоритм. Алгоритмом неважливо, які саме дані будуть шифруватися.

Спробуємо розшифрувати наш результат

Беремо зашифровані дані - 001, і застосовуємо до нього ключ – 101:

Дані	Ключ	Результат шифрування
0	1	1
0	0	0
1	1	0

Ми отримали знову слово, яке шифрували спочатку.

2.2. Генератори псевдовипадкових чисел

Джерела для створення випадкових чисел є навіть якщо спостерігати за навколишнім середовищем. Тому щоб створити дійсно випадкові числа, достатньо вимірювати випадкові коливання, відомий як шум. У разі проведення вимірювання шуму або дискретизацію, ми отримаємо числа.

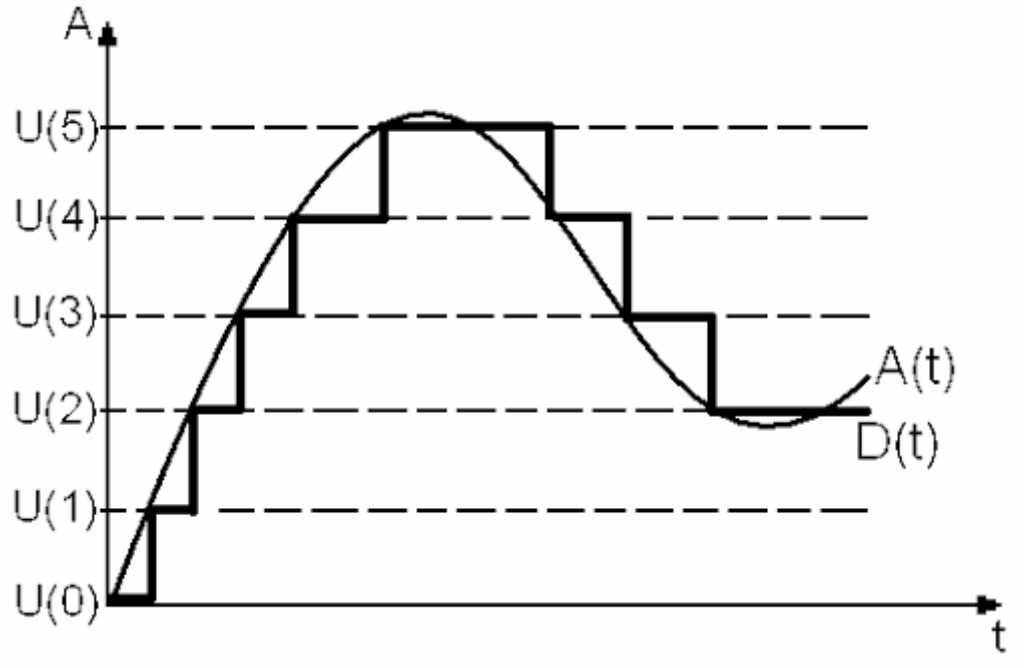


Рис. 2.0 – Дискретізація сигнала

Наприклад, якщо вимірювати статичній електронний потік монітора телевізора деякий час, ми зможемо генерувати дійсно випадкову послідовність.

Випадкові процеси називають не детерміністичними, тому що їх неможливо визначити заздалегідь. В той же час, машини є детерміністичними. Їх операції є передбачуваними і повторюваними.

В 1946 році, Джон фон Нейман брав участь в розробці водневої бомби. Його задача полягала в багаторазовому підрахунку апроксимацій в процесах атомного поділу за допомогою комп’ютеру ENIAC. Для цього було потрібно мати доступ до швидкої генерації чисел стільки, скільки було потрібно створити. Але ENIAC мав обмежену внутрішню пам'ять, тому Нейман створив алгоритм, який механічно імітував ознаки випадковості.

Для початку потрібно взяти дійсно випадкове число, що називали “насінням”. Джерело цього числа може бути будь-яке: вимірювання шуму(дискретизація) або даний час в мілісекундах. Після цього це “насіння” використовували як вхідне значення для простого розрахунку.

Алгоритм розрахунку:

1. Вхідне значення помножити саме на себе.
2. З вихідного значення першої операції взяти середину.
3. Потім це число знову помножити себе на себе.
4. Повторювати стільки цю процедуру скільки потрібно.

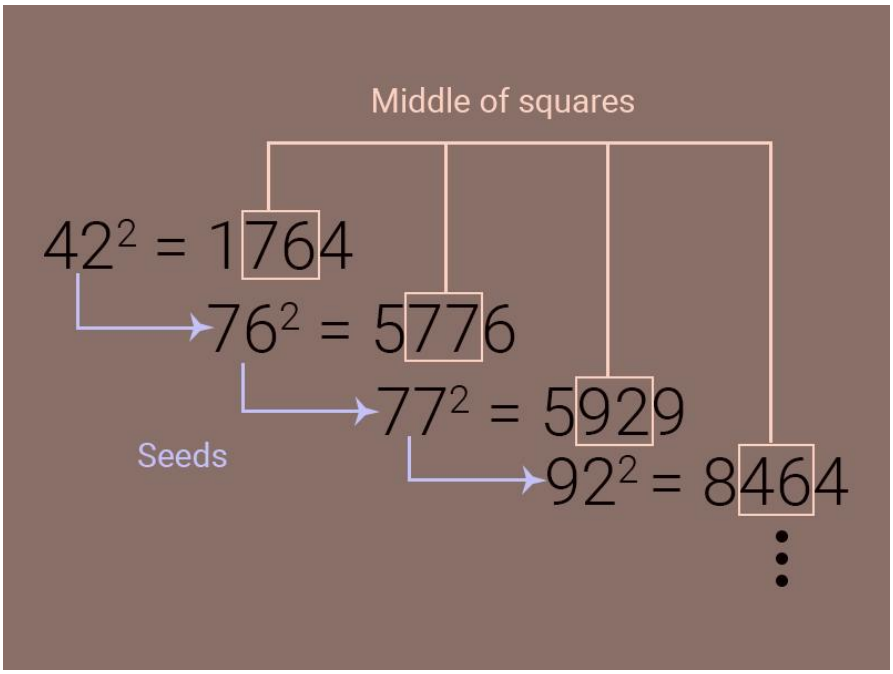


Рис. 2.1 Метод середніх квадратів

Таким чином можно сгенерувати одну з послідовностей за допомогою генератора псевдовипадкових чисел. Випадковість цієї послідовності напряму залежить від випадковості “насіння” і тільки.

В чому тоді різниця між псевдовипадковою генерованою послідовністю і випадковою?

Представимо кожну з послідовностей у вигляді випадкового процессу “випадкового блукання”. Цей термін описує шлях, що відбувається завдяки послідовним випадковим крокам. Тоді спостерігаючи за цим, можна побачити як псевдовипадкова послідовність починає повторюватися, в той час як випадкова послідовність продовжує свій шлях випадковим чином.

Чи можливо створити таку псевдовипадкову послідовність, яка б не відрізнялась від випадковою послідовності?

Так, це можливо. Для цього потрібно створити таку псевдовипадкову послідовність, яку комп’ютер не зможе перевірити усі “насіння” на збіг.

2.3. Лінійний конгруентний метод

Один із найбільш відомих алгоритмів, які зустрічаються для генерації псевдовипадкових чисел. Цей алгоритм генерує псевдовипадкову послідовність за формулою:

$$X_{n+1} = X_n \cdot A + B(modM)$$

В цій формулі для створення X_{n+1} , множимо константу A на попереднє випадкове число X_n і додаємо ще одну константу B , беремо результат і ділимо з залишком на третю константу M.

Значення, які використовуються для ініціалізації генератора випадкових чисел також називають значенням насіння. Якщо його змінити, то ми отримаємо іншу послідовність чисел.

Також слід зазначити такий термін для лінійного конгруентного методу як повний період. Повний період можна отримати у випадку, коли за алгоритмом генерована послідовність має повтори чисел, починаючи з деякого значення.

Умови появи повного періоду:

- B та M мають бути взаємно простими, в них не має бути загальних дільників, крім одиниці.
- $A - 1$ повинно ділитися на всі прості дільника M.
- $A - 1$ має бути кратно чотирьом, якщо M кратне чотирьом.

Представимо приклад, які не виконує ці умови:

$$A = 4$$

$$B = 3$$

$$M = 8$$

$$x_0 = 0$$

$$x_1 = 0 \cdot 4 + 3(mod8) = 3(mod8) = 3$$

$$x_2 = 3 \cdot 4 + 3(mod8) = 15(mod8) = 7$$

$$x_3 = 7 \cdot 4 + 3(mod8) = 31(mod8) = 7$$

Переваги та недоліки лінійного конгруентного методу:

- Швидкість

Для генерації випадкової послідовність цей алгоритм є достатньо швидким, так як потребує лише декілька операцій: одне множення, одне додавання і ділення з залишком. Кожна з цих операцій займає фіксований час і створення n чисел займе порядку n шагів.

- Не завжди генерує всі можливі числа

Такий висновок можна зробити с прикладу, які був розібраний раніше. Тому є вірогідність того, що в послідовності буде створена не всі можливі числа, до того, як вони будуть повторюватися.

- Послідовність не є досить випадковою

В разі, якщо знати значення констант A, B, M і одне із значень в послідовності. то можна обчислити інші значення з послідовності.

Якщо в прикладі було використано достатньо малі числа, які можна порахувати достатньо легко, то на практиці для більшого періоду використовують достатньо велике число.

Нижче наведено таблицю “гарних” значень для констант в лінійних конгруентних генераторах для різних компіляторів мов програмування:

Мови програмування	A	B	M
Microsoft Visual Basic	1140671485	12820163	2^{24}
Microsoft Visual C++	214013	2531011	2^{32}
Borland C/C++	22695477	1	2^{32}

2.4. Алгоритм RSA

Перша стаття об алгоритмі RSA була опублікована в 1977 році. Абревіатура RSA є розшифровкою прізвищ авторів даного алгоритму – Рон Рівест, Аді Шамір, Леонард Адельман. Ці вчені з Массачусетського технологічного інституту, які довго працювали над пошуком математичної функції, яка б дозволила реалізувати модель криптографічної системи з

відкритим ключем. З 40 варіантів, які вони опрацьовували, був обраний один, яким користуються і сьогодні.

RSA – криптографічний алгоритм асиметричного шифрування. В ньому використовуються приватний та публічний ключ. Вони формуються при використанні простих чисел.

Етап	Опис операції	Результат операції
Генерація ключів	Обрати два простих різних числа	$p = 3557,$ $q = 2579$
	Обчислити добуток	$n = p \cdot q = 3557 \cdot 2579 = 9173503$
	Обчислити функцію Ейлера	$\varphi(n) = (p-1)(q-1) = 9167368$
	Обрати відкриту експоненту	$e = 3$
	Обчислити секретну експоненту	$d = e^{-1} \pmod{\varphi(n)}$ $d = 6111579$
	Опублікувати *відкритий ключ*	$\{e, n\} = \{3, 9173503\}$
	Зберегти *секретний ключ*	$\{d, n\} = \{6111579, 9173503\}$
Шифрування	Обрати текст для шифрування	$m = 111111$
	Обчислити шифротекст	$c = E(m)$ $= m^e \mod n$ $= 111111^3 \mod 9173503$ $= 4051753$
Розшифрування	Обчислити вихідне повідомлення	$m = D(c) =$ $= c^d \mod n$ $= 4051753^{6111579} \mod 9173503$ $= 111111$

Рис. 2.2 Алгоритм роботи RSA

2.5. Висновки до розділу 1-4

Дослідження по різним типам криптографічних алгоритмам було проведено. Ознайомившись з теоретичною частиною криптографічних алгоритмів, в якості прикладу для демонстрації роботи криптографічного алгоритму буде обрано алгоритм XOR в наступному розділі.

Слід додати, що за саме методом лінійного конгруентного методу здійснено ініціалізація ГВЧ або ГПВЧ в різних мовах програмування.

РОЗДІЛ 3. Реалізація на Python алгоритмів формування випадкових чисел

3.1. Інструменти для програмної розробки

Для програмної реалізації обрана мова програмування Python, яка популярна в дослідженнях, пов'язані з природничими науками, зокрема криптографії. Вона підходять для розробки алгоритму XOR, створений за допомогою вбудованих методів. Також для проектування ключа для криптографічного алгоритму є багато бібліотек такі як – cryptography, random. В своїй програмі я використовую random як генератор випадкових чисел, бо cryptography генерує числа, які є недостатньо крипостійкими, перевіряючи на статичних тестах NIST.

Звісно є інші мови програмування для розробки програми для поліпшення продуктивності такі C++ або Java, але для дослідницьких цілей Python є зрозумілою с низьким порогом входження. Документацію є в Інтернеті, яка описано детально розписано та з багатьма прикладами для кращого розуміння в вивченні.

Серед відомих середовищ розробки обрано PyCharm. Хоча в разі розробки програми для наукових досліджень частіше всього обирають Jupyter, але програмної реалізації краще демонструвати результати та лістинг коду в PyCharm для ліпшого сприйняття.

3.2. Алгоритм XOR на Python

Реалізувати бітову операцію XOR на Python достатньо легко, бо вона є вбудованою в стандартні бібліотеки Python. Але має інший вигляд, відрізняючись від класичного виду – "^".

Так як в попередньому розділі повністю розібрано як працює операція XOR, то далі буде виконано шифрування без використання випадкової чи псевдовипадкової послідовності чисел.

Реалізація XOR на Python:

```
key = '0'
text_for_encode = 'Hello world!'

def binary(num, length=8):
    b = bin(num).lstrip("0b")
    b = "0" * (length - len(b)) + b
    return b


keyLength = len(key)
cipherBin = ""

for i in range(0, len(text_for_encode)):
    j = i % keyLength
    xor = ord(text_for_encode[i]) ^ ord(key[j])
    cipherBin = cipherBin + binary(xor) + " "

print("\nCipher (Binary form): \n" + cipherBin)
```

Рис. 3.0 – Алгоритм XOR на мові програмування Python

1. Ключ має зміст в один нуль, тому кожен символ з відкритого тексту матиме операцію XOR кожен раз с нулем.

2. Після змінних ключа та тексту, який ми шифруємо, написана функція в подальшому перетворить вихідні числа у вигляді двійкової послідовності.

3. Далі в описаному циклі йде перебір по всій довжині відкритого тексту, і в змінній xor для кожної ітерації береться по одному символу з відкритого тексту та ключа, і до кожного з них застосовується метод “ord”, який повертає число, яке репрезентує цей символ в таблиці Unicode. І вже тоді застосовується операції XOR

4. В кінці результати змінної xor переносяться cipherBin і вже виводиться результат.

```
Cipher (Binary form):
01111000 01010101 01011100 01011100 01011111 00010000 01000111 01011111 01000010
```

Рис. 3.1 – Зашифрований текст

В наступних розділах цю реалізацію XOR на Python буде вдосконалене таким чином:

- Зміст ключа буде містить випадкову двійкову послідовність, створеною генератором випадкових чисел, яка буде перевірена статичними тестами NIST.
- Зміст відкритого тексту збільшиться, від слова до тексту, тобто буде подаватися файл.

3.3. Статистичні тести NIST на Python

В першому розділі було описано перші шість класичних тестів NIST, а саме їх алгоритм перевірки. В цьому розділі розглянуто функціонал програми з 15-ю тестами NIST реалізованих на Python.

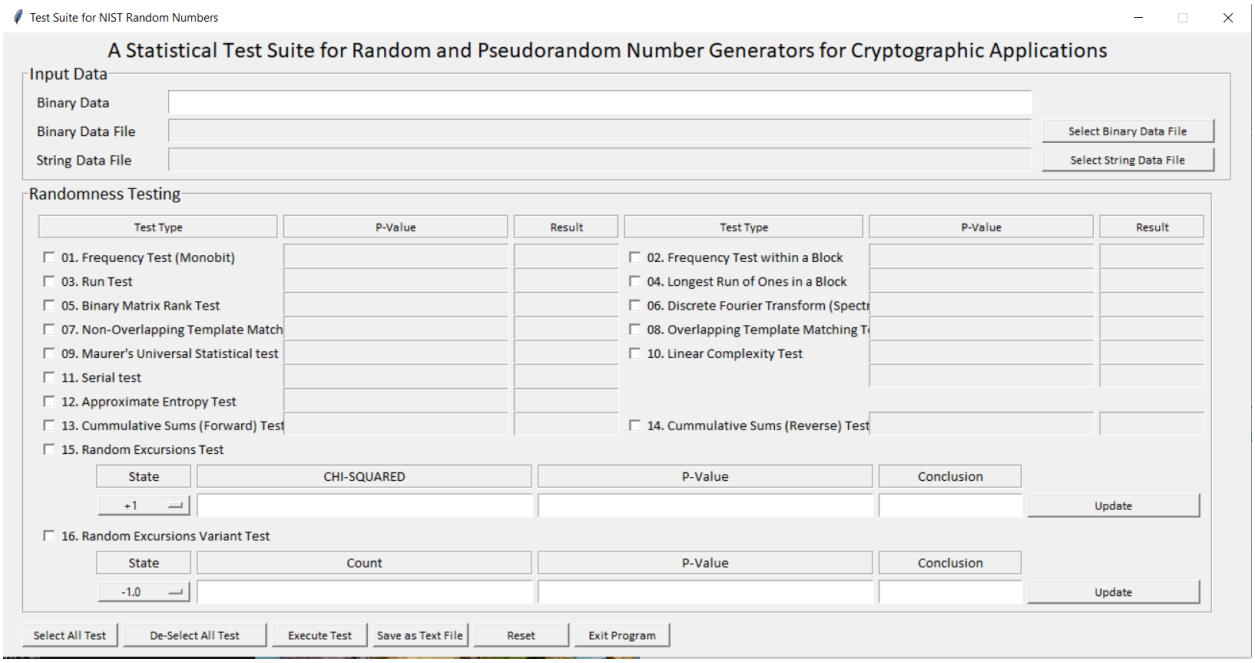


Рис. 3.2 Додаток тестів NIST

Автор додав можливість задати послідовність в додатку різними способами:

1. Написати свою послідовність безпосередньо в додатку;
2. Вибрати бінарний файл в файловому менеджері;
3. Вибрати текстовий файл в файловому менеджері;

Результати перевірок тестів NIST виходить в двох значеннях:

1. P-значення;
2. Чи є послідовність випадкова на основі P-значення;

Треба ще замітити, що у додатку не 15, а 16 тестів, так як з тестом, який пов’язаний з кумулятивними сумами розділений на два тести. В першому кумулятивні суми рахуються, починаючи з першого елемента послідовності. Другий рахує кумулятивні суми в зворотному порядку, з останнього елемента послідовності до першого.

Для прикладу в проекті є перевірка статичними тестами NIST число π, яке представлене у вигляді двійкової послідовності.

```
2.1. Frequency Test:                                (0.5782108547724232, True)
2.2. Block Frequency Test:                          (0.3806151975768745, True)
2.3. Run Test:                                      (0.41926842044315493, True)
2.4. Run Test (Longest Run of Ones):                (0.024389698533896585, True)
2.5. Binary Matrix Rank Test:                       (0.08355314298526481, True)
2.6. Discrete Fourier Transform (Spectral) Test:    (0.010185826152692532, True)
Non-Overlapping Template Test DEBUG BEGIN:
    Length of input:         1000000
    Value of Mean (μ):       244.125
    Value of Variance(σ):    236.03439331054688
    Value of W:              [277. 238. 217. 230. 266. 234. 239. 254.]
    Value of xObs:           11.686326561616164
    P-Value:                 0.1657574574552243
DEBUG END.
2.7. Non-overlapping Template Matching Test:        (0.1657574574552243, True)
2.8. Overlappong Template Matching Test:            (0.2968971234000873, True)
2.9. Universal Statistical Test:                    (0.669012438062546, True)
2.10. Linear Complexity Test:                       (0.2554745740659604, True)
2.11. Serial Test:                                  ((0.1430052395815389, True), (0.034353591982495095, True))
2.12. Approximate Entropy Test:                     (0.3615949310757841, True)
2.13. Cumulative Sums (Forward):                    (0.6283080853765901, True)
2.13. Cumulative Sums (Backward):                   (0.6633686090204551, True)
2.14. Random Excursion Test:
```

Рис. 3.3 Тести NIST числа π

```
2.14. Random Excursion Test:
        STATE           xObs                P-Value             Conclusion
        '-4'        -4          6.287557990511665       True
        '-3'        -3          3.3942951156812353          True
        '-2'        -2          6.409025992573551       True
        '-1'        -1          3.568123393316195       True
        '+1'        1       2.0359897172236505          True
        '+2'        2       2.379383668158304       True
        '+3'        3       2.405297686375321       True
        '+4'        4       3.474312866639757       True
2.15. Random Excursion Variant Test:
        STATE       COUNTS              P-Value         Conclusion
        '-9.0'          -9.0        779                     True
        '-8.0'          -8.0        764                     True
        '-7.0'          -7.0        804                     True
        '-6.0'          -6.0        836                     True
        '-5.0'          -5.0        814                     True
        '-4.0'          -4.0        820                     True
        '-3.0'          -3.0        793                     True
        '-2.0'          -2.0        747                     True
        '-1.0'          -1.0        766                     True
        '+1.0'          1.0         752                     True
        '+2.0'          2.0         753                     True
        '+3.0'          3.0         783                     True
        '+4.0'          4.0         817                     True
        '+5.0'          5.0         807                     True
        '+6.0'          6.0         769                     True
        '+7.0'          7.0         766                     True
        '+8.0'          8.0         761                     True
        '+9.0'          9.0         778                     True
```

Рис. 3.4 Тести NIST числа π

В подальшому на цій програмі буде проведено аналіз різніх ГВЧ для порівняльної характеристики, щоб визначити найкращий ГВЧ в якості шифрування.

3.4. Генератори випадкових чисел на Python та їх порівняльна характеристика

В цьому розділу проводиться дослідження як створюється випадкова двійкова послідовності на бібліотеках мови Python та який з генераторів є кращим для подальшого використання.

Слід перелічити такі бібліотеки як:

- random
- secrets

Перш ніж створювати число за допомогою ГВЧ, слід зазначити, що згенероване число матиме довжину 10^6.

Почнемо з бібліотеки random. Тут є два варіанти створення такими функціями як:

1. “random.choices()”.

2. “randrange()”, який знаходиться в класі “SystemRandom”.

Розглянемо “random.choices()”:

```
import random

binary = ['1', '0']
results = random.choices(binary, k=1000000)
```

Рис. 3.5 Функція “random.choices()”

Ця функція кожною ітерацією випадково обирає число зі списку binary та ставить його наступним числом в змінну results. В параметрах функції є два аргументи, перший описує на основі яких чисел виконується випадкова розстановка чисел, другий описує скільки таких розстановок буде.

Розглянемо “randrange()”:

```
from random import SystemRandom
cryptogen = SystemRandom()
r_numbs = [cryptogen.randrange(2) for i in range(10 ** 6)]
```

Рис. 3.6 Функція randrange()

Використано клас “SystemRandom” з бібліотеки “random”. Різниця між цієї функцію і попередньою “random.choices()” тим, що цей метод використовує ресурси операційної системи.

Спочатку імпортуються клас “SystemRandom”, в якому використовується метод цього класу “randrange()”.Всередині цього метода записується число, яке починаючи з 0 потрібно генерувати. В моєму випадку це число 2. Далі за допомогою циклу я вказує довжину потрібної мені послідовності і все це записується в змінну r_numbs.

Далі бібліотека secrets, яка визнає себе свій генератор достатньо крипто стійким завдяки тому, що вона використовує методи синхронізації, при якому два процеси не можуть отримати одночасно ті ж самі дані.

Так само як і в попередній функції, інструкції по генерації випадкових чисел такі ж самі.

```
import secrets
instance = secrets.SystemRandom()
key = [instance.randrange(2) for i in range(10 ** 6)]
```

Рис. 3.7 Функція randrange() в бібліотеці secrets

Маючи три бінарні файли, потрібно провести аналіз щодо створених послідовностей за статичними тестами NIST.

Спочатку перевіримо бібліотеку random с її двома функціями: “random.choices()” та “randrange()”. Результати тестів запишемо в таблицю для подальшого порівняння з іншими ГВЧ.

Таблиця перевірки на тестах NIST функції “random.choices()”:

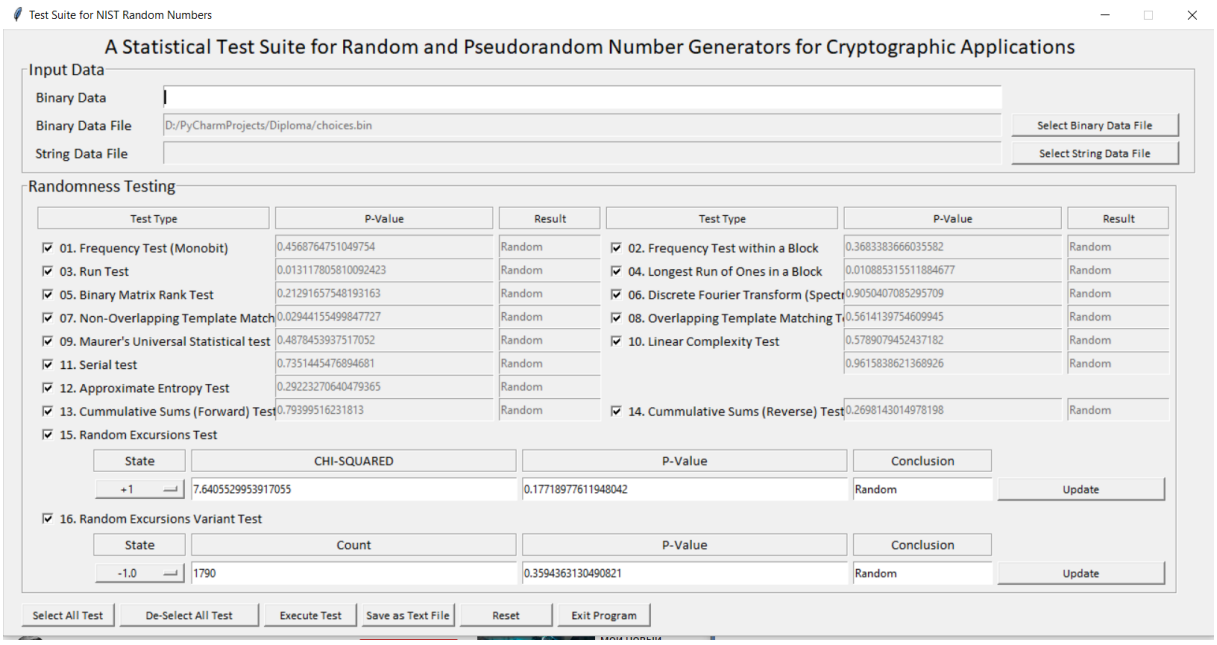


Рис. 3.8. Перевірка функції “random.choices()”

Тести NIST	Випадкова / Не випадкова
Частотний побітовий тест	Випадкова
Частотний блоковий тест	Випадкова
Тест на однакові біти, що йдуть поряд	Випадкова
Тест на найдовшу послідовність з одиниць в блоці	Випадкова
Тест рангів бінарних матриць	Випадкова
Спектральний тест	Випадкова
Тест на відповідність шаблону, що не перекривається	Випадкова
Тест на відповідність шаблону накладання	Випадкова
"Універсальний статистичний" тест Маурера	Випадкова
Тест на лінійну складність	Випадкова
Серійний тест	Випадкова
Приблизний тест на ентропію	Випадкова
Тест кумулятивних сумм(прямим перебором)	Випадкова
Тест кумулятивних сумм(зворотнім перебором)	Випадкова
Тест на випадкові відхилення	Випадкова
Різновидність тесту на випадкове відхилення	Випадкова

Таблиця перевірки на тестах NIST функції “random.randrange()”:

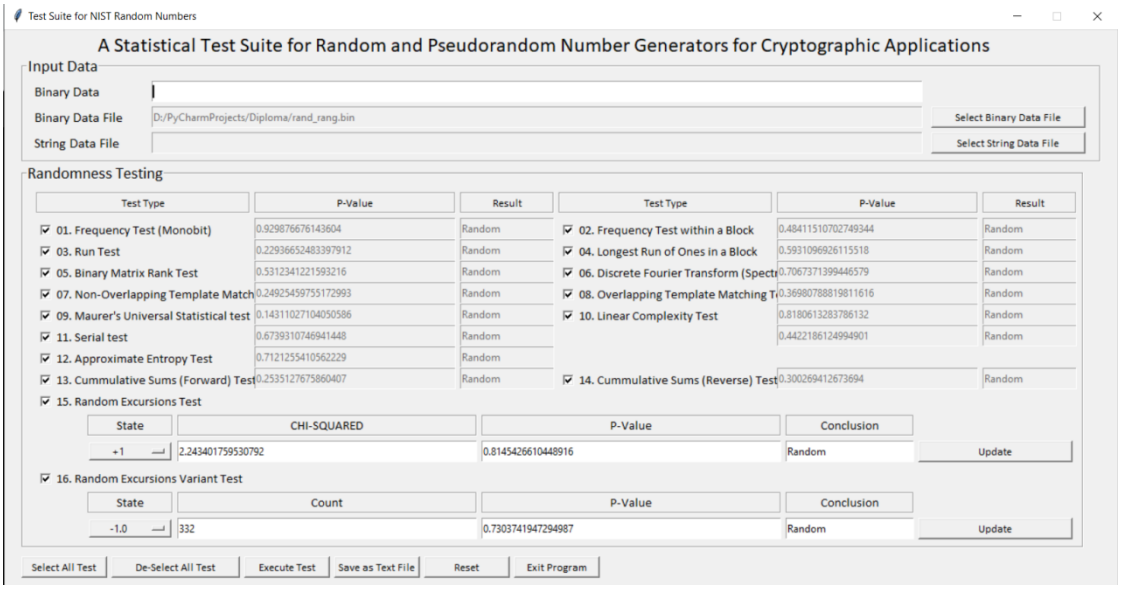


Рис. 3.9. Перевірка функції “randrange()”

Тести NIST	Випадкова / Не випадкова
Частотний побітовий тест	Випадкова
Частотний блоковий тест	Випадкова
Тест на однакові біти, що йдуть поряд	Випадкова
Тест на найдовшу послідовність з одиниць в блоці	Випадкова
Тест рангів бінарних матриць	Випадкова
Спектральний тест	Випадкова
Тест на відповідність шаблону, що не перекривається	Випадкова
Тест на відповідність шаблону накладання	Випадкова
"Універсальний статистичний" тест Маурера	Випадкова
Тест на лінійну складність	Випадкова
Серійний тест	Випадкова
Приблизний тест на ентропію	Випадкова
Тест кумулятивних сумм(прямим перебором)	Випадкова
Тест кумулятивних сумм(зворотнім перебором)	Випадкова
Тест на випадкові відхилення	Випадкова
Різновидність тесту на випадкове відхилення	Випадкова

Перевіряємо бібліотеку secrets на статистичних тестах NIST.

Таблиця перевірки на тестах NIST функції “randrange()”:

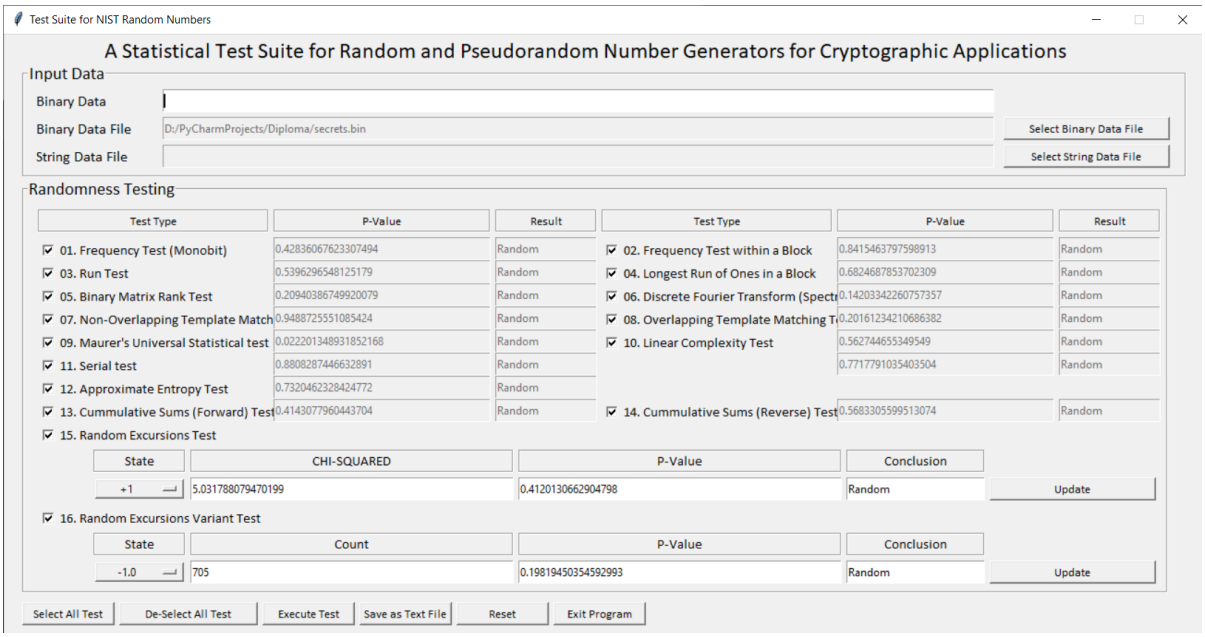


Рис. 3.10. Перевірка функції “randrange()”

Тести NIST	Випадкова / Не випадкова
Частотний побітовий тест	Випадкова
Частотний блоковий тест	Випадкова
Тест на однакові біти, що йдуть поряд	Випадкова
Тест на найдовшу послідовність з одиниць в блоці	Випадкова
Тест рангів бінарних матриць	Випадкова
Спектральний тест	Випадкова
Тест на відповідність шаблону, що не перекривається	Випадкова
Тест на відповідність шаблону накладання	Випадкова
"Універсальний статистичний" тест Маурера	Випадкова
Тест на лінійну складність	Випадкова
Серійний тест	Випадкова
Приблизний тест на ентропію	Випадкова
Тест кумулятивних сумм(прямим перебором)	Випадкова
Тест кумулятивних сумм(зворотнім перебором)	Випадкова
Тест на випадкові відхилення	Випадкова
Різновидність тесту на випадкове відхилення	Випадкова

З розглянутих ГВЧ за тестами NIST, бібліотеки “random” та “secrets” підходять у якості генерації випадкових чисел.

3.5. Вдосконалення алгоритму шифрування з використанням випадкових чисел

Розібравши з бібліотеками генераторів випадкових чисел на Python, алгоритм XOR модифікується у такий формі:

1. Ключ задається за допомогою генератора випадкових чисел у вигляді випадкової двійкової послідовності.

2. Після змінних ключа та тексту, який ми шифруємо, написана функція в подальшому перетворить вихідні числа у вигляді двійкової послідовності.

3. Далі в описаному циклі йде перебір по всій довжині відкритого тексту, і в змінній xor для кожної ітерації береться по одному символу з відкритого тексту та ключа, і до кожного з них застосовується метод “ord”, який повертає число, яке репрезентує цей символ в таблиці Unicode. І вже тоді застосовується операції XOR.

4. В кінці результати змінної xor переносяться cipherBin і вже виводиться результат.

Таким чином для злому такого алгоритму потребується більше ресурсів та часу і надійність алгоритму зросла для використання у шифруванні.

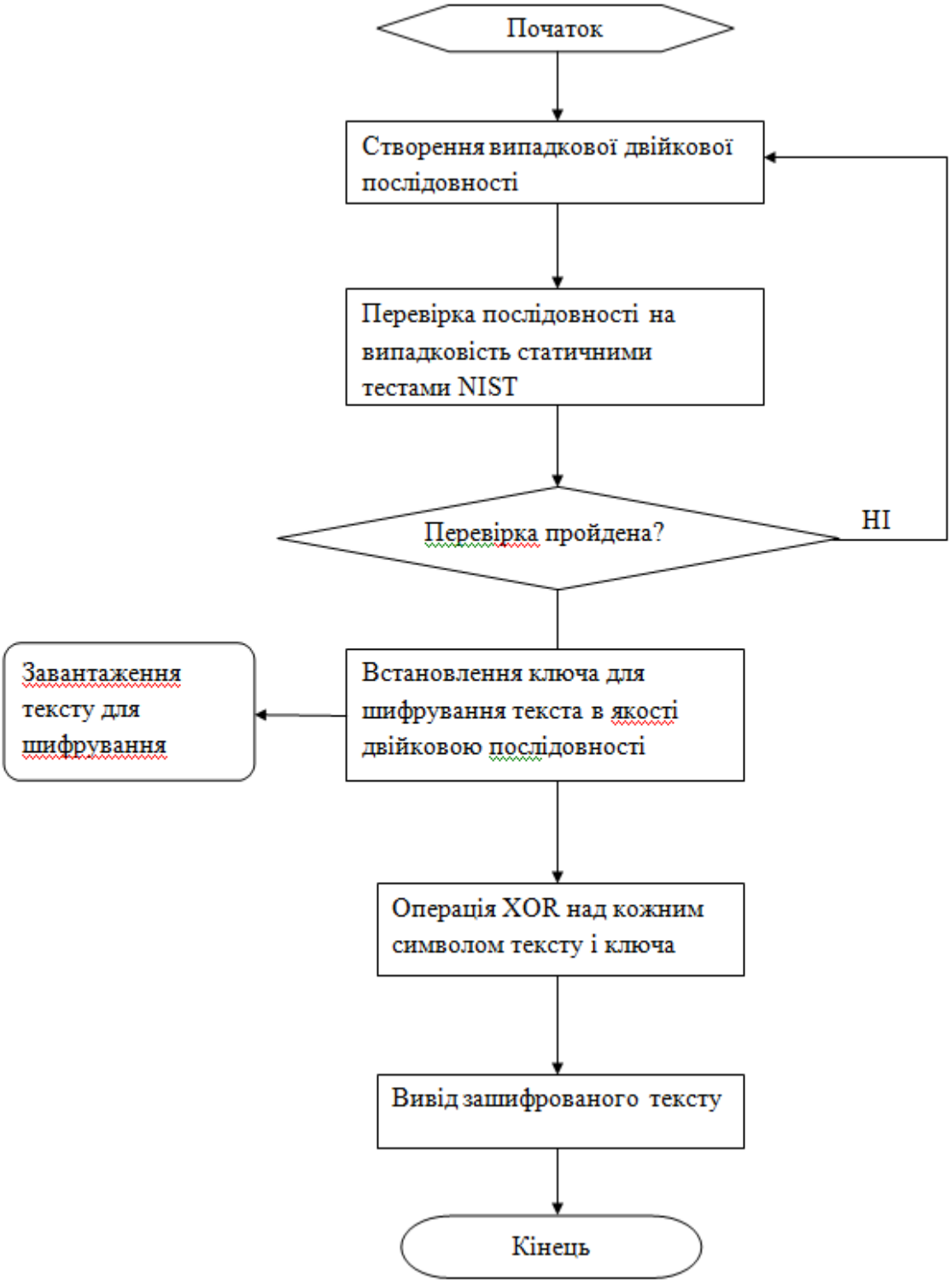
3.6. Демонстрації програмної реалізації

На основі аналізу та досліджень попередніх розділів, далі буде представлено проектування програмної реалізації криптографічного алгоритму з використанням випадкових чисел.

Структура файлів для вхідних та вихідних значень:

1. “key.txt” – випадкова двійкова послідовність, створеною за допомогою генератора випадкових чисел бібліотеки “random” на Python.
2. ”dost.txt” – відкритий текст для шифрування.
3. “cipherText.txt” – зашифрований текст.

Принцип роботи:



Тестування програмного забезпечення:

1. Створення випадкової двійкової послідовності

```
from random import SystemRandom
cryptogen = SystemRandom()
r_numbs = [cryptogen.randrange(2) for i in range(10 ** 6)]
```

Рис. 3.11. Генерації випадкової послідовність

2. Записуємо в файл “key.txt”

```
my_file = open('key.txt', 'w')

for item in r_numbs:
    my_file.write("%s" % item)
print("File r_numbs is written")
my_file.close()
```

Рис. 3.12. Запис файлу

3. Перевіряємо на випадковість статичними тестами NIST

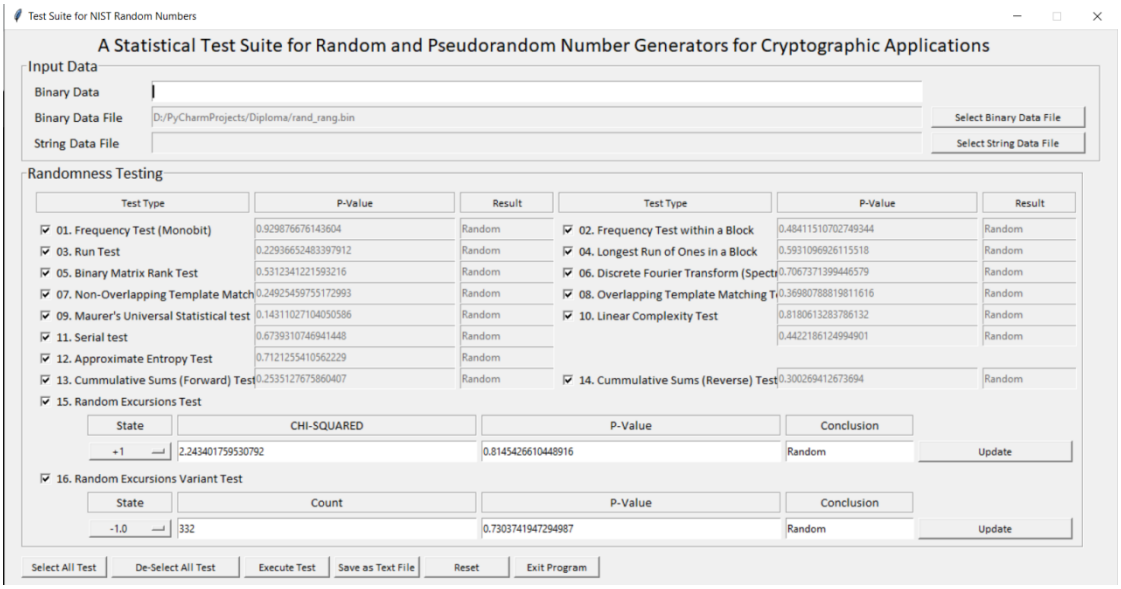


Рис. 3.12. Перевірка на тестах NIST

4. Завантажуємо текст для шифрування в файл “dost.txt”

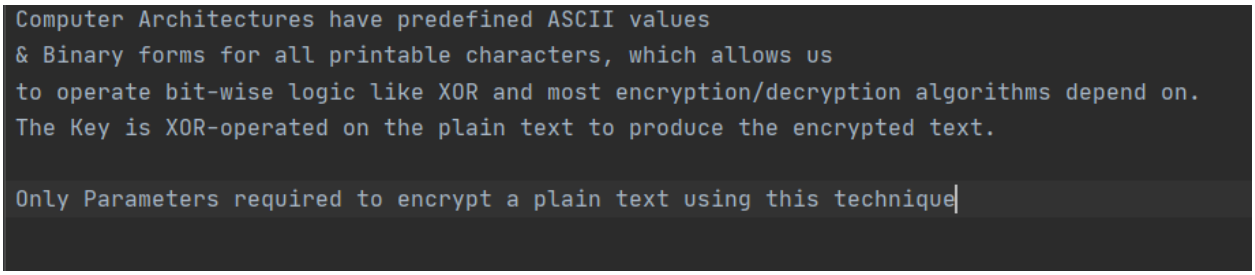


Рис. 3.13. Текст для шифрування

5. Шифруємо текст алгоритмом XOR

```
with open('key.txt', 'r') as f:
    k = f.read().splitlines()
    int_number = []

    for i in k:
        int_number.append(i.strip())

with open("dost.txt") as file:
    text_for_encode = file.read()

def binary(num, length=8):
    b = bin(num).lstrip("0b")
    b = "0" * (length - len(b)) + b
    return b

plaintext = text_for_encode
key = int_number
keyLength = len(key)
cipherBin = ""

for i in range(0, len(plaintext)):
    j = i % keyLength
    xor = ord(plaintext[i]) ^ ord(key[j])
    cipherBin = cipherBin + binary(xor) + " "

print("\nCipher (Binary form): \n" + cipherBin)
```

Рис. 3.14. Алгоритм XOR

6. Вивід результату у вигляді зашифрованого тексту

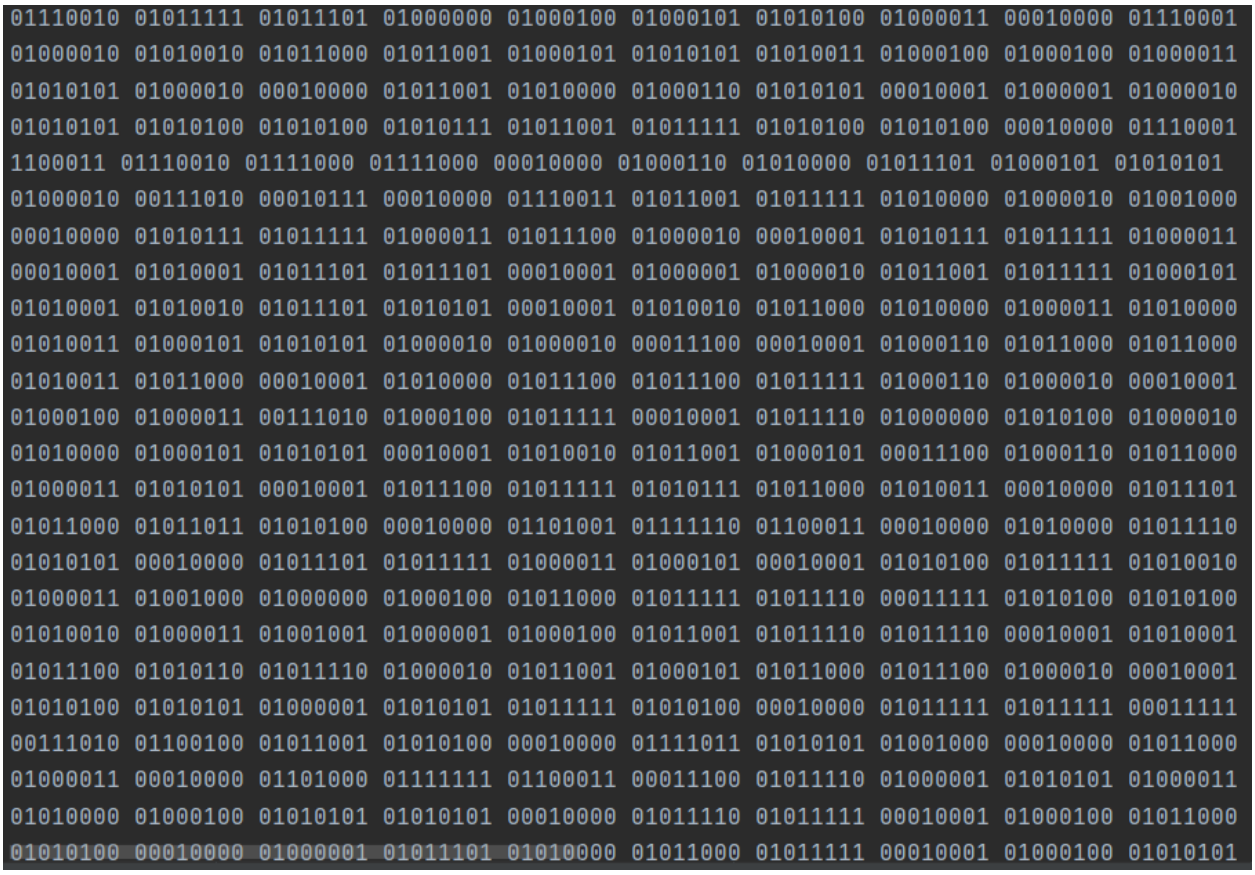


Рис. 3.15. Зашифрований текст

ВИСНОВКИ

Завдання стосовно дипломної роботи виконано. В ході виконання дипломної роботи було проаналізовано літературу стосовно теми розробки та проаналізовані і розібрані терміни криптографічного алгоритму, ГВЧ, лінійно конгруентного методу та алгоритму XOR.

Була реалізована програма для шифрування тексту, яка складалась з таких частин:

- Генерація випадкової послідовності для шифрування тексту у вигляді ключа за допомогою бібліотеки “random” мови програмування Python.
- Алгоритм шифрування XOR, використовуючи вбудовані методи Python.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Яковина В., Одуха О., Сенів М.М., Білас О., Дослідження основних характеристик алгоритму симетричного шифрування RC5 для побудови модуля захисту розподіленої системи теплового проектування // Національний університет “Львівська політехніка”, кафедра програмного забезпечення, 2008 – С. 143-144

[Computer File] URL: http://ena.lp.edu.ua:8080/bitstream/ntb/1329/1/23.pdf

2. Слеповичев И.И., Генераторы псевдослучайных чисел, 2017 - С.17

[Computer File] URL: https://www.sgu.ru/sites/default/files/textdocsfiles/2018/07/09/slepovichev_i.i._generatory_psevdosluchaynyh_chisel_2017.pdf.

3. Block Cipher // Encyclopedia[Електронний ресурс] - Режим доступу до ресурсу: https://www.hypr.com/black-cipher/.

4. Applied Informatics Software Engireering GmbgH., Random Numbers and Cryptographic Hashes

[Computer File] URL: https://pocoproject.org/slides/060-RandomCrypto.pdf

5. Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid , Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo - A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications – 2010.

[Computer File] URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

6. GitHub // steveang[Електронний ресурс] - Режим доступу до ресурсу: https://github.com/stevenang/randomness_testsuite

7. Python // random[Електронний ресурс] - Режим доступу до ресурсу:

https://docs.python.org/3/library/random.html

8. Pynative // python-secrets-module[Електронний ресурс] - Режим доступу до ресурсу:

https://pynative.com/python-secrets-module/