МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_______________ С.В. Казмірчук
«______»_____________ 20__ р.

На правах рукопису
УДК 004.056.5:510.22(043.3)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Удосконалений алгоритм Xorshift для генерації одноразових паролів для двофакторної автентифікації

Виконавець:	В.С. Музика
Науковий керівник: к. т. н., доцент	А. Б. Петренко
Нормоконтролер: к. т. н., доцент	А. Б. Петренко

Київ 2020

ВСТУП

Актуальність. Розвиток інформаційних технологій приводить до розвитку інформаційних систем, що потребують захисту від кіберзлочинів. Одним із методів захисту персональних даних та облікових записів користувачів є системи багатофакторної автентифікації. Серед них найбільшої популярності набула система двофакторної автентифікації з використанням одноразових паролів. Цей метод потребує використання генератора випадкових чисел. Існують різні типи генераторів: табличні, апаратні та програмні, але найбільш зручними у використанні є останні. Для якісної генерації випадкових чисел даним типом треба використовувати алгоритм, що буде водночас швидким, складно передбачуваним, займати невеликий обсяг пам'яті, проходити статистичні тести та бути легким для інтеграції в інформаційні системи.

Метою дипломної роботи є розробка програмної реалізації двофакторної автентифікації за допомогою одноразових паролів з вдосконаленим алгоритмом генерації псевдовипадкових чисел Xorshift.

Об'єкт дослідження: процес генерації випадкових чисел для двофакторної автентифікації одноразовими паролями.

Предмет дослідження: методи та засоби генерації випадкових чисел для двофакторної автентифікації.

Наукова новизна. Вдосконалено алгоритм генерації псевдовипадкових чисел Xorshift за рахунок його ускладнення та модифікації самого процесу генератора, що дозволяє використовувати його для генерації одноразових паролів для двофакторної автентифікації.

Практичне значення роботи полягає у створенні удосконаленого швидкого алгоритму генерації для його використання у програмному модулі двофакторної автентифікації мовою програмування, що найкраще розкриває його якості, — C++.

Розділ 1. ГЕНЕРАТОРИ ВИПАДКОВИХ ЧИСЕЛ

1.1. Типи генераторів випадкових чисел

Давно існувала необхідність використання випадкових чисел у науковій роботі. Спочатку для цієї мети використовували урну з кулями, з якої випадковим чином витягували кулі з цифрами. Пізніше був створений механічний генератор випадкових чисел. З появою електронних схем також з’явилися електронні генератори випадкових чисел. Один з перших таких генераторів, запропонований А.М. Тьюрингом, використовує резистивний генератор шуму для отримання 20 випадкових бітів суматора. Однак генератори випадкових чисел не завжди дають "якісні" результати. Крім того, апаратні генератори випадкових чисел часто виходили з ладу. Через обмежену пам’ять комп’ютера попередньо розраховані «якісні» таблиці випадкових чисел були надзвичайно незручні у використанні.

Генератори випадкових чисел (ГВЧ) за способом отримання чисел діляться на:

- апаратні;
- табличні;
- алгоритмічні.

Табличний генератор як джерело випадкових чисел використовує заздалегідь підготовлені таблиці, які містять перевірені нерелевантні числа і не є генераторами в строгому сенсі. Недоліки цього методу очевидні: використання зовнішніх ресурсів для зберігання чисел, обмежена послідовність та умовність значень. Прикладом табличного методу є книга.

Апаратний генератор (справжньої) випадкової послідовності повинен мати джерело ентропії. Розробити генератор із використанням джерела ентропії, яке виробляє некорельовані та статистично незалежні числа, є досить складним завданням. Крім того, для більшості програм криптографії

такий ГВЧ не повинен бути предметом дослідження та впливу з боку іншої сторони.

Джерело ентропії (джерело випадковості) - це механізм, який може генерувати фізичні випадкові величини для подальшої ініціалізації вектора ГВЧ. Джерело ентропії (апаратний генератор випадкових чисел) та вектор ініціалізації, який створюється для ГВЧ, вони повинні бути захищені та триматися в секреті. Конфіденційність цієї інформації є основною вимогою до безпеки ГВЧ. Якщо криптографічна стійкість ГВЧ не потрібна, джерело ентропії має видавати лише потрібну кількість випадкових чисел.

Джерелами ентропії можуть бути:

- підкидання монети;
- тимчасові затримки між моментами випромінювання частинок в процесі радіоактивного розпаду;
- теплові шуми при роботі напівпровідникового діода або резистора;
- частотні відхилення вільно працює генератора частот;
- фотоефект - випускання електронів речовиною під дією світла;
- звук від мікрофона або відео з підключеною камери;
- стан деяких блоків пам'яті комп'ютера.

Побудова апаратного генератора випадкових чисел - трудомістка задача, при якій вам потрібно вирішити деякі проблеми, такі як встановлення діапазону фіксованих випадкових величин, оцифрування аналогових даних та ізоляція фізичних джерел випадковості від зовнішніх впливів.

Алгоритмічний генератор - це поєднання фізичного генератора та детермінованого алгоритму. Такий генератор використовує обмежений набір даних, отриманий на виході фізичного генератора, для створення довгої послідовності випадкових чисел шляхом перетворення вихідних

чисел. Оскільки цей тип генератора має очевидні переваги перед іншими типами генераторів випадкових чисел, він становить найбільший інтерес.

Порівняно зі створенням апаратного генератора випадкових чисел, розробка алгоритму генератора може бути більш складним завданням. Через ціну апаратного генератора випадкових чисел, в більшості випадків, як джерело ентропії використовуються ресурси комп'ютера, на якому виконується програма генерування випадкових чисел. За відсутності апаратного генератора випадкових чисел ви можете використовувати як джерело ентропії:

1. стан системних годин;
2. час затримок між натисканням клавіш клавіатури або рухами мишки;
3. вміст буферів введення / виведення;
4. значення, одержувані при роботі системи (час завантаження системи, взаємодія і т. П.).

Генератори ГВЧ, засновані на перетворенні системного часу, мають деякі недоліки: алгоритми ГВЧ можуть мати свої функції пов'язані з системним часом, тому числа, що генеруються в цей час, матимуть менше випадковості. Наприклад, програма повинна спочатку виводити ряд випадкових чисел щосекунди.

ГВЧ, що використовує згенеровані користувачем події як джерело ентропії (затримка між натисканнями клавіш та координати руху миші), нерівномірно розподіляються при детальному аналізі, що говорить про те, що деякі числа менш випадкові.

Якісний програмний генератор випадкових бітів повинен використовувати якомога більше різних джерел ентропії. Це зменшить вірогідність зловмисника проаналізувати алгоритми генерації випадкових бітів та підвищить надійність генератора у випадку, якщо одне або кілька

джерел ентропії вийдуть з ладу. Потрібно використовувати деякі спеціальні функції для "змішування" сформованої випадкової послідовності.

Табл. 1.1. Порівняння типів генераторів

Генератор	Джерело ентропії	Випадковість послідовності	Переваги	Недоліки
Табличний	Відсутнє	Відстутня	● Простота; ● Мала ціна; ● Швидкість	● Таблиця може бути втрачена; ● Низкий рівень захищенності; ● Відсутнє джерело ентропії
Апаратний	Фізичне	Повністю випадкова	● Істино випадкові числа	● Складність та ціна встановлення; ● Генерація чисел повільніша за алгоритмічний генератор
Алгоритмічний	Алгоритм	Може бути передбачуваною	● Швидкість; ● Ціна; ● Можливість відтворити послідовність чисел	● У простих алгоритмів є ризик бути передбачуваними

Детермінований алгоритм не може генерувати повністю випадкове число, він може лише наблизити деякі його властивості.

Будь-який генератор псевдовипадкових чисел з обмеженими ресурсами рано чи пізно замкне цикл - почне повторювати ту саму послідовність чисел. Довжина періоду генератора псевдовипадкових чисел залежить від самого генератора, приблизно 2n / 2, де n - розмір внутрішнього стану (у бітах), хоча максимальний період лінійного конгруентного генератора та генератора LFSR становить близько 2n . Якщо послідовність генерації генератора псевдовипадкових чисел скорочується до занадто короткого періоду, генератор псевдовипадкових чисел стає передбачуваним і непридатним для практичних застосувань.

Хоча найпростіший арифметичний генератор дуже швидкий, він має багато серйозних недоліків:

- Занадто короткий період / періоди.
- Послідовні значення не є незалежними.
- Деякі біти «менш випадкові», ніж інші.
- Нерівномірний одномірний розподіл.
- Зворотність.

Зокрема, алгоритм RANDU, який використовується десятиліттями, виявився дуже поганим, що робить сумнівними надійність багатьох результатів досліджень із використанням цього алгоритму.

Найбільш поширеними є: лінійний конгруентний метод, метод Фібоначчі із затримкою, регістр зсуву з лінійним зворотним зв’язком та регістр зсуву із узагальненим зворотним зв’язком.

Серед сучасних генераторів псевдовипадкових чисел популярним став також «вихор Мерсена», запропонований Мацумото та Нішимурою в 1997 році. Його перевагами є: величезний період (2^19937-1), рівномірний розподіл в 623 вимірах (лінійний послідовний метод забезпечує більш-менш рівномірний розподіл максимум у 5 вимірах), швидке формування випадкових чисел (порівняно зі стандартним генератором, що використовує

лінійний конгруентний метод, в 2-3 рази швидший). Однак є деякі алгоритми, які можуть визнати послідовність, породжену вихором Мейсона, невипадковою.

Разом з існуючою необхідністю генерувати легко відтворювані послідовності випадкових чисел, існує також необхідність генерувати абсолютно непередбачувані або повністю випадкові числа. Цей вид генератора називається генератором випадкових чисел (ГВЧ - англ. Random number generator, RNG). Оскільки такі генератори найчастіше використовуються для створення унікальних симетричних та асиметричних ключів для шифрування, вони, як правило, складаються з комбінації криптостійких генераторів псевдовипадкових чисел та зовнішніх джерел ентропії (тепер розумійте цю комбінацію як генератор випадкових чисел).

Майже всі великі виробники мікрочіпів забезпечують апаратні генератори випадкових чисел різними джерелами ентропії, і вони використовують різні методи для усунення неминучої передбачуваності. Однак швидкість, з якою всі існуючі мікрочіпи збирають випадкові числа (тисячі біт в секунду), не відповідає швидкості сучасних процесорів.

Сучасні дослідження намагаються використовувати результати вимірювання фізичних властивостей об’єктів (таких як температура) і навіть квантові коливанння вакууму як джерела ентропії для генераторів випадкових чисел.

На персональних комп'ютерах автори програмних генераторів випадкових чисел використовують більш швидкі джерела ентропії, такі як шум звукової карти або лічильник тактів процесора. Збір ентропії є найбільш вразливою частиною генераторів випадкових чисел. На багатьох пристроях (наприклад, на смарт-картах) ця проблема досі не вирішена повністю, тому вона все ще вразлива до атак. Багато генераторів випадкових чисел використовують традиційні, хоча і повільні, методи збору ентропії, такі як вимірювання реакції користувача (руху миші тощо)

у PGP та Yarrow, або взаємодія між потоками як наприклад у JavaSecureRandom.

Відповідно до основних використовуваних арифметичних операцій ГВЧ для генерації рівномірного розподілу випадкових чисел можна класифікувати на дві групи: ГВЧ на основі модульної арифметики та ГВЧ на основі двійкової арифметики.

1.1.1. ГВЧ на основі арифметики за модулем

Цей тип ГВЧ дає послідовність випадкових чисел за допомогою лінійного повторення за модулем m, де m - велике ціле число.

1.1.1.1. Лінійний конгруентний метод (ЛКМ)

ЛКМ - один з найвідоміших генераторів випадкових чисел. ЛКМ визначається за такою формулою:

$$x_i = a \cdot x_{i-1} + c \bmod m$$

де x - послідовність згенерованих випадкових чисел і $m > 0$, $0 < a < m$, $0 \leq c, x_0 < m$. Якщо розподіл потрібен у вигляді [0, 1), тоді використовуйте $u = \frac{x}{m}$ як вихідну послідовність.

Для ЛКМ слід ретельно вибирати a, c та m, щоб переконатися, що максимальний період можна архівувати. ЛКМ може бути легко реалізований на комп'ютерному обладнанні, яке може забезпечити модульну арифметику шляхом усікання бітів. ГВЧ з використанням ЛКМ постачається з бібліотекою C (rand ()), а також багатьма іншими мовами, такими як Java (java.lang.Random). ЛКМ має відносно короткий період (щонайбільше 2^32 для 32-розрядних цілих чисел) порівняно з іншими більш складними.

Особливою умовою ЛКМ є, коли c = 0, що представляє клас мультиплікативних конгруентних методів (МКМ). Кілька ретельно

відібраних МКМ можна об'єднати в більш складні алгоритми, такі як генератор Whichmann-Hill.

1.1.1.2. Багаторазовий рекурсивний генератор (БРГ)

БРГ є похідним ЛКМ і може досягти набагато більш тривалого періоду. БРГ порядку k визначається таким чином:

$x_i = (a_1 \cdot x_{i-1} + a_2 \cdot x_{i-2} + \cdot\cdot\cdot + a_k \cdot x_{i-k}) \bmod m$

Рекурсія має максимальну довжину періоду m^k-1.

1.1.1.3. Комбінований багаторазовий рекурсивний генератор (КБРГ)

КБРГ поєднує кілька БРГ і може отримати кращі статистичні властивості та триваліші періоди порівняно з одним БРГ. Добре відома реалізація КБРГ, CMR32k3a [12], поєднує два БРГ:

$$xi = a11 \cdot xi-1 + a12 \cdot xi-2 + x13 \cdot xi-3 \bmod m1 \quad yi = a21 \cdot yi-1 + a22 \cdot yi-2 + x23 \cdot yi-3 \bmod m2 \quad z_i = x_i,$$

де z утворює необхідну послідовність.

1.1.2. Бінарні ГВЧ на основі арифметики

ГВЧ цього типу визначаються безпосередньо з точки зору бітових рядків та послідовностей. Оскільки комп’ютери швидко виконують двійкові арифметичні операції, ГВЧ на основі двійкової арифметики можуть бути ефективнішими, ніж орієнтовані на модульну арифметику.

1.1.2.1. Xorshift

Xorshift генерує випадкові числа за допомогою багаторазового використання побітових операторів виключне-або (xor, $\oplus$) та бітових зсувів ($\ll$ для лівого та $\gg$ для правого).

Xorshift з чотирма вхідними даними (x, y, z, w) може бути реалізований наступним чином:

$$t = (x \oplus (x_i \ll a))$$

$$x = y$$

$$y = z$$

$$z = w$$

$w = (w \oplus (w \gg b)) \oplus (t \oplus (t \gg c))$, де w утворює необхідну послідовність. За допомогою ретельно підібраного параметрів (a, b, c) сформована послідовність може мати період до $2^{128} - 1$.

1.1.2.2. Mersenne Twister (MT)

MT - один із найбільш поважаємих ГВЧ, це витковий регістр зсуву з узагальненою віддачею. За належних значень параметрів MT може генерувати послідовність із періодом до 2^19937-1 та надзвичайно якісними статистичними властивостями.

1.2.Внутрішні функції ГПВЧ

Внутрішні функції керують внутрішнім станом і реалізують ключові алгоритми ГПСЧ. У NIST SP 800-90A рекомендують використовувати п'ять функцій.

1.2.1. Функція ініціалізації

Функція ініціалізації - отримує число від джерела ентропії, поєднує його з рядком персоналізації, поточним часом, щоб створити вектор ініціалізації ГПВЧ.

ГПВЧ потрібно ініціалізувати перед початком генерації псевдовипадкових чисел. Функція ініціалізації робить наступне:

1. перевіряє коректність вхідних параметрів;
2. визначає рівень криптостійкості;
3. визначає специфічні параметри ГПВЧ (наприклад, набір параметрів еліптичної кривої);
4. отримує значення від джерела ентропії з достатнім рівнем надійності,
5. отримує код сеансу (nonce);
6. визначає початковий внутрішній стан, використовуючи алгоритм ініціалізації.

1.2.2. Функція встановлення початкового вектора (реініціалізації)

Функція встановлення початкового вектора отримує нове значення від джерела ентропії, поєднує його з поточним внутрішнім станом та додатковими вхідними значеннями та генерує новий початковий вектор та новий внутрішній стан для наступного запиту до ГПВЧ. Повторна ініціалізація додає випадковість у процесі генерації псевдовипадкових чисел, які можуть бути:

- запрошеною додатком споживача ПВЧ;
- виконана відповідно до правил захисту від передбачення згенерованих чисел;
- виконана після певного числа згенерованих чисел;
- виконана по зовнішній події (наприклад, в момент доступності джерела ентропії).

У деяких ГПВЧ немає окремої функції реініціалізації. Замість цього використовується функція ініціалізації.

1.2.3. Функція реініціалізації

1. Перевіряє правильність вхідних параметрів.
2. Отримує істинно випадкове число від джерела ентропії.
3. Використовуючи алгоритм реініціалізації, комбінує поточний внутрішній стан з новим істинно випадковим числом і додатковим вхідним значенням для визначення нового внутрішнього стану.

1.2.4. Функція генерації значення

Функція генерації значення - створює псевдовипадкове число за запитом, використовуючи поточний внутрішній стан і змінюючи внутрішній стан для наступного запиту. Ця функція виконує наступні дії:

1. Перевіряє правильність вхідних параметрів.
2. Викликає функцію реініціалізації, якщо це необхідно
3. Генерує число, використовуючи алгоритм генерації ПВЧ.
4. Обчислює новий внутрішній стан.

1.2.5. Функція деініціалізації

Функція деініціалізації очищає значення внутрішнього стану. Це необхідно робити в цілях протидії аналізу системи.

1.2.6. Функція, що тестує ГПВЧ на коректність

Функція, що тестує ГПСЧ на коректність проводить тест на коректність роботи ГПВЧ і повідомляє про результат додатком-споживачеві ГПСЧ.

1.3. Алгоритми генераторів псевдовипадкових чилел

1.3.1. Mersenne Twister

Цей алгоритм є одним із найбільш широко використовуваних і рекомендованих ГВЧ. Він існує в C ++ 11 як mt19937 та mt19937_64, а також є генератором випадкових чисел за замовчуванням для Python.

Позитивні якості

- Виробляє 32-розрядні або 64-розрядні числа (таким чином, його можна використовувати як джерело випадкових бітів)
- Проходить більшість статистичних тестів

Нейтральні якості

- Надмірно величезний період 2^19937 - 1
- 623 виміри рівного розподілу
- Період можна розділити для емуляції декількох потоків

Негативні якості

- Не проходить деякі статистичні тести, налічуючи лише 45 000 чисел.
- Передбачуваний - після 624 вихідних даних ми можемо повністю передбачити його вихід.
- Генератор займає 2504 байт оперативної пам'яті - на відміну від цього, придатний для використання генератор з великим періодом, займає 8 байт оперативної пам'яті.
- Не особливо швидкий.
- Не особливо ефективний. Генератор використовує 20000 біт для зберігання свого внутрішнього стану (20032 біта на 64-розрядних машинах), але має період лише 2^19937, що в 263 (або 295) разів менше, ніж ідеальний генератор такого ж розміру.
- Нерівномірність вихідних значень; генератор може потрапити в “поганий стан”, з якого повільно відновлюється.

- Вхідні дані, які лише незначно відрізняються, довго розходяться між собою; їх значення потрібно обирати обережно, щоб уникнути поганих станів.

1.3.2. Minstd та Unix rand

Minstd і rand (деякі реалізації) є прикладами лінійних конгруентних генераторів з сумою по модулю 2.

Позитивні якості

- Використовує дуже малий обсяг пам'яті

Негативні якості

- Короткий період
- Передбачуваний - отримавши лише одне випадкове число, ми можемо повністю передбачити його результат.
- Не особливо якісний для створення 32-розрядних чисел (або потоку випадкових бітів)
- Виробляє кожне число лише один раз
- Не проходить багато статистичних тестів
- Minstd досить повільний (тоді як швидкість rand залежить від реалізації)
- Типові реалізації не забезпечують декілька потоків, хоча це можна було б зробити
- Типові реалізації не забезпечують швидкого просування, хоча це можна було б зробити

1.3.3. Arc4Random

Цей ГВЧ існує в системах OS X та BSD як arc4random, який базується на комерційному секреті алгоритму.

Позитивні якості

- Криптографічно захищений (не передбачуваний, широко використовується як потоковий шифр).
- Проходить емпіричні статистичні тести TestU01
- Виробляє 32-розрядні числа (таким чином, можна використовувати потік випадкових бітів)

Нейтральні якості

- Занадто величезний очікуваний період, приблизно 2^1699
- Різні ініціалізації можуть генерувати різні випадкові потоки

Негативні якості

- Статистично посередній
- Хоча стандартний варіант проходить тест BigCrush TestU01, це не є великим досягненням, якщо врахувати 2064 біти внутрішнього стану - простий ЛКМ проходить з 88 бітами внутрішнього стану.
- Математично було доведено, що алгоритм є неоднорідним. (Насправді, незважаючи на те, що тест не входить до набору TestU01, існують тести, які дозволяють відрізнити вихід arc4random від справжньої випадкової послідовності.)
- На відміну від деяких генераторів з дуже великим періодом, не забезпечує k-вимірного рівномірного розподілу.
- Період варіюється залежно від вхідних даних.
- Дуже повільний.
- Реалізації BSD (які є єдиними широко використовуваними) мають кілька додаткових проблем
- Періодично "перемішує" генератор, використовуючи ентропію, надану ядром; цей код повинен бути видалений, якщо є потреба у відтворювані результатів
- Неодноразово “виправлявся” для усунення помилок та проблем безпеки.

1.3.4. ChaCha20

Цей алгоритм надається в системах OpenBSD як заміна arc4random. Він заснований на потоковому шифрі ChaCha20 Даніеля Дж. Бернштейна, який є варіантом його шифру Salsa20.

Позитивні якості

- Криптографічно захищений (не передбачуваний, був перевірений криптографічним співтовариством).
- Підтримує різні варіанти (наприклад, ChaCha8), які намагаються бути менш захищеними, але працюють швидше, зменшуючи кількість раундів.
- Проходить емпіричні статистичні тести TestU01
- Виробляє 32-розрядні числа (таким чином, можна використовувати потік випадкових бітів)
- Стверджується, що потоковий шифр ChaCha20 приблизно в 3 рази швидший за потоковий шифр Arc4.
- Потіковий шифр ChaCha20 є доступним, що дозволяє перейти вперед (але ця функція не передбачена реалізацією OpenBSD).
- Приблизно на 70% швидше, ніж arc4random, якщо він використовується як загальний генератор 32-розрядних випадкових чисел
- У випадку з OpenBSD їх реалізація arc4random "перемішується" (тобто вимагає зовнішньої ентропії від ядра) набагато частіше, ніж їх реалізація ChaCha20, що призводить до прискорення роботи у 4,5 рази на цій платформі.

Нейтральні якості

- Призначений для використання як потоковий шифр, тим самим створює блоки випадкових чисел. Незважаючи на те, що підхід "блокування часу" може дозволити певні оптимізації, він може

витратити місце, коли є типовим загальнодоступним ГВЧ загального призначення.

Негативні якості

- Повільний порівняно із генераторами загального призначення (хоча він на 70% швидший, ніж arc4random, він на порядок повільніший, ніж високопродуктивні загальноприйняті ГВЧ).
- Реалізація OpenBSD використовує 1104 байта для зберігання стану генератора (приблизно в 4 рази більше пам’яті ніж у arc4random, оскільки він одночасно обчислює шістнадцять 64-байтових блоків). Більш позитивним є той факт, що реалізація C ++ від Орсона Пітерса вимагає лише 104 байти, оскільки вона зберігає лише один 64-байтовий блок.
- Має значно менший період, ніж можна було б зробити з розміру стану генератора.
- Менша кількість раундів призводить до поганих статистичних показників; ChaCha2 сильно провалює статистичні тести, ChaCha4 проходить TestU01, але складний математичний аналіз показав, що він демонструє деяке зміщення. ChaCha8 вважається вже якісним. Тим не менш, ChaCha потрібно докласти більше зусиль для досягнення задовільної статистичної якості, ніж більшість інших генераторів.
- На відміну від деяких генераторів з дуже великим періодом, він не забезпечує k-вимірного рівномірного розподілу.
- На відміну від реалізації C ++ Орсоном Пітерсом, реалізація OpenBSD (яку досить часто пропонують як “якісну” реалізацію) має кілька додаткових проблем:
    - Періодично “перемішує” генератор, використовуючи ентропію, надану ядром; цей код потрібно видалити, якщо бажано відтворювати результати

- Надається як єдиний глобальний ГВЧ.
- Багатопотокові програми повинні спільно використовувати глобальний ГВЧ і боротися за його блокування.

1.3.5. Unix drand48, java.util.Random Java

Ці генератори є прикладами лінійних конгруентних генераторів з сумою по модулю 2.

Позитивні якості

- Швидкий (якщо операція множення працює швидко)
- Використовує невеликий обсяг пам'яті
- Можливий ефективний стрибок уперед
- Добре працює для створення 32-розрядних чисел (або потоку випадкових бітів)

Негативні якості

- Досить короткий період
- Передбачуваний (хоча генерація 48 бітів і зниження низьких 16 бітів робить прогнозування трохи важчим, ніж rand)
- Не проходить більшість статистичних тестів
- Типові реалізації не забезпечують декілька потоків, хоча це можна було б зробити
- Типові реалізації не забезпечують швидкого просування, хоча це можна було б зробити

1.3.6. Unix random

Unix random - це приклад ГВЧ із лінійним зворотнім зсувом регістру.

Позитивні якості

- Швидкий

- Добре працює для створення 32-розрядних чисел (або потоку випадкових бітів)

Негативні якості

- Досить короткий період
- Передбачуваний - після отримання 16 випадкових чисел ми можемо повністю передбачити його результат
- Не проходить більшість статистичних тестів
- Типові реалізації не забезпечують швидкого просування, хоча це було б можливо зробити (проте реалізація була б досить складною)
- Технічно не рівномірний - нуль буде видаватися раз рідше, ніж будь-який інший результат

1.3.7. XorShift (32-розрядна та 64-розрядна)

Ці варіанти генератора XorShift, різновид узагальненого лінійного зворотного зсуву регістру.

Позитивні якості

- Швидкий (навіть якщо операція множення не є швидкою)
- Використовує невеликий обсяг пам'яті
- Добре працює для створення 32-розрядних чисел (або потоку випадкових бітів)

Негативні якості

- Досить короткий період (лише 32-розрядна версія)
- Передбачуваний - виробивши лише одне випадкове число, ми можемо повністю передбачити його вихідні дані.
- Не проходить багато статистичних тестів
- Технічно не рівномірний - нуль буде видаватися рідше, ніж будь-який інший результат

- 32-розрядний XorShift, як правило, не слід використовувати для створення 32-розрядних чисел, оскільки він створює кожне число лише один раз і ніколи не виробляє нуль.
- Не забезпечує кілька потоків
- Типові реалізації не забезпечують швидкого просування, хоча це було б можливо зробити (проте реалізація була б досить складною)

1.3.8. RanQ і XorShift * 64/32

Ці генератори використовують узагальнений лінійний зворотній зсув регістру у поєднанні з мультиплікативним кроком для вихідної функції. RanQ і XorShift * 64/32 - це один і той самий генератор із 64 бітами стану, але XorShift * 64/32 повертає лише 32 біти вихідних даних, тоді як RanQ видає всі 64 біти.

Позитивні якості

- Швидкий (якщо операція множення швидка)
- Використовує невеликий обсяг пам'яті
- Добре працює для створення 32-розрядних чисел (або потоку випадкових бітів)
- Легко проходить емпіричні статистичні тести (лише XorShift * 64/32)

Нейтральні якості

- Періоду в 2^64 достатньо для багатьох застосувань, але в деяких випадках це буде вважатись недостатнім

Негативні якості

- Технічно не рівномірний - нуль буде видаватися рідше, ніж будь-який інший результат
- Передбачуваний - отримавши лише одне випадкове число, ми можемо повністю передбачити його вихід (лише RanQ).

- RanQ заявляє, що є 64-розрядним генератором, але його 32 біти нижчого порядку не проходять статистичні тести.
- Не забезпечує кілька потоків
- Типові реалізації не забезпечують швидкого просування, хоча це було б можливо зробити (проте реалізація була б досить складною)

Висновок до розділу 1.

З розвитком інформаційних технологій інформаційна система потребує більшого захисту, ніж просто логін і пароль. На вихід цій ситуації прийшла багатофакторна автентифікація і її найбільш популярний метод – одноразові паролі. Даний вид багатофакторної автентифікації потребує надійного та швидкого генератора випадкових чисел, що дозволить користувачам отримувати доступ до своїх облікових записів та персональних даних. В даному розділу ми розглянули методи генерації випадкових чисел та типи генераторів. Було визначено основні переваги та недоліки табличних, апаратних та алгоритмічних генераторів, що дало змогу обрати такий тип, що задовільнить сучасні потреби у найкращий спосіб, – програмний. Серед алгоритмічних генераторів було обрано алгоритм Xorshift через його швидкість та простоту інтеграції в інформаційну систему для його подальшого вдосконалення для використання у двофакторній автентифікації одноразовими паролями. А вхідними даними для нього будуть виступати апаратні параметри комп’ютера, на якому здійснюється робота генератора Xorshift.

Розділ 2. ВДОСКОНАЛЕННЯ АЛГОРИТМУ XORSHIFT ТА ЙОГО ПОРІВНЯННЯ З ІНШИМИ АЛГОРИТМАМИ

Генератори випадкових чисел Xorshift (їх також називають генераторами регістрів зсуву) - це тип генератора псевдовипадкових чисел, відкритий Джорджем Марсальєю. Вони є підмножиною генераторів лінійного регістра зсуву зворотного зв'язку (LFSR). Вони згенерують наступний число у послідовності, неодноразово беручи ексклюзивне-або з числа із бітовим зсувом самого себе. Це робить їх дуже швидкими в сучасних комп'ютерних архітектурах. Як і всі LFSR, параметри потрібно вибирати дуже ретельно, щоб досягти більш тривалого періоду.

Генератор Xorshift є одним з найшвидших генераторів випадкових чисел і вимагають дуже малого розміру коду та стану. Хоча вони не проходять всі статистичні тести без вдосконалення, ця слабкість добре відома і легко виправляється (як зазначив Марсалья в оригінальній роботі) поєднанням їх з нелінійними функціями.

Генератор псевдовипадкових чисел xorshift (xorshift RNG) виробляє послідовність 2^32-1 цілих чисел, або послідовність 2^64-1 при 2 вхідних даних x, y, або послідовність 2^96-1 при 3 вхідних даних x, y, z за допомогою багаторазового використання простої комп’ютерної конструкції: виключне - або (xor) вхідних даних зі зміщеною версією самих себе. У C основна операція – y^(y << a) для зсувів ліворуч, yˆ (y >> a) - для зрушень праворуч. Поєднання таких операцій xorshift для різних зрушень та аргументів забезпечує надзвичайно швидкі та прості RNG, які добре справляються з тестами на випадковість. Щоб дати уявлення про потужність та ефективність операцій xorshift, ось основна частина процедури C, яка, маючи лише три операції xorshift за виклик, забезпечить 2^128−1 випадкових 32-розрядних цілих чисел, з урахуванням чотирьох випадкових насінин x, y , z, w:

tmp = (x^ (x << 15));

x = y;

y = z;

z = w;

вихідні дані: w = (w^ (w >> 21)) ^ (tmp^ (tmp >> 4));

Така процедура є дуже швидкою, як правило, перевищує 200 мільйонів чисел/секунду, і отримані випадкові цілі числа проходять усі тести на випадковість, які були застосовані до них. Довші періоди доступні, наприклад, у RNG з помноженням на перенесення, але вони використовують ціле множення і вимагають ведення (іноді великої) таблиці останніх згенерованих значень.

Сам генератор має ряд позитивних якостей та недоліків. Як можна побачити з таблиці 2.1 прості генератори Xorshift 32/64 у порівнянні з іншими є легкими для передбачення, мають проблеми із статистичною якістю, але досить до них додати нелінійну функцію і вона закриває ці проблеми. Також для ускладнення їх передбачуваності є можливість додати до них циклічний зсув.

Основна логіка роботи даного алгоритму побудована на використанні лінійних функцій, що на відміну від інших алгоритмів робить його дуже швидким, хоч і не дуже криптозахищенним. Сам автор алгоритму Джордж Марсалья рекомендує використовувати його як додатковий алгоритм у зв'язці з іншим генератором.

Таблиця 2.1 Порівняння генераторів

	Статистична якість	Складність передбачення	Репродуктивність даних	Багатопоточність	Період	Швидкість	Потрібне місце	Розмір коду та складність
Mersenne Twister	Декілька помилок	Легко	Да	Ні	Великий 2^{19937}	Допустима	Багато (2 KB)	Складний

	Статистична якість	Складність передбачення	Репродуктивність даних	Багатопоточність	Період	Швидкість	Потрібне місце	Розмір коду та складність
Arc4Random	Декілька проблеми	Безпечно	Не завжди	Ні	Великий 2^{1699}	Мала	Багато (0.5 KB)	Складний
ChaCha20	Хороша	Безпечно	Да	Да (2^{128})	2^{128}	Дуже мала	Середньо (0.1 KB)	Складний
Minstd (LCG)	Багато проблем	Тривіально	Да	Ні	Дуже малий $< 2^{32}$	Допустима	Дуже компактний	Дуже малий
LCG 64/32	Багато проблем	Опубліковані алгоритми	Да	Да 2^{63}	Достатній 2^{64}	Дуже висока	Дуже компактний	Дуже малий
XorShift 32	Багато проблем	Тривіально	Да	Ні	Малий 2^{32}	Висока	Дуже компактний	Дуже малий
XorShift 64	Багато проблем	Тривіально	Да	Ні	Достатній 2^{64}	Висока	Дуже компактний	Дуже малий
RanQ	Декілька проблем	Тривіально	Да	Ні	Достатній 2^{64}	Висока	Дуже компактний	Дуже малий
XorShift* 64/32	Прекрасна	Невідомо	Да	Ні	Достатній 2^{64}	Висока	Дуже компактний	Дуже малий

Використовуючи його в даній роботі, я вирішив вдосконалити його використанням нелінійної функції додавання, логічної операції побітове заперечення (NOT) та або (OR), а також циклічним зсувом, збільшивши бітовий розмір вхідних даних до 128 (2 по 64). Для вхідних даних було обрано використовувати апаратні дані комп’ютера з якого запускається генератор за допомогою вбудованого в бібліотеку random.h пакету random_device.

Також завдяки швидкості Xorshift для формування одноразового паролю даний алгоритм запускається 250000 разів, де кожне сформоване число проходить логічну операцію АБО (OR) або додавання за модулем 2 (XOR) в залежності від того, чи є сформоване число парним або ні. Та після операції ділення з остачею у циклі ми отримуємо восьмизначне число, яке буде використовуватися як одноразовий пароль, що є дійсним лише 20 секунд.

Дані маніпуляції над алгоритмом дали змогу використовувати його швидкість та параметри для генерування одноразових паролів для двофакторної автентифікації.

2.1. Тестування вдосконаленого алгоритму Xorshift візуальним методом

Одним із методів аналізу генераторів випадкових чисел є створення візуалізації чисел, які він генерує. Люди дійсно добре розпізнають візерунки, а візуалізація дозволяє використовувати для цього безпосередньо очі та мозок. Хоча такий підхід не можна вважати вичерпним або офіційним аналізом, це хороший і швидкий спосіб скласти враження про ефективність даного генератора.

Для такого тестування було обрано бібліотеку bitmap, що дозволяє видавати зображення декілька інших генераторів серед яких C rand(), php rand(), LCG, Delphi RTL Random та вдосконалений Xorshift.

На зображенні Рис.2.1 ми можемо побачити алгоритм rand(). Виразно можна побачити вертикальні лінії, що говорить о поганому розподіленні випадкових чисел та малому періоду.

На зображенні Рис. 2.2 алгоритм LCG як і rand() має невеликий період і поганий розподіл випадкових чисел, що означає, що використовувати його у базовому вигляді є не доцільним.

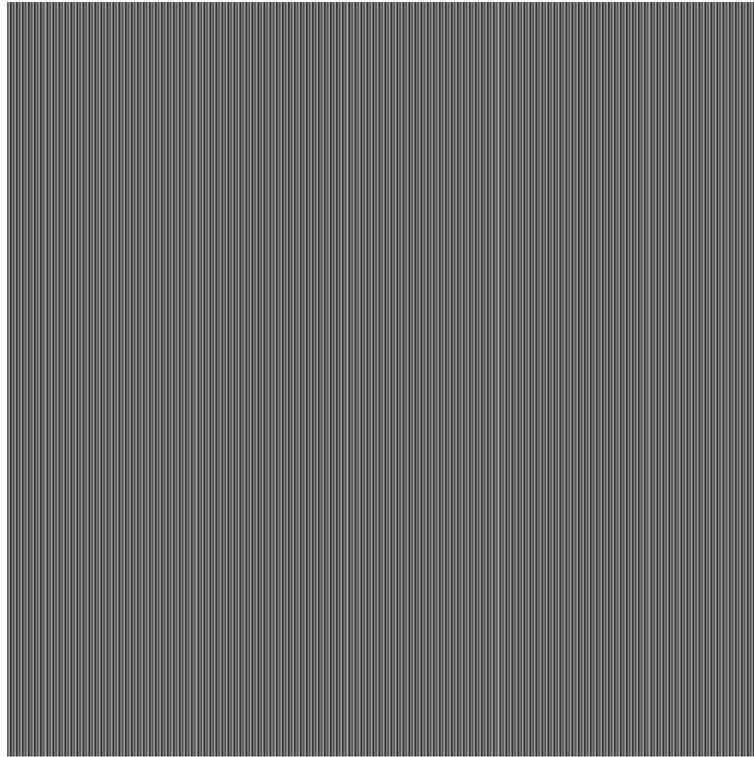

Рис. 2.1. C rand()

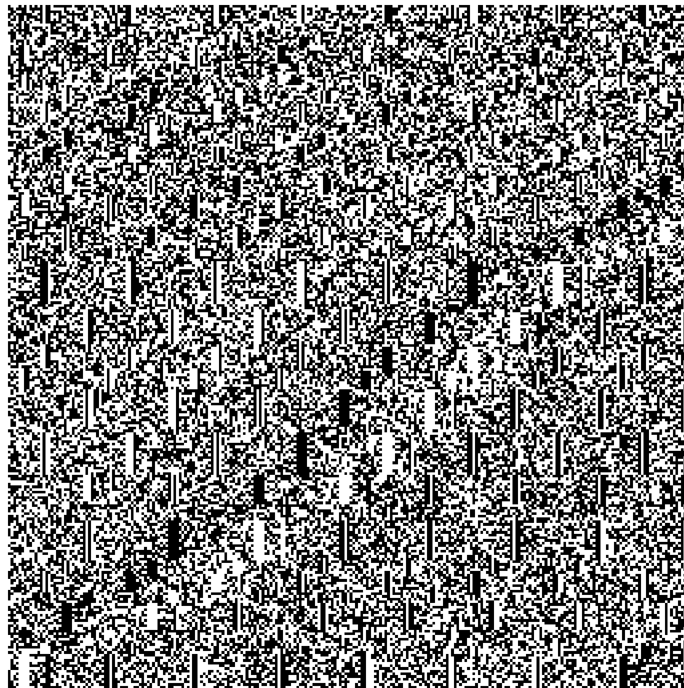

Рис. 2.2. LCG

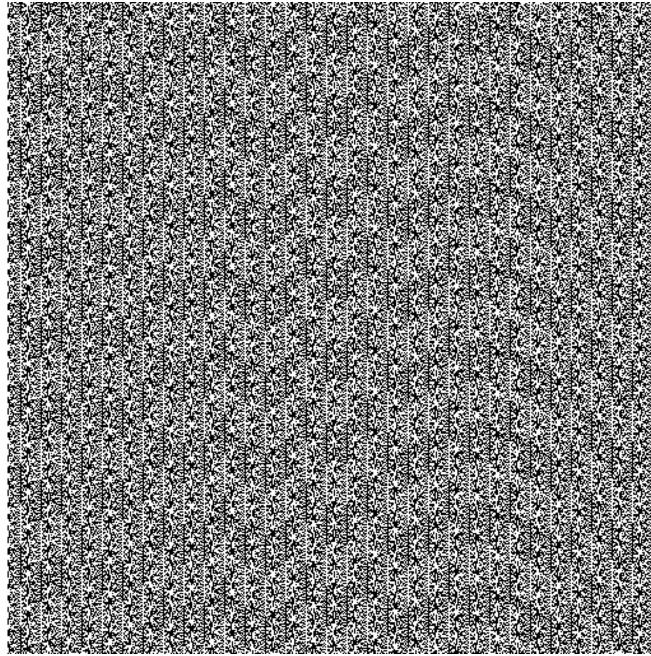

Рис. 2.3. PHP rand()

На зображенні Рис.2.3 можна побачити алгоритм PHP rand() який вже виглядає більш випадковим, але все одно проявляються візерунки та вертикальні лінії.

На Рис. 2.4 у алгоритму Delphi RTL Random чітко проявляються горизонтальні полоси, що означає, що він недостатньо випадковий.

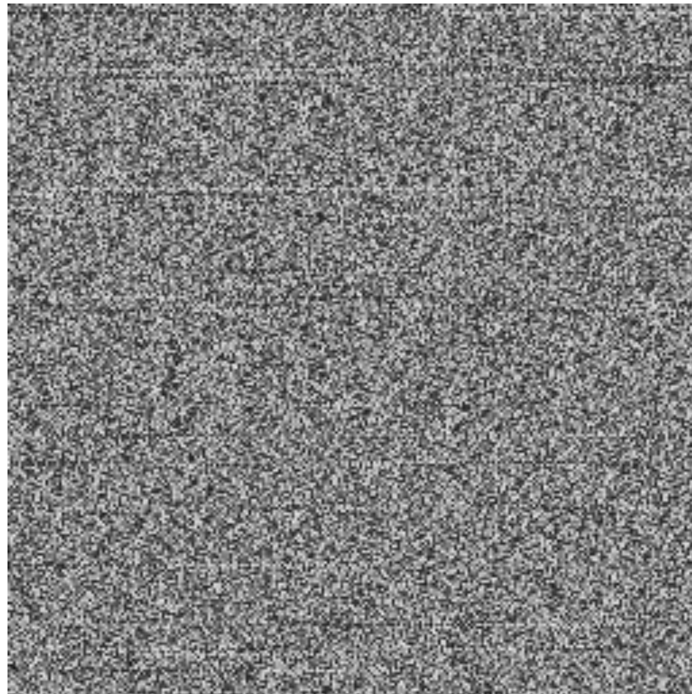

Рис. 2.4. Delphi RTL Random

На Рис. 2.5 алгоритм Xorshift 32/64. Хоч алгоритм і швидкий та якісний, але ми можемо побачити візерунки та діагональні полоси.

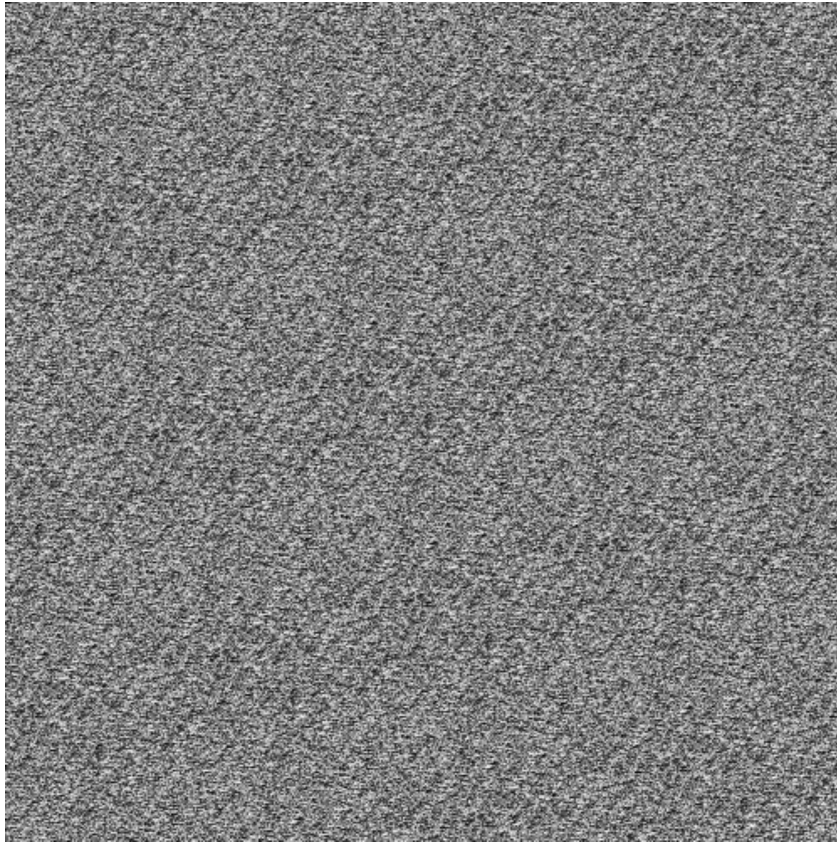

Рис. 2.5. Xorshift 32/64

На Рис. 2.6 зображено вже вдосконалений алгоритм Xorshift, де не проявляються ніякі візерунки чи полоси, що говорить про те, що його випадкові числа можна використовувати, як приклад хорошого генератору.

Для прикладу на Рис. 2.7 зображено розподіл випадкових чисел генератору справжніх випадкових чисел з сайту random.org, де числа генеруються не по алгоритму, а завдяки фізичним явищам.

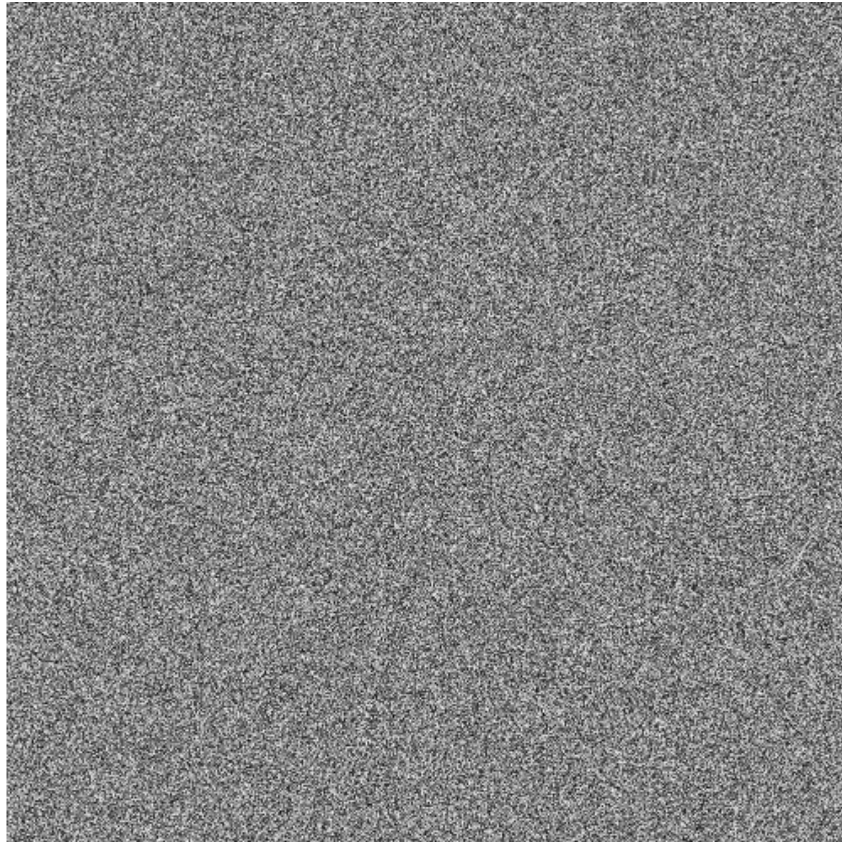

Рис. 2.6. Вдосконалений алгоритм Xorshift

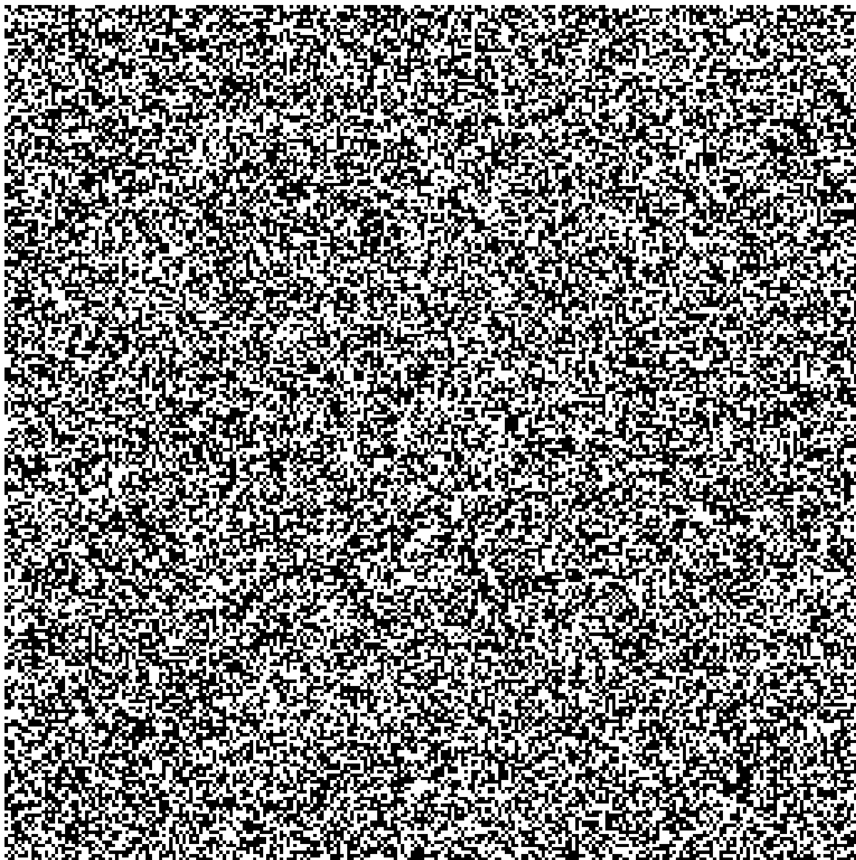

Рис. 2.7 Random.org TRNG

2.2. Тестування алгоритму пакетом статистичних тестів TestU01

TestU01 - це пакет статистичних емпіричних тестів, реалізований на мові ANSI C. Він забезпечує набір корисних програм для тестування генераторів випадкових чисел.

Пакет включає в себе кілька типів ГПВЧ, включаючи деякі запропоновані в літературі і деякі широко використовувані в програмному забезпеченні. Він забезпечує реалізацією класичних статистичних тестів для генераторів випадкових чисел, а також для запропонованих в літературі і деяких оригінальних. Ці тести можна застосувати до власних генераторів та генераторів, які вже існують у бібліотеці, а також до потоку випадкових чисел. Спеціальні тести перевіряють будь-які послідовності рівномірно розподілених випадкових чисел в [0,1] або бітові послідовності. Основні інструменти для побудови векторів точок, що виробляються генераторами, теж присутні.

SmallCrush, маленька і швидка батарея тестів, яка зчитує генератор як файл, що містить числа з плаваючою точкою в діапазоні [0,1). Обмеження на файл трохи менше 51320000 випадкових чисел. Файл буде переписаний на початку кожного тесту. Деякі тести вимагають від генератора повернення як мінімум 30 біт рішення, в іншому випадку генератор з великою ймовірністю провалить їх. Зазвичай використовується для перевірки доцільності проведення тестів на більш строгих батареях.

Crush – Батарея, що складається із строгих статистичних тестів. Включає в себе як класичні тести Кнута, так і безліч інших. Деякі з цих тестів припускають, що генератор повертає як мінімум 30 біт рішення, в іншому випадку тести будуть вважатися проваленими. Один з тестів вимагає 31 біт даних. Батарея використовує приблизно 235 випадкових чисел.

BigCrush - батарея найсуворіших статистичних тестів. Так само як і в інших батареях, деякі тести вимагають як мінімум 30 біт вхідних даних,

інакше тести будуть провалені. Так само один з тестів вимагає 31 біт даних. Батарея використовує майже 238 випадкових чисел. Коли BigCrush вперше з'явився, навіть ГПВЧ, що вважалися хорошими, насилу його проходили.

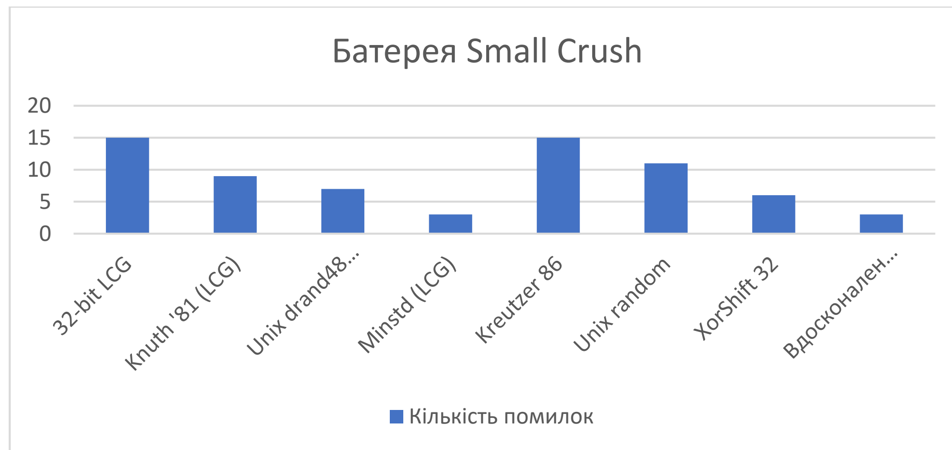

Рис. 2.8. Результат проходження батареї Small Crush пакету статистичних тестів TestU01

У тестах Crush та BigCrush ми можемо побачити, що вдосконалений алгоритм Xorshift сильно відрізняється від свого прототипу. Це виходить через те, що вдосконалений алгоритм завдяки нелінійній функції відкидає 32 біти меншого порядку.

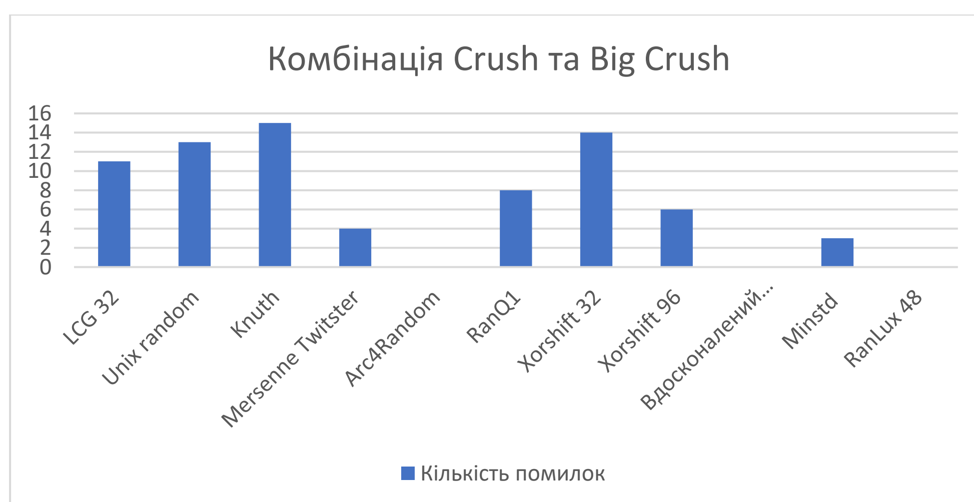

Рис. 2.8. Комбінований результат проходження батарей Crush та Big Crush пакету статистичних тестів TestU01

Висновок до розділу 2.

В даному розділі було описано алгоритм Xorshift та приведено порівняння з іншими алгоритмами генерації псевдовипадкових чисел. Було вирішено удосконалити алгоритм додавши до нього ще логічних операцій таких як OR та NOT, циклічний зсув та функцій додавання та ділення з остачею. Це дало змогу покращити його проходження пакету статистичних тестів та більш випадковості його послідовності чисел. А також зробити складнішою передбачуваність наступних елементів послідовності псевдовипадкових чисел.

	Xorshift 32	**Удосконалений Xorshift**
Швидкість	Швидкий	Швидкий
Потрібне місце	Компактний	Компактний
Розмір коду	Дуже малий	Дуже малий
Статистична якість	Багато проблем	Висока
Складність передбачення	Дуже просто	Досить складно
Період	Малий 2^{32}	Достатній 2^{128}
Використання логічних операцій та нелінійних функцій	XOR, зсув вліво та вправо	OR, XOR, зсув вліво та вправо, циклічний зсув, додавання, NOT, ділення з остачею

Розділ 3. РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ДВОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ ЗА ДОПОМОГОЮ ОДНОРАЗОВИХ ПАРОЛІВ З ВИКОРИСТАННЯМ ВДОСКОНАЛЕНОГО АЛГОРИТМУ XORSHIFT

Першою частиною програмного модуля є користувацький інтерфейс, де користувач вводить свої логін та пароль, тут пароль користувача оброблюється хеш-функцією для безпеки та порівнюється із хешем пароля, що зберігається на сервері. Так само ж порівнюються логіни. У випадку, якщо логін та хеш паролю співпадають, користувач отримує доступ до вікна з одноразовим паролем. Він змінюється за таймером кожні 20 секунд. Після 5 генерацій одноразового паролю генератор зупиняється, а зміст файлу з хешем цього паролю перезаписується на «Empty», для того, щоб неможливо було скористуватися останнім згенерованим паролем.

Також у першій частині програми знаходиться вдосконалений алгоритм Xorshift, блок-схема якого зображена на Рис. 3.2. На блок-схемі показаний процес роботи генератора, що викликає 250000 разів функцію next(), описану в підрозділі 3.2, функцію jump(), та зведення згенерованого числа до 8-мизначного. Ці дії генерують одне число, що буде використовуватися як одноразовий пароль протягом 20 секунд.

Другою частиною програмного модуля є інтерфейс клієнта, де йому потрібно ввести свій логін та одноразовий пароль, що був згенерований у першій програмі. Після цього програма перевіряє чи збігаються вони з тими, що зберігаються у файлі, і, якщо вони однакові, то клієнту надається доступ до ресурсу.

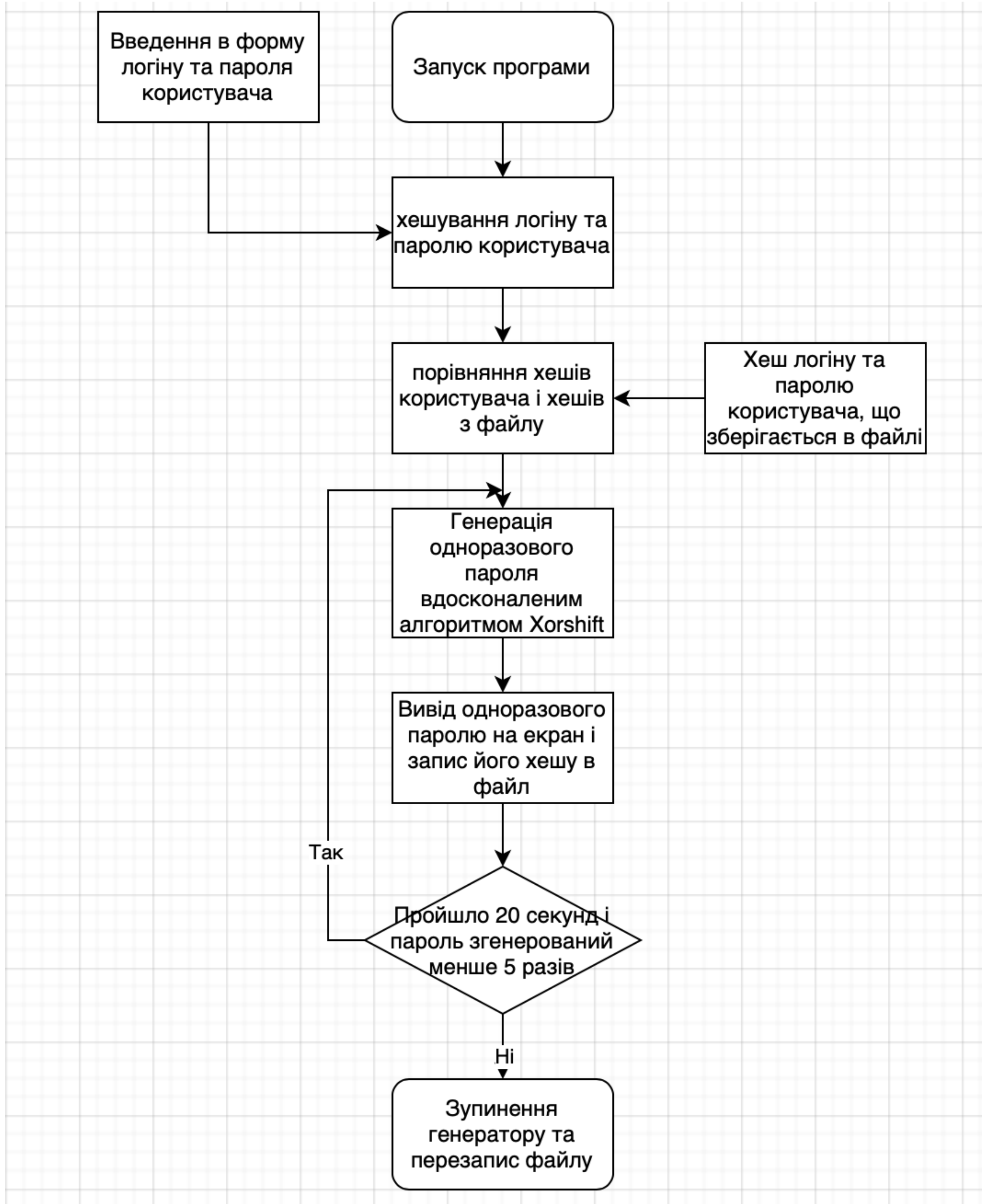

Рис. 3.1. Блок-схема першої частини програмного модуля.

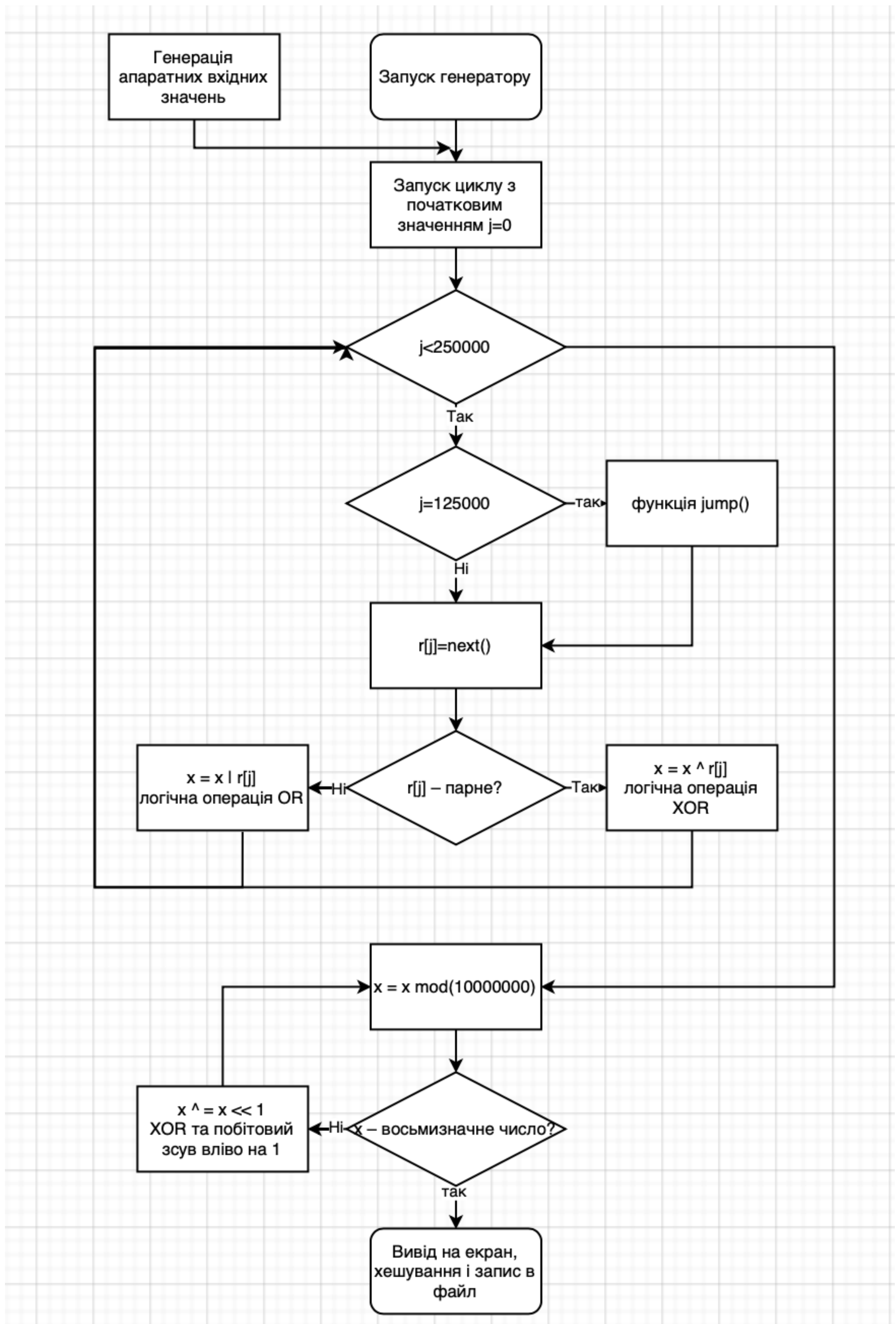

Рис. 3.2. Блок-схема першої частини програмного модуля.

Рис. 3.3. Блок-схема другої частини програмного модуля.

3.1 Середовище розробки, бібліотеки, класи та функції.

Для розробки даного програмного модуля було обрано середовище Qt Creator. Воно найбільш задовольняє потреби розробки, так як підтримує роботу з мовою програмування C++ та форми для графічного інтерфейсу користувача. Для розробки було підключено бібліотеки та заголовні файли з потрібними функціями.

<string> – клас стандартної бібліотеки C++, що потрібен для роботи з рядками.

<fstream> – заголовний файл стандартної бібліотеки C++, що включає набір класів, методів та функцій, які надають змогу записувати дані в файл та

читати їх з нього. Для роботи з цими даними використовуються об'єкти, що називаються потоками.
Функція ifstream – функція зчитування інформації з файлу.
Функція ofstream – функція запису інформації у файл.
<QCryptographicHash> – клас вбудованої бібліотеки Qt Creator, що дозволяє генерувати криптографічний хеш. На вибір в нас є велика кількість алгоритмів: MD4, MD5, SHA1, SHA224, SHA256, SHA384, SHA512, SHA3_224, SHA3_256, SHA3_384, SHA3_512, Keccak_224, Keccak_256, Keccak_384, Keccak_512.
<ctime> – заголовний файл, що містить функції для роботи з часом та датою.
<unistd.h> – заголовний файл, що містить функції для генерації псевдовипадкових чисел на UNIX-системах.
<QTimer> – клас для роботи з таймерами.
<QCloseEvent> -– клас для роботи з програмним та користувацьким вимкненням програми.
<QApplication> – клас для програмою, її графічним інтерфейсом та основними параметрами.
<QProcess> – клас, потрібний для запуску зовнішніх програм та комунікацією з ними.
<QMassageBox> – клас для виведення діалогового вікна для повідомлень.

3.2 Детальний розгляд методів, функцій та складових програмного модуля

Перша частина програмного модуля складається з 3 .cpp файлів, 2 заголовних та 2 форм користувацького інтерфейсу.

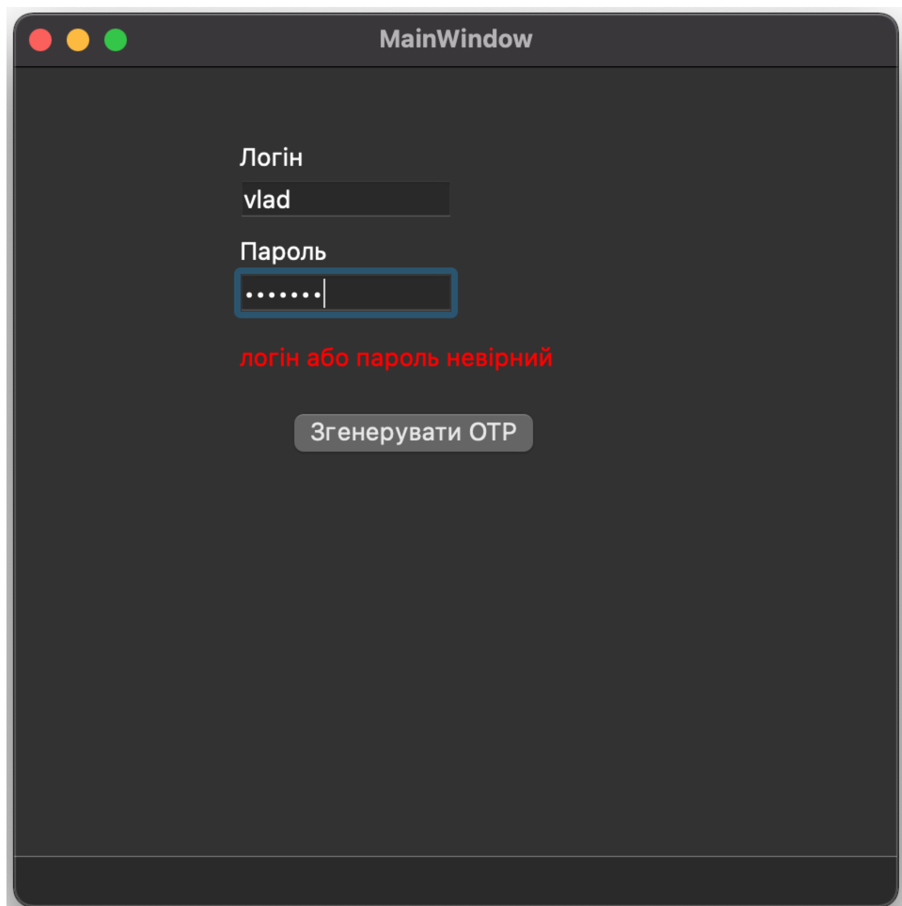

Рис. 3.4. Головне вікно першої програми.

На цій формі ми можемо побачити 3 об’єкти класу QLabel (“Логін”, “Пароль” та “логін или пароль невірний”). Останній стає видимим у випадку, якщо було введено невірний логін або пароль. 2 об’єкти класу QLineEdit (поля для вводу логіна та пароля). І кнопка об’єкт класу QPushButton.

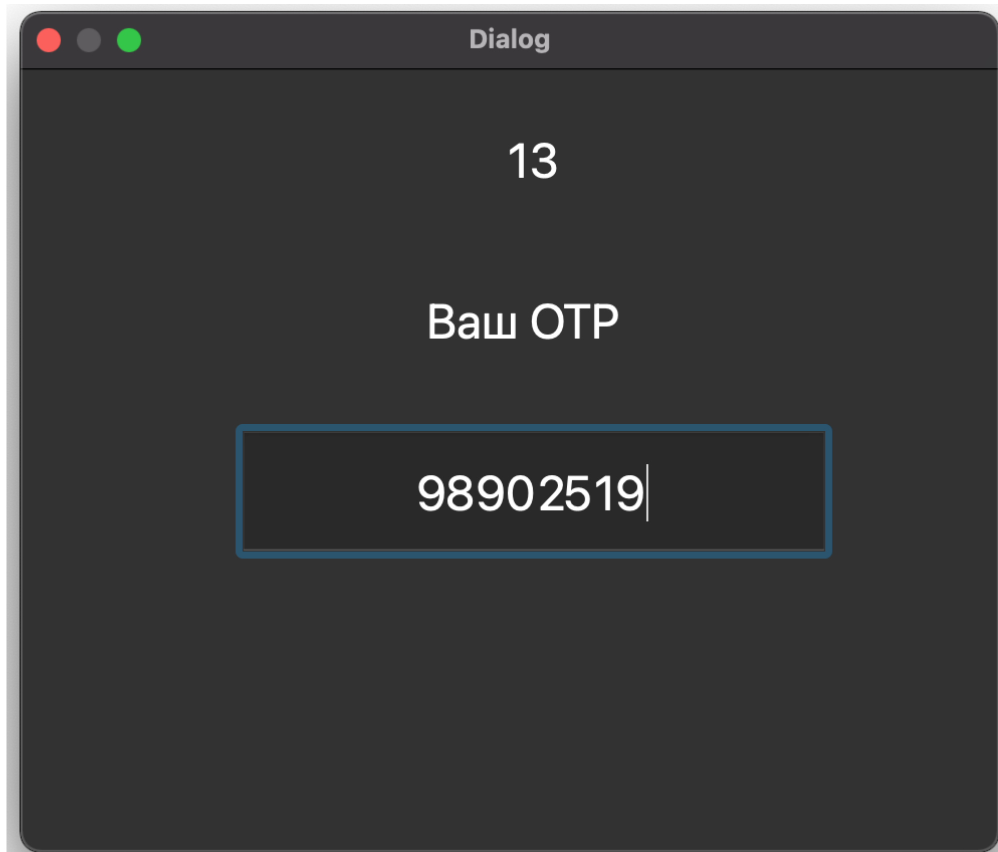

Рис. 3.5. Вікно з таймером і генератором одноразових паролів.

На другій формі ми можемо побачити 3 об’єкти класу QLabel («До зміни пароля», «OTP» та таймер), поле – об’єкт класу QLineEdit, в який виводиться наш одноразовий пароль для зручного копіювання, та кнопка об’єкт класу QPushButton, що з’являється після закінчення генерації.

Після того, як програмний модуль згенерував одноразовий пароль протягом двох хвилин, його робота зупиняється. Два об’єкти класу QLabel ховаються, а кнопка QPushButton з’являється. Об’єкт QLabel, що відображав таймер, тепер має надпис, що час вийшов («Час вийшов!»). А натискання на кнопку веде до перезапуску програми.

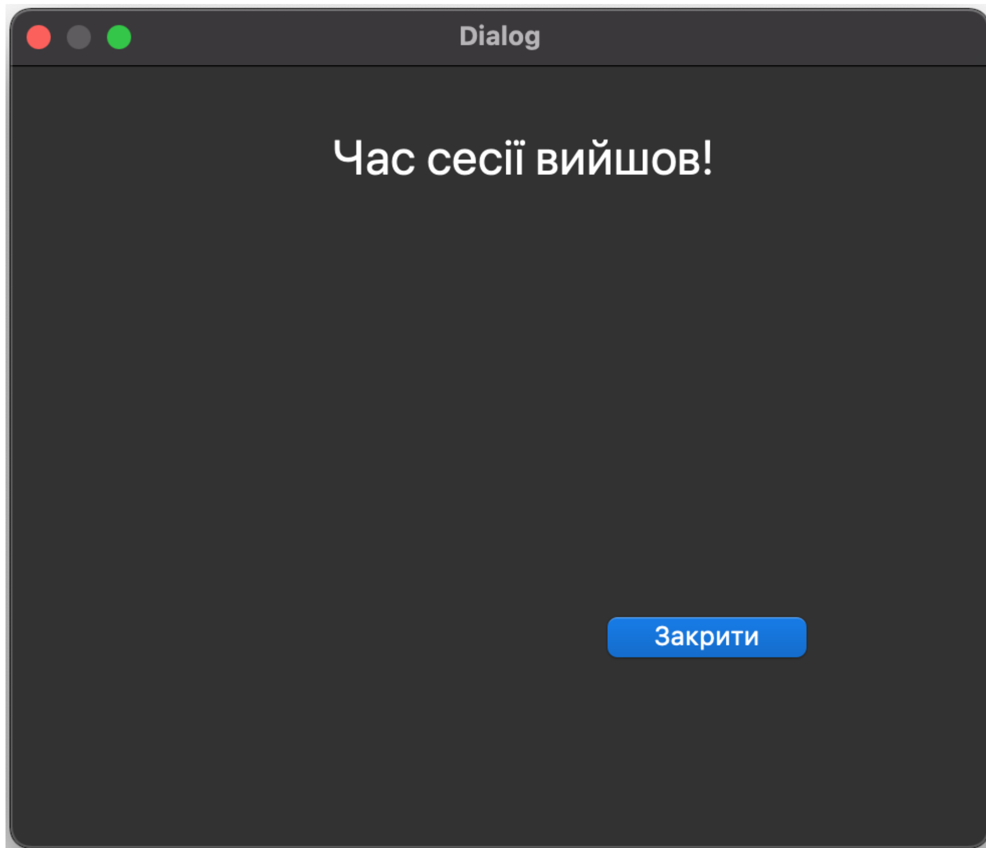

Рис. 3.6. Вікно, що попереджає про закінчення часу на введення пароля і пропозицією повторити.

Заголовний файл mainwindow.h містить інформацію та об'явлення об'єктів та класів головного вікна програми. Також ми можемо побачити функцію натискання на кнопку on_pushButton_clicked(), у розділі private slots. Саме ця функція запускає перевірку інформації введеної користувачем у поля – об'єкти класу QLineEdit з даними, що зберігаються на сервері.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();

private slots:
   void on_pushButton_clicked();

private:
   Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Заголовному файл **secondwindow.h** так само як і mainwindow.h містить інформацію про класи та об’єкти, але для другого вікна, що відкривається після введення правильних даних у першому вікні. Серед об’єктів у даному заголовочному файлі можна побачити функції private slots серед яких функція void slotTimerAlarm() та void slotTimerAlarm2(), що відповідають за

зміну часу у вікні та затримкою перед генерацією нового одноразового пароля.

```
#ifndef SECONDWINDOW_H
#define SECONDWINDOW_H
#include <QDialog>

namespace Ui {
class SecondWindow;
}

class SecondWindow : public QDialog
{
   Q_OBJECT

public:
   explicit SecondWindow(QWidget *parent = nullptr);
   ~SecondWindow();

private slots:
   void slotTimerAlarm();
   void slotTimerAlarm2();
   void closeEvent(QCloseEvent *event);
   void on_pushButton_clicked();
private:
   Ui::SecondWindow *ui;
};

#endif // SECONDWINDOW_H
```

Програмному файлі **Main.cpp** викликає вікно для користувача, де він буде вводити свою інформацію для отримання доступу до другого вікна з одноразовим паролем.

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

У програмному файлі **Mainwindow.cpp** ми викликаємо вікно з полями для введення інформації щодо логіну та паролю користувача та оголошуємо об’єкт батьківського класу MainWindow – ui, що дає змогу звертатися до об’єктів та класів користувацького інтерфейсу.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "secondwindow.h"
#include <fstream>
#include <string>
#include <QCryptographicHash>
```

```
using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

Функція **on_pushButton_clicked**(), що була об'явлена у заголовному файлі mainwindow.h запускає процес зчитування інформації про логін та пароль з віджету QLineEdit.

```
void MainWindow::on_pushButton_clicked()
{
    QString login=ui->login->text();
    QString pass=ui->pass->text();
    string text1="", text2="";
    ifstream in;
    in.open("/Users/vladyslavmuzyka/login.txt");
    in>>text1;
    in.close();
    in.open("/Users/vladyslavmuzyka/pass.txt");
```

```
   in>>text2;
   in.close();
   QString str = QString::fromUtf8(text1.c_str());
   QString str2 = QString::fromUtf8(text2.c_str());
   QByteArray ba = login.toUtf8();
   QString aba = QCryptographicHash::hash(ba, QCryptographicHash::Sha256).toHex();
   QByteArray baz = pass.toUtf8();
   QString abaz = QCryptographicHash::hash(baz, QCryptographicHash::Sha256).toHex();
   if(aba==str && abaz==str2)
   {hide();
   SecondWindow window;
   window.setModal(true);
   window.exec();}
   else
   {
      ui->label_3->setText("<html><body><p><span style='color:#ff0000;'>логін або пароль невірний</span></p></body></html>");
   }
}
```

Оголошуємо змінні типу QString та зчитуємо за допомогою функції text().

```
QString login=ui->login->text();
QString pass=ui->pass->text();
```

Оголошуємо змінні типу string, куди ми запишемо інформацію про логін і пароль, що зберігаються на сервері.

```
string text1="", text2="";
```

За допомогою функцій заголовного файлу fstream ми створюємо об'єкт in класу ifstream для зчитування інформації з файлу.

```
ifstream in;
```

Використовуючи метод open() ми надаємо нашому об'єкту шлях до файлу, в якому зберігається логін.

```
in.open("/Users/vladmuzyka/login.txt");
```

Записуємо у створену змінну типу string інформацію з цього файлу.

```
in>>text1;
```

Та закриваємо файл.

```
in.close();
```

Повторюємо те ж саме з паролем.

```
in.open("/Users/vladmuzyka/pass.txt");
in>>text2;
in.close();
```

Так як у об’єктах класу QLineEdit інформація зберігається як тип QString, потрібно зчитану інформацію з файлу конвертувати в цей самий тип.

```
QString str = QString::fromUtf8(text1.c_str());
QString str2 = QString::fromUtf8(text2.c_str());
```

Пароль зберігається як хеш пароля за алгоритмом SHA-256 сімейства SHA-2. Ця хеш-функція є найбільш оптимальною з точки зору швидкості та стійкості до колізій і пошуку першого та другого праобразу. Він створений на основі структури Меркла-Дамгора. Оригінал тексту після доповнення розбивається на блоки, кожен блок - на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64 ітераціями (раундами). На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням хеш-функції. Проте, ініціалізація внутрішнього стану проводиться результатом обробки попереднього блоку. Тому незалежно обробляти блоки і складати результати не можна.

Так як функція hash() класу QCryptographichash може працювати лише зі стандартом кодування тексту Utf8, то ми виконуємо переформатування нашого пароля.

```
QByteArray ba = pass.toUtf8();
```

Виконуємо хеш-функцію та призводимо до вигляду Hex (16-розрядного).

QString aba = QCryptographicHash::hash(ba, QCryptographicHash::Sha256).toHex();

Табл. 3.1.

Порівняння алгоритмів хешування SHA-1 та SHA-256.

Варіації алгоритму	**SHA-1**	**SHA-256**
Розмір вхідного хешу (біт)	160	256
Проміжний розмір хешу (біт)	160	256
Розмір блоку (біт)	512	512
Максимальна довжина вхідного повідомлення (біт)	$2^{64} - 1$	$2^{64} - 1$
Розмір слова (біт)	32	32
К-ть раундів	80	64
Операції	+,and, or, xor, rotl	+,and, or, xor, shr, rotr
Знайдені колізії	2^{52} операцій	Ні

Задаємо умову, при якій, якщо логін та хеш паролю збігаються з тими, що зберігаються на сервері, то ховається дане вікно та відкривається інше, інакше – виводиться повідомлення, що логін або пароль неправильний.

```
if(login==str && aba==str2)
    {hide();
    SecondWindow window;
    window.setModal(true);
    window.exec();}
    else {
        ui->label_3->setText("<html><body><p><span
style='color:#ff0000;'>логин или пароль
неверный</span></p></body></html>");
    }
```

Програмний файл **Secondwindow.cpp** відкриває друге вікно, що виконує генерацію одноразового паролю для подальшої автентифікації у другій програмі. Перш за все ми підключаємо бібліотеки для коректної роботи генератора.

```
#include "secondwindow.h"
#include "ui_secondwindow.h"
#include "mainwindow.h"
#include <string>
#include <fstream>
#include <ctime>
```

```
#include <unistd.h>
#include <QApplication>
#include <QProcess>
#include <QCloseEvent>
#include <QCryptographicHash>
#include <QTimer>
#include <random>
```

Для таймеру та лічильника генерацій одноразових паролів ми оголошуємо змінні і та k.

```
int i=20;
int k=0;
```

Оголошуємо масив змінних s у розмірі 2 типу uint64_t, що закріплює довжину цілого числа у 64 біти. Ці змінні потрібні для ініціалізації генератора, так як вони будуть містити перші числа, з яких почне працювати генератор.

```
uint64_t s[2];
```

Оголошуємо 2 таймера, що потрібні для лічильника та зміни одноразових паролів.

```
QTimer *timer = new QTimer();
QTimer *timer2 = new QTimer();
```

Функція closeEvent(QCloseEvent *event) відповідає за заміну змісту файлу з одноразовим паролем на «Empty» при закритті програми, для того,

щоб після процесу автентифікації не можливо було скористатися останнім одноразовим паролем.

```
void SecondWindow::closeEvent(QCloseEvent *event)
{
    ofstream out;
    out.open("/Users/vladyslavmuzyka/otp.txt");
    out<<"Empty";
    out.close();
    event->accept();
}
```

Дана функція відповідає за виклик вікна, апаратну генерацію даних для ініціалізації генератора та запуску таймерів.

```
SecondWindow::SecondWindow(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SecondWindow)
{
    ui->setupUi(this);
    ui->label->setText(QString::number(i));
```

Даний процес генерує випадкові числа типу int опираючись на апаратні властивості комп’ютеру в діапазоні від 1000 до 1000000.

```
    random_device zx;
    uniform_int_distribution<int> uid(1000, 1000000);
    s[0]=uid(zx);
    s[1]=uid(zx);
```

Ховаємо графічний елемент з кнопкою, що потім буде використовуватися як вихід з програми.

```
    ui->pushButton->hide();
```

Ініціалізуємо функції з лічильником та генерацією одноразових паролів.

```
    slotTimerAlarm();
    slotTimerAlarm2();
```

Запускаємо таймери, що змінюють кожну секунду число на лічильнику та кожні 20 секунд одноразовий пароль.

```
    timer = new QTimer();
    timer2 = new QTimer();
    connect(timer, SIGNAL(timeout()), this, SLOT(slotTimerAlarm()));
    connect(timer2, SIGNAL(timeout()), this, SLOT(slotTimerAlarm2()));
    timer->start(1000);
    timer2->start(20000);
}
```

Функція SecondWindow::slotTimerAlarm() відповідає за зміну числа на екрані програми у вигляді лічильника, що починається з 20, і коли число доходить до 1, відлік починається з початку.

```
void SecondWindow::slotTimerAlarm()
{
    ui->label->setText(QString::number(i));
```

```
    if(i>1) {
        i--;
    }
    else {
        i=20;
    }
}
```

Дана функція використовується як циклічне зрушення бітів для генератора одноразових паролів.

```
static inline uint64_t rotl(const uint64_t x, int k) {
    return (x << k) | (x >> (64 - k));
}
```

Ця функція є генератором випадкових чисел, що включає в себе декілька бітових операцій, серед яких: XOR, NOT та циклічні зсуви, що дає змогу цьому генератору працювати дуже швидко та формувати велику послідовність випадкових чисел за короткий час.

```
uint64_t next(void) {
    const uint64_t s0 = s[0];
    uint64_t s1 = s[1];
    const uint64_t result = s0 + s1;
    s1 ^=  s0;
    s[0] = rotl(s0, 24) ^ s1 ^ (s1 << 16);
    s[1] = rotl(s1, 37);
    return ~result; }
```

Функція void jump(void) відповідає за «стрибок» функції у послідовності, що еквівалентно 2^64 викликам генератора. Ця функція потрібна, щоб ускладнити передбачення наступних чисел в послідовності.

```
void jump(void) {
   static const uint64_t JUMP[] = { 0xdf900294d8f554a5, 0x170865df4b3201fc
};

   uint64_t s0 = 0;
   uint64_t s1 = 0;
   for(int i = 0; i < sizeof JUMP / sizeof *JUMP; i++)
      for(int b = 0; b < 64; b++) {
         if (JUMP[i] & UINT64_C(1) << b) {
            s0 ^= s[0];
            s1 ^= s[1];
         }
         next();
      }
   s[0] = s0;
   s[1] = s1;
}
```

Функція SecondWindow::slotTimerAlarm2() викликається кожні 20 секунд і включає в себе процес перевірки змісту файлу з одноразовими паролями, його заповнення хешем результату генерації випадкових чисел та запуск самого генератору.

```
void SecondWindow::slotTimerAlarm2()
{
```

```
string tmp="";
string check;
ifstream in;
in.open("/Users/vladyslavmuzyka/otp.txt");
in>>check;
in.close();
if(check!="Empty"||k==0)
{
    ofstream out;
    out.open("/Users/vladyslavmuzyka/otp.txt");
    uint64_t r[250000];
    uint64_t x=1;
    for(int j=0; j<250000; j++)
    {
        r[j]=next();
        if(j==125000)
        {jump();}
         if(r[j]%2==0)
            {
            x|=r[j];
            }
            else
            {
            x^=r[j];
            }
    }
string x_str="";
x=x%100000000;
x_str=to_string(x);
```

```
    while(x_str.length()<8)
    {
        x^=x<<1;
        x=x%100000000;
        x_str=to_string(x);
    }
    tmp=to_string(x);
    QString str = QString::fromUtf8(tmp.c_str());
    ui->otp->setText(str);
    QByteArray ba = str.toUtf8();
    QString aba = QCryptographicHash::hash(ba,
QCryptographicHash::Sha256).toHex();
    string tmp2 = aba.toStdString();
    out<<tmp2;
    out.close();
    k++;
    if(k==5)
    {
    timer->stop();
    timer2->stop();
    ui->label->setText("Час сесії вийшов!");
    ui->pushButton->show();
    ui->label_3->hide();
    ui->otp->hide();
    ui->label_2->hide();
    ofstream out;
    out.open("/Users/vladyslavmuzyka/otp.txt");
    out<<"Empty";
    out.close();
```

```
    }}
    else
    {
        timer->stop();
        timer2->stop();
        ui->label->setText("Вхід вже виконаний!");
        ui->pushButton->show();
        ui->label_3->hide();
        ui->otp->hide();
        ui->label_2->hide();
    }
}
```

Оголошуємо змінні типу string, в які ми будемо записувати дані з файлу та результат генерації, для его внесення в файл.

```
    string tmp="";
    string check;
```

Відкриваємо файл, зчитуємо його зміст, записуємо його у змінну та закриваємо файл.

```
    ifstream in;
    in.open("/Users/vladyslavmuzyka/otp.txt");
    in>>check;
    in.close();
```

Перевіряємо умову для запуску алгоритму, при якій зміст файлу не повинен бути «Empty» або кількість генерацій одноразового пароля дорівнює 0.

```
if(check!="Empty"||k==0)
```

При виконанні даної умови ми перш за все відкриваємо файл одноразового пароля для запису.

```
ofstream out;
out.open("/Users/vladyslavmuzyka/otp.txt");
```

Оголошуємо змінні типу uint64_t в які будуть записуватися результати функції next(). Змінна r є масивом на 250000 елементів.

```
uint64_t r[250000];
uint64_t x=1;
```

Запускаємо цикл на 250000 ітерацій, при якому на кожну з них у відповідний елемент записується результат функції next(), через 125000 ітерацій виконується функція jump(). Далі у випадку, якщо згенероване число є парним, то у змінну x записується результат бінарної операції OR, якщо число непарне, то результат операції XOR, що дає змогу ускладнити предиктивність наступного випадкового числа, що буде використовуватися як одноразовий пароль.

```
for(int j=0; j<250000; j++)
{
    r[j]=next();
```

```
    if(j!=125000)
    {jump();}
     if(r[j]%2==0)
        {
        x|=r[j];
        }
        else
        {
        x^=r[j];
        }
  }
```

Так як результат генерації випадкового числа нам потрібен у вигляді 8-значного числа ми проводимо операцію ділення з остачею результат алгоритму на 100000000, результат ми записуємо у об'явлену змінну x_str та запускаємо цикл з умовою, що довжина змінної менше 8. Якщо умова виконується, то змінна x перезаписується результатом операцій XOR та бінарного зсуву вліво та діленням з остачею на 100000000. Цей цикл працює доки умова не буде виконана.

```
string x_str="";
x=x%100000000;
x_str=to_string(x);
while(x_str.length()<8)
{
   x^=x<<1;
   x=x%100000000;
   x_str=to_string(x);
}
```

Після цього ми записуємо в змінну tmp результат. Так як в елемент QLabel можна записувати тільки змінні типу QString то створюємо таку змінну і записуємо в неї tmp. Створюємо хеш цієї змінної та записуємо його у файл з одноразовими паролями.

```
tmp=to_string(x);
    QString str = QString::fromUtf8(tmp.c_str());
    ui->otp->setText(str);
    QByteArray ba = str.toUtf8();
    QString aba = QCryptographicHash::hash(ba, QCryptographicHash::Sha256).toHex();
    string tmp2 = aba.toStdString();
    out<<tmp2;
    out.close();
```

Збільшуємо лічильник кількості виконаних генерацій, при досягненні 5 разів ми зупиняємо таймери, виводимо на екран повідомлення, що час сесії вийшов. Та перезаписуємо файл с одноразовим паролем на «Empty».

```
    k++;
    if(k==5)
    {
    timer->stop();
    timer2->stop();
    ui->label->setText("Час сесії вийшов!");
    ui->pushButton->show();
    ui->label_3->hide();
    ui->otp->hide();
    ui->label_2->hide();
```

```
    ofstream out;
    out.open("/Users/vladyslavmuzyka/otp.txt");
    out<<"Empty";
    out.close();
    }}
```

Якщо у файлі записано «Empty», а значення лічильника k більше 0, то виконання програми зупиняється та виводиться повідомлення, що вхід вже виконаний.

```
    else
    {
        timer->stop();
        timer2->stop();
        ui->label->setText("Вхід вже виконаний!");
        ui->pushButton->show();
        ui->label_3->hide();
        ui->otp->hide();
        ui->label_2->hide();
    }
}
```

Функція, що при натисканні на кнопку закриває программу.

```
void SecondWindow::on_pushButton_clicked()
{
    qApp->quit();
}
```

Друга частина програмного модуля складається з одного .cpp файлу, однієї форми та одного заголовного файлу.

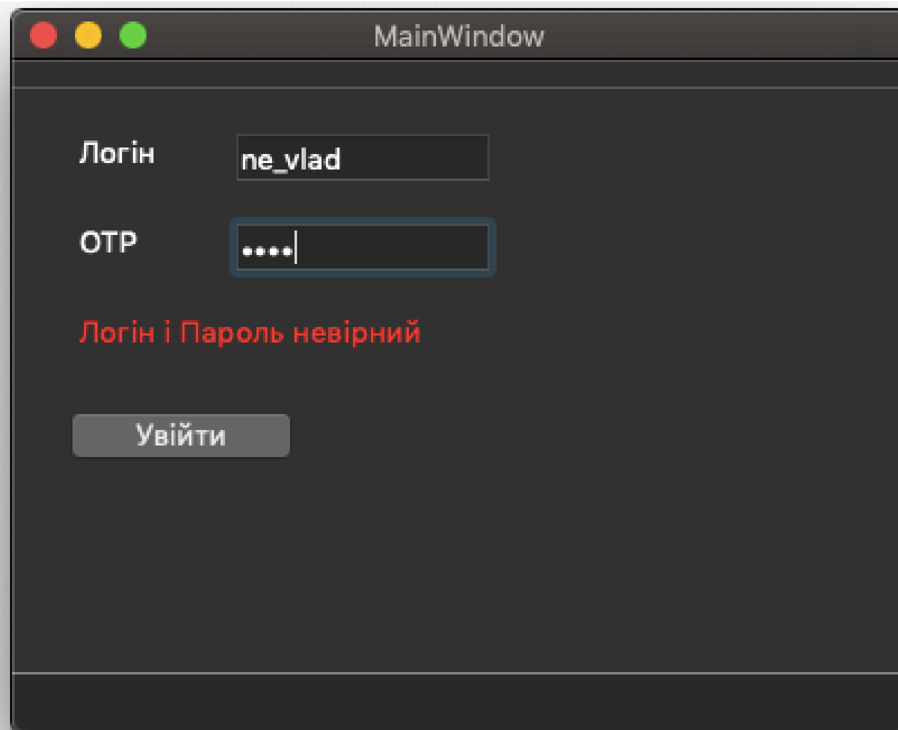

Рис. 3.7. Головне вікно другої частини програмного модуля.

На даній формі ми можемо побачити 3 об’єкти класу QLabel («Логін», «OTP» та «Пароль невірний» (у випадку невірного логіна буде надпис «логін невірний», а якщо невірні і логін, і пароль, то надпис буде «Логін и пароль невірні»)). Останній стає видимим у випадку, якщо було введено невірний логін або пароль. 2 об’єкти класу QLineEdit (поля для вводу логіна та пароля). І кнопка об’єкт класу QPushButton.

Якщо введено правильний логін і пароль, то випливає повідомлення, що доступ надано.

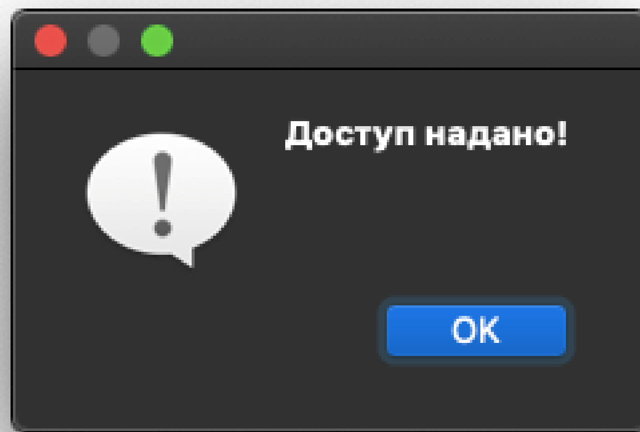

Рис. 3.8. Попередження, що виникає при вдалому вході.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
```

```
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

Заголовний файл містить інформацію про створення класу MainWindow та оголошення приватного слота під функцію onPushButton_clicked(), що спрацьовує при натисканні на кнопку.

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Файл main.cpp запускає програму та відкриває вікно MainWindow.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <fstream>
```

```
#include <iostream>
#include <string>
#include <QMessageBox>
#include <QCryptographicHash>
using namespace std;

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->label_3->hide();
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{
    QString login=ui->login->text();
    QString pass=ui->pass->text();
    string text1="", text2="";
    ifstream in;
    ofstream out;
    in.open("/Users/muzyka/login.txt");
    in>>text1;
    in.close();
```

```
    in.open("/Users/muzyka/otp.txt");
    in>>text2;
    in.close();
    QString str = QString::fromUtf8(text1.c_str());
    QString str2 = QString::fromUtf8(text2.c_str());
    QByteArray ba = pass.toUtf8();
    pass = QCryptographicHash::hash(ba,
QCryptographicHash::Sha256).toHex();
    if(login==str && pass==str2 && pass!="Empty")
    {
        QMessageBox::information(this,"Login Successful","Access is Granted!");
        out.open("/Users/muzyka/otp.txt");
        out<<"Empty";
        out.close();
    }
    else {
        if(login!=str && pass!=str2)
        {
        ui->label_3->show();
        }
        else {
            if((pass!=str2&&login==str)||pass=="Empty")
            {
                ui->label_3->show();
                ui->label_3->setText("<font color = red>Пароль
неверный<\\font>");
            }
            else {
                ui->label_3->show();
```

```
            ui->label_3->setText("<font color = red>Логин неверный<\\font>");
          }
       }}
}
```

В програмі mainwindow.cpp задаємо спочатку об’єкт графічного інтерфейсу ui і його деструктор та ховаємо об’єкт QLabel, що повідомляє про неправильно введений логін або пароль.

```
MainWindow::MainWindow(QWidget *parent) :
   QMainWindow(parent),
   ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   ui->label_3->hide();
}

MainWindow::~MainWindow()
{
   delete ui;
}
```

Додаємо функцію onPushButton_clicked(), що спрацьовує при натисканні на кнопку.

```
void MainWindow::on_pushButton_clicked()
```

Оголошуємо змінні типу QString, де будуть зберігатись логін та одноразовий пароль, що користувач ввів у відповідні поля.

```
QString login=ui->login->text();
QString pass=ui->pass->text();
```

Оголошуємо змінні типу string, куди ми запишемо логін та одноразовий пароль, що зберігаються у файлах.

```
string text1="", text2="";
```

Створюємо об’єкт класу ifstream для зчитування з файлів логіну і паролю.

```
ifstream in;
```

Створюємо об’єкт класу ofstream для перезапису одноразового паролю при закінченні роботи програми.

```
ofstream out;
```

Відкриваємо файл, де зберігається логін, з яким будемо порівнювати введений користувачем.

```
in.open("/Users/muzyka/login.txt");
```

Записуємо дані у створену раніше змінну типу string.

```
in>>text1;
```

Та закриваємо файл.

```
in.close();
```

Повторюємо те ж саме з файлом, де зберігається одноразовий пароль.

```
in.open("/Users/muzyka/otp.txt");
in>>text2;
in.close();
```

Перетворюємо змінні типу string на об'єкти типу QString, що порівняння було моживим.

```
QString str = QString::fromUtf8(text1.c_str());
QString str2 = QString::fromUtf8(text2.c_str());
```

Так як згенерований пароль зберігається у вигляді хешу, а користувач для зручності отримує його у вигляді випадкової комбінації з 8 цифр, нам потрібно пропустити одноразовий пароль, що вводив клієнт через таку саму хеш-функцію SHA-256. Для цього ми спочатку записуємо цей одноразовий пароль у масив типу QByteArray.

```
QByteArray ba = pass.toUtf8();
```

Та перезаписуємо змінну, в якій містився одноразовий пароль введений користувачем, на результат хеш-функції.

```
pass = QCryptographicHash::hash(ba, QCryptographicHash::Sha256).toHex();
```

Створюємо умову для порівняння паролів. В неї, окрім логічних умов, щоб логін та пароль введені користувачем збігалися з тими, що

зберігаються у файлах, ми додаємо ще, що одноразовий пароль не повинен збігатися зі словом «Empty», так як це слово ми використовуємо для запобігання автентифікації при вимкненій програмі генерації (іншими словами, «Empty» – це заглушка).

```
if(login==str && pass==str2 && pass!="Empty")
```

При виконанні даної умови, відкривається вікно з повідомленням, що доступ гарантовано.

```
QMessageBox::information(this,"Вхід вдалий","Доступ надано!");
```

Після цього відкривається файл, де зберігається одноразовий пароль.

```
out.open("/Users/muzyka/otp.txt");
```

В нього записується слово «Empty», та він закривається.

```
out<<"Empty";
out.close();
```

Якщо ж умова не виконується, то програма перевіряє іншу умову.

```
else {
    if(login!=str && pass!=str2)
```

При її виконанні у вікні з’являється повідомлення, що логін та пароль неправильні.

```
ui->label_3->show();
```

Інакше програма перевіряє останню умову, щоб виявити неправильний логін чи пароль і видає повідомлення користувачу.

```
else {
    if((pass!=str2&&login==str)||pass=="Empty")
```

У даному випадку програма виводить повідомлення, що пароль невірний.

```
ui->label_3->show();
ui->label_3->setText("<font color = red>Пароль невірний<\\font>");
```

А у протилежному випадку повідомлення попереджує про невірний логін.

```
else {
      ui->label_3->show();
      ui->label_3->setText("<font color = red>Логін невірний<\\font>");
    }
```

Висновок до розділу 3.

В даному розділі була розроблена програмна реалізація системи забезпечення двофакторної автентифікації одноразовими паролями. Були розроблені алгоритми та блок-схеми, описані методи та класи, що використовувалися при створенні даного програмного модуля. Програмне забезпечення складається з 2 частин. Перша частина відповідає за введення

даних авторизації користувача, їх хешування та порівняння з хешем, що зберігається у спеціальному файлі, генерування одноразового пароля вдосконаленим алгоритмом Xorshift. Друга частина програми відповідає за перевірку логіну та одноразового паролю користувача, що вводяться у відповідні поля. У випадку, якщо користувач ввів все правильно, йому надається доступ до ресурсу. Всі персональні дані та одноразові паролі зберігаються у вигляді хешу, щоб зловмисники навіть отримавши доступ до файлів з цими даними, не змогли отримати доступ до ресурсу. Після виконання входу та при виході з програми, одноразовий пароль у файлі замінюється на слово «Empty», що перевіряється програмою, аби зловмисник не міг скористуватися останнім використаним паролем.

ВИСНОВКИ

Проаналізувавши наявні методи генерації випадкових чисел, а також типи генераторів, їх переваги та недоліки, було зроблено висновок, що найбільш доцільнішим є використання програмного (алгоритмічного) генератору, вхідними даними якого виступають апаратні параметри комп’ютера, на якому здійснює роботу генератор. Серед багатьох алгоритмів для його вдосконалення був обраний алгоритм Xorshift через його швидкість, об’єм займаної пам’яті та простоту інтеграції в інформаційну систему. Було розроблено вдосконалення цього алгоритму та програмну реалізацію двофакторної автентифікації одноразовими паролями з використанням вдосконаленого алгоритму Xorshift. Всі персональні дані та одноразові паролі проходять хешування за алгоритмом SHA-256 для забезпечення їх безпеки від витоку даних.

В результаті вдосконалений алгоритм Xorshift показав себе порівняно краще у візуальних та статистичних тестах. При його використанні у програмному модулі двофакторної автентифікації він спрацьовує дуже швидко та надійно. Сам алгоритм залишився легким для інтегрування в систему та не вимагає багато місця і ресурсів.