КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (22), 2023
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2023.22.191203
УДК 004.077

Черненко Роман Миколайович
аспірант кафедри інформаційної та кібернетичної безпеки імені професора Володимира Бурячка
Київський університет імені Бориса Грінченка, Київ, Україна
ORCID 0000-0002-1439-961X
r.chernenko.asp@kubg.edu.ua

ГЕНЕРАЦІЯ ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ НА МІКРОКОНТРОЛЕРАХ З ОБМЕЖЕНИМИ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ, ДЖЕРЕЛА ЕНТРОПІЇ ТА ТЕСТУВАННЯ СТАТИСТИЧНИХ ВЛАСТИВОСТЕЙ

Анотація. Традиційні алгоритми шифрування не можуть бути реалізовані на пристроях інтернету речей через їх обмежені обчислювальні ресурси. Це зумовлює необхідність в пошуку та розробці рішень для криптозахисту даних, що обробляються та передаються такими пристроями. При шифруванні даних на пристроях з обмеженими обчислювальними ресурсами можна використовувати прості алгоритми шифрування засновані на елементарних побітових операціях, таких як додавання за модулем 2 (XOR), оскільки такі операції виконуються за одну одиницю процесорного часу та не потребують складних обчислень. Недоліком таких операцій є те, що вони обернені. Тобто, знаючи ключ шифрування можна з легкістю розшифрувати повідомлення застосувавши дану операцію до шифротексту. Для забезпечення надійності таких шифрів існує необхідність в постійній генерації випадкових чисел ключа шифрування. В роботі розглядається робота лінійного конгруентного методу для створення випадкових послідовностей чисел. Для початкового значення генератора наведено декілька джерел ентропії доступної на мікроконтролерах та пропонуються алгоритми збору початкових даних за таких джерел. В якості основного джерела пропонується використання шуму з непідключених контактів аналого-цифрового перетворювача, а в якості додаткового — неініціалізовану область оперативної пам’яті мікроконтролера. Реалізовано метод генерування випадкових послідовностей із вказаними джерелами ентропії та проведено оцінку функціонування алгоритму, а саме ключової характеристики — випадковості ключа шифрування. Для оцінки використовується набір тестів NIST STS 800-22. У всіх тестах алгоритм формування випадкової послідовності показав результат згідно якого можна підтвердити гіпотезу про те, що послідовність може вважатись випадковою.

Ключові слова: Інтернет речей; IoT; мережева безпека; пристрої з обмеженими обчислювальними ресурсами; алгоритми шифрування; ГПВЧ; джерело ентропії.

ВСТУП

У широкому спектрі нових галузей використовуються пристрої з обмеженими обчислювальними ресурсами, які взаємопов’язані та взаємодіють для виконання різноманітних задач. Інтернет речей (IoT), розподілені системи управління, транспортні системи, бездротові сенсорні мережі, телеметрія та смарт-мережа — це лише деякі приклади таких галузей. У будь-якому з цих контекстів безпека та конфіденційність є важливими аспектами. Одним із способів забезпечення захисту передачі даних є криптографія [1]. Оскільки частина пристроїв IoT мають обмежені ресурси, такі як енергія, пам’ять та частота процесору, то це зумовлює необхідність розробки алгоритмів шифрування які повинні споживати мінімум обчислювальних ресурсів для забезпечення достатнього рівня захисту даних.

Різні алгоритми шифрування вимагають певних випадкових даних для функціонування, особливо це стосується потокових шифрів, та шифрів побудованих за принципом одноразових блокнотів в яких до послідовності ключа ставляться особливі вимоги [2]:

- Мати випадковий рівномірний розподіл: $P_k(k) = 1 / 2^N$, де k - ключ, а N - кількість бінарних символів в ключі;
- Довжина ключа повинна бути такою ж як і довжина тексту;
- Ключ ніколи не повинен повторюватись;
- Ключ повинен зберігатись в секреті.

Виходячи з вищесказаного особливої уваги потребує метод генерації випадкового ключа шифрування в якому необхідно передбачити різні варіанти формування початкового заповнення, або «зерна» із доступних джерел ентропії.

Постановка проблеми. Випадковість відіграє важливу роль у багатьох областях криптографії. Комп'ютери не здатні генерувати справжні випадкові числа, це ж стосується і мікроконтролерів. Цим зумовлене використання так званих Генераторів Псевдовипадкових Чисел (ГПВЧ). ГПВЧ працюють, використовуючи односторонню математичну функцію, яка перетворює початкові дані в послідовність чисел з початкових даних, а потім ітеративно використовує цю функцію для генерації послідовності [3]. Однак важливо зазначити, що числа згенеровані ГПВЧ не є справді випадковими.

Якщо джерело насіння (англ. seed, тобто початкове значення функції) ГПВЧ слабке, то вся криптографічна система буде слабкою. Отож, ГПВЧ буде випадковим, тільки у випадку якщо його вихідні дані є дійсно випадковими, а результат лише функцією вихідних даних, причому ентропія результату ніколи не може перевищувати ентропію вхідних даних.

Генератори Справжніх Випадкових Чисел (ГВЧ) використовують недетерміновані джерела для створення випадкових чисел. Один із способів — використовувати фізичний процес, який неможливо передбачити. Джерелами ентропії в даному випадку можуть бути випадкові явища у природі, такі як: тепловий шум, видача випадкових електронів напівпровідником, або фонове випромінювання лічильника Гейгера, але це є дорогим та складним варіантом для застосування у промислових масштабах.

Це зумовлює необхідність пошуку джерел ентропії які повинні бути легко доступними для використання практично з будь-якими мікроконтролерами. Або ж повинні бути такими, щоб практично до будь-якого мікроконтролера можна було додати такі джерела з використанням мінімуму ресурсів.

Аналіз останніх досліджень і публікацій. Наукові дослідження та публікації, присвячені генерації псевдовипадкових чисел та їх тестуванню, стали об'єктом значного інтересу в області криптографії, інформаційної безпеки та комп'ютерних наук. Останні дослідження в цій області зосереджені на розробці нових алгоритмів генерації псевдовипадкових чисел, вдосконаленні існуючих методів генерації та тестування їх якості та безпеки, а також на пошуку нових підходів для створення більш надійних та ефективних генераторів та джерел ентропії.

В роботі [4] автори приводять та порівнюють різні означення випадкових та псевдовипадкових послідовностей. Розглядають питання щодо методик тестування послідовностей та генераторів.

У роботі [5] автори досліджують статистичні властивості даних отриманих від непідключеного Аналогово-Цифрового Перетворювача (АЦП) на мікроконтролері Arduino Duemilanove. Автори отримали результати які свідчать про те, що зчитане повне

значення за допомогою вбудованих функцій розміром 8 біт не проходить деякі статистичні тести, проте молодші біти зчитаних значень демонструють більш випадкову поведінку, та в режимі зчитування двох молодших біт проходять тести які не були пройдені в інших режимах збору випадкових значень, на більшості комп'ютерів до якого підключено мікроконтролер. Така поведінка може свідчити про те, що шуми в навколишньому середовищі є невеликими, тому їх вплив можна побачити тільки на молодших бітах зчитаних значень. Також варто зауважити, що автори не проводять повного тестування з використанням наборів тестів, а використовують певний перелік окремих тестів. Таким чином при використання такого джерела ентропії необхідно орієнтуватись саме на молодші біти, та розробити певний алгоритм накопичення.

В [6] авторами запропоновано підхід до використання мікросхеми ATmega328p для створення неклонованого ключа на основі внутрішньої структури чіпа оперативної пам'яті. Згенерований ключ успішно проходить різноманітні статистичні тести при різних умовах. Автори стверджують що отриманий ключ може бути використаний для аутентифікації у налаштуванні безпечного бездротового передавання даних між клієнтом та сервером або між вузлами для застосувань в Інтернеті речей та вбудованих системах з обмеженими ресурсами. В статті розглянуто, що існують комірки пам'яті які при ввімкненні мікроконтролера випадковим чином встановлюють свій стан в 0 або 1, окрім цього відбиток кожної такої мікросхеми є унікальним через особливості їх виробництва. Для взяття ентропії на відміну від генерування ідентифікатора необхідно саме зосередити увагу на тих комірках пам'яті які мають випадковий стан після подачі живлення.

Мета статті. Мета статті полягає у тестуванні алгоритму ГПВЧ на основі лінійного конгруентного методу, з декількома джерелами ентропії доступних на мікроконтролерах для формування початкового значення.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

В цій роботі в якості основного джерела ентропії пропонується до використання шум з непідключених контактів Аналогово-Цифрового Перетворювача (АЦП).

Вбудовані АЦП є на більшості мікроконтролерів, а у випадку відсутності такого модуля він може бути включений додатково. Основною перевагою є низька ціна таких модулів близько 1.5 доларів та сумісність з практично усіма мікроконтролерами, в порівнянні наприклад з тими само лічильниками Гейгера ціна яких близько 300 доларів. Окрім цього чим дешевший АЦП тим більші шуми в ньому будуть присутні, що підвищує випадковість «зерна».

Окрім цього необхідно передбачити використання додаткового джерела ентропії для забезпечення більшої вибірки та зменшення потенційного впливу зловмисника на систему з ціллю підміни початкового значення. В якості додаткового джерела ентропії, прийнято рішення використовувати неініціалізовану область оперативної пам'яті. Через особливості виробництва напівпровідників, частина регістрів оперативної пам'яті встановлюються випадковим чином при подачі живлення на пристрій. Ці значення можуть бути джерелом початкової ентропії.

Тестування розробленого алгоритму буде виконуватись на мікроконтролері ATmega328p. Для доступу до оперативної пам'яті, достатньо встановити вказівник на початок пам'яті та інкрементувати його в циклі до кінця області пам'яті. Проте необхідно зважати на специфіку платформи, пам'ять ATmega328P починається з 32 регістрів, потім

64 I/O регістрів, потім 160 розширених I/O регістрів, а потім 2048 байт оперативної пам'яті. Відповідно читання необхідно розпочати з 256 регістру [7]. Для того, щоб перемішати всі дані з регістрів пам'яті можна скористатись побітовою операцією XOR, оскільки деякі регістри встановлюються завжди в одне і те саме положення. Алгоритм такого рішення наведений на рис. 1.

Ентропія з неініціалізованої пам'яті повинна видобуватись на самому початку роботи системи, відразу після подачі живлення оскільки чим більше програмних об'єктів буде створено, тим менше неініціалізованої пам'яті залишиться.

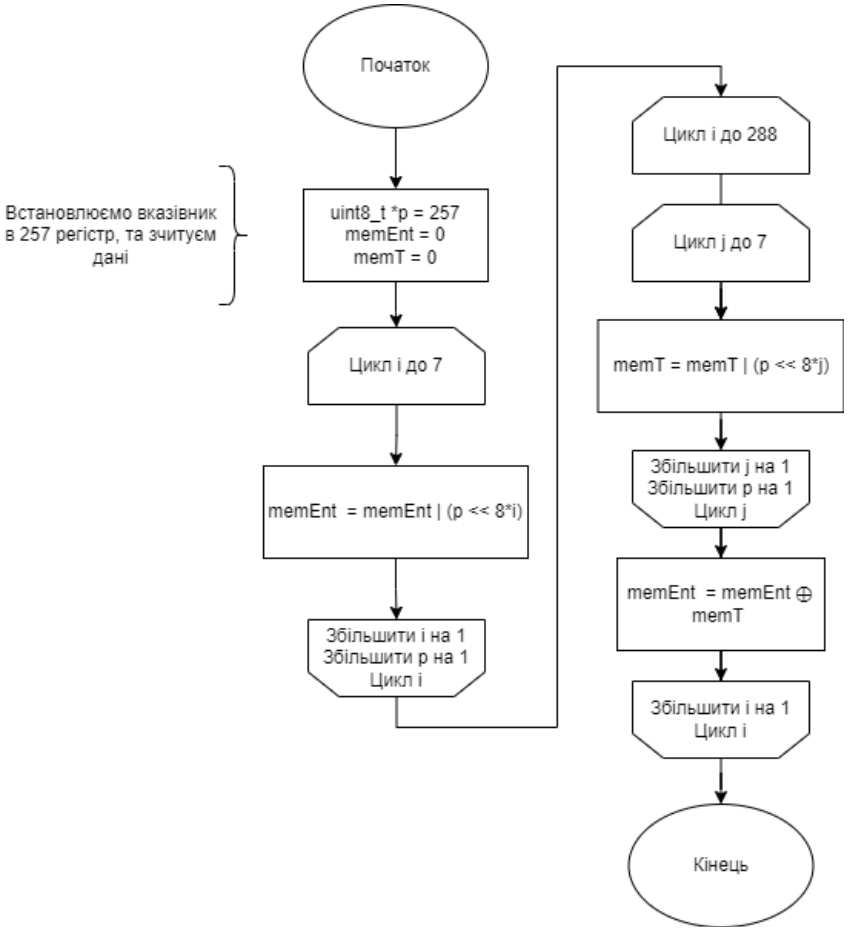

Рис.1. Алгоритм взяття ентропії з неініціалізованої частини пам'яті

До «зерна» неініціалізовної області оперативної пам'яті, прийнято рішення підмішати значення отримані з АЦП. Для цього складено алгоритм який зображено на рис. 2. Зважаючи на розроблені рекомендації для зменшення наслідків фізичної атаки на пристрій та не допущення злому процедури генерації ключа, варто додати додаткову перевірку отриманих з АЦП значень. Якщо згенерований ключ з АЦП дорівнює 0, його необхідно відхилити. Для запобігання таким атакам і використовується подвійне джерело ентропії. Якщо не вийшло отримати ключ з АЦП, для першого згенерованого зародку ГПВЧ повинен використовуватись ключ отриманий тільки з пам'яті, для подальших, він повинен бути об'єднаний операцією Xor з останнім згенерованим випадковим числом. Таким чином при фізичній атаці з використанням електромагнітного впливу, система зможе функціонувати, та забезпечувати певний необхідний рівень захисту.

ISSN 2663 - 4023

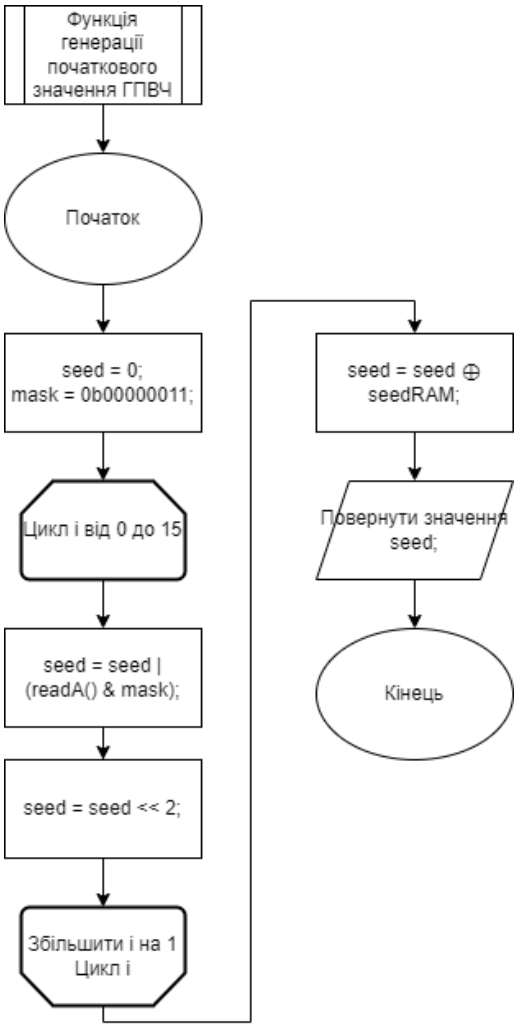

Рис. 2. Алгоритм формування «зерна» з АЦП

При зчитуванні шуму з АЦП найбільше шуму зазвичай містять перші два біти значення [4]. Тому пропонується зчитувати значення з АЦП не один раз, а декілька. При цьому брати з кожного зчитування всього 2 біти інформації та складувати їх. Для взяття конкретних біт з числа можна скористатись операцією побітового І, об'єднавши дані з деякою бітовою маскою. Для двох останніх біт маска буде виглядати так: 0b00000011. Самі ж біти можна додати до змінної за допомогою побітового АБО. Після додавання даних до пулу необхідно виконати бітовий зсув вліво на 2 позиці для запису наступних двох бітів. Цю операцію необхідно повторювати до заповнення пулу, розмір якого в даному прикладі 64 біти.

Після завершення генерації «зерна» зібраного з АЦП його можна поєднати з пулом зібраним з неініціалізованої оперативної пам'яті за допомогою операції XOR.

Для генерації випадкових значень було вирішено використовувати класичний лінійний конгруентний метод. Його суть полягає в обчислення послідовності випадкових чисел X_n, припускаючи

$$X_{n+1} = (aX_n + c) mod\ m,$$

де m — модуль (натуральне число, відносно якого обчислюється залишок від ділення, $m \geq 2$), a – множник $(0 \leq a \leq m)$, c – приріст $(0 \leq c < m)$, X_0 – початкове значення $(0 \leq X_0 < m)$.

ISSN 2663 - 4023

Період такої послідовності не може бути вищим за m [8]. Статистичні властивості послідовності створеної таким генератором дуже сильно залежать від обраних параметрів. Зазвичай обирають $m = 2^e$, оскільки при використанні в якості дільника ступінь двійки можна замінити операцію ділення застосувавши до числа побітове І з бітовою маскою 2^{e-1}, ця операція виконується в 17 разів швидше за ділення. Деякі параметри для роботи генератора були запозичені із роботи [9], де розглядаються деякі параметри котрі дають статистично випадкові послідовності. Блок-схема такого алгоритму наведена на рис. 3.

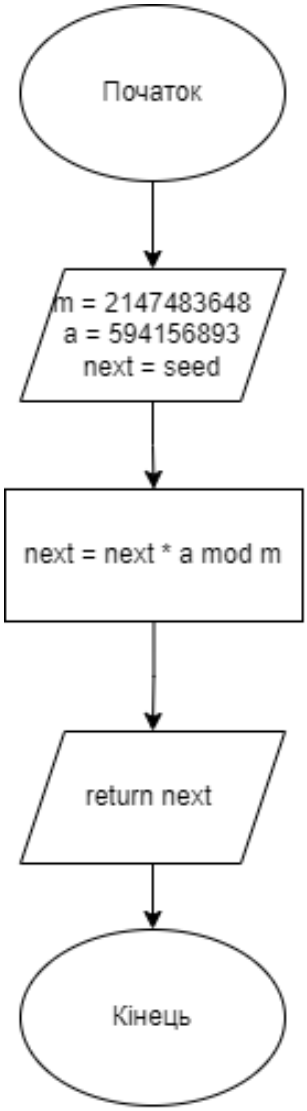

Рис. 3. Блок-схема реалізації лінійного конгруентного методу

Лінійні конгруентні методи не рекомендуються до використання в криптографії через те, що вони є передбачуваними. Але для передбачення результатів його виконання зловмиснику необхідно знати або початкове значення, або хоча б декілька вихідних значень. В пропонованому методі зловмисник не може отримати доступу до таких значень, через те що початкове пересилається в зашифрованому вигляді, а вихідні

значення взагалі не пересилаються, а використовуються виключно як ключ для шифрування та розшифрування даних.

Генерування випадкових чисел є складним завданням, так само як і оцінка якості згенерованих даних. На практиці оцінка випадковості значною мірою покладається на емпіричні тести випадковості [4].

Зважаючи на таке було вирішено провести оцінку реалізованого генератора в частині генерації випадкових послідовностей. Більшість емпіричних тестів випадковості засновані на перевірці статистичних гіпотез. Кожен тест порівнює певні характеристики даних (частота одиниць, частота m-бітових блоків тощо) з очікуваною тестовою статистикою (0.5, 2-m і т.д.), яка попередньо обчислюється для випадкових нескінченних послідовностей. У цьому контексті випадковість є імовірнісною властивістю і може бути охарактеризована та описана в термінах ймовірності. Це пов'язано з тим, що навіть хороший генератор випадкових чисел видає послідовності (наприклад послідовність всіх одиниць) з характеристиками, що значно відрізняються від значень очікуваних у тестах. Тому не можливо відрізнити, чи дана послідовність з «поганими» характеристиками була згенерована дефектним генератором, чи послідовність була випадково згенерована хорошим генератором. В контексті емпіричних тестів випадковості, вона описується як ймовірність того, що ідеальний генератор випадкових чисел згенерує послідовності з такою самою або меншою якістю випадковості, ніж та, що була продемонстрована в аналізованій послідовності.

Для перевірки ГПВЧ можуть використовуватись різні статистичні тести. Тести часто групуються в набори тестів, також відомі як батареї, щоб надати більш всебічне дослідження випадковості. Існує три загальноприйняті набори тестів для аналізу випадковості: Набір статистичних тестів Statistical Test Suite (STS) від National Institute of Standards and Technology (NIST) [10], Dieharder[11] і TestU01[12].

NIST STS має особливе значення, оскільки він був опублікований як стандарт NIST і використовується для офіційної сертифікації. Тобто його можна вважати певним стандартом галузі. Окрім цього документація NIST STS дає чіткі вказівки щодо інтерпретації результатів тестів, але все ж таки інтерпретація інколи використовує лише наближені значення.

Результати статистичних тестів випадковості зазвичай представлені у вигляді значення p, яке представляє ймовірність того, що ідеальний генератор випадкових чисел створить менш випадкову послідовність, ніж послідовність, яка тестується.

Хоча значення p для єдиного тесту на випадковість має чітку статистичну інтерпретацію, інтерпретація результатів наборів тестів є більш проблемною. Це відбувається тому, що емпіричні тести на випадковість і їхні результати зазвичай залежать один від одного і корелюються.

Наприклад, якщо частоти одиниць і нулів упереджені (нерівні) для заданої послідовності, ймовірно, що частоти блоків з двох бітів теж упереджені. Щоб надати чітку статистичну інтерпретацію результатів набору тестів, потрібно проаналізувати залежність/кореляцію між результатами тестів, які застосовуються до випадкових даних.

Набір STS складається з 15 статистичних тестів, кожен такий тест здійснює перевірку бінарної послідовності на ознаки відхилення від випадковості. Кожен тест сформульований для оцінки нульової гіпотези, а саме того, що послідовність, яка тестується, є випадковою. Статистична ознака тесту є функцією протестованих даних, яка стискає виміряну якість випадковості в єдине значення, спостережувану статистичну ознаку.

ISSN 2663 - 4023

Щоб оцінити тест, розподіл статистичної ознаки має бути відомим за нульовою гіпотезою (коли очікується, що дані будуть випадковими). Більшість тестів з набору NIST мають розподіл x^2 або нормальний як свій референтний розподіл. Спостережувана статистична ознака зазвичай перетворюється на p-значення за допомогою референтного розподілу, оскільки p-значення можна легше інтерпретувати. Це значення представляє ймовірність того, що ідеальний генератор випадкових чисел створить послідовність менш випадковою, ніж послідовність, що аналізується.

Найважливішою властивістю p-значення є те, що для довільних статистичних тестів (і не лише для тестів на випадковість), які задовольняють нульову гіпотезу, значення p рівномірно розподілені на інтервалі (0, 1). Це означає, що випадкові послідовності, оброблені довільним емпіричним тестом, повинні рівномірно розподілятися на (0, 1). Тому ймовірність того, що p, обчислені для випадкової послідовності, лежать в інтервалі [a, b], можна висловити як: $P(a \leq p \leq b) = b - a$.

Рівень значущості α є імовірністю того, що послідовність буде сприйнята як невипадкова. Якщо отримане значення більше або дорівнює α, це означає що текст пройдено, і гіпотеза про те, що послідовність є випадковою може бути підтверджена. Рівень значущості рекомендований NIST $a = 0.01$ [10]. Такий рівень значущості і був відібраний для тестування послідовності.

Кожен тест STS NIST визначається статистичним показником одного з наступних трьох типів і досліджує випадковість послідовності відповідно до:

1. біти — ці тести аналізують різні характеристики бітів, такі як пропорція бітів, частота зміни бітів і накопичувальні суми;
2. блоки m-біт — ці тести аналізують розподіл m-бітних блоків (m зазвичай менше 30 біт) у послідовності або її частинах;
3. групи M-біт — ці тести аналізують складні властивості M-бітних (M зазвичай більше 1000 біт) частин послідовності, такі як ранг послідовності, побачений як матриця, спектр послідовності або лінійна складність потоку бітів.

Всі тести параметризовані n, що позначає довжину бінарної послідовності, яка підлягає тестуванню. Деякі тести також параметризовані другим параметром, позначеним m або M. Оскільки еталонні розподіли статистичних ознак тестів STS NIST наближені асимптотичними розподілами, то тести дають точні результати (p-значущі) лише для певних значень їх параметрів. Значення параметрів для кожного конкретного тесту брались відповідно до рекомендацій NIST [10].

Деякі з тестів STS NIST виконуються в декількох варіантах, тобто вони виконують кілька підтестів і досліджують більше властивостей послідовності того ж типу. Наприклад, тест накопичувальної суми досліджує послідовність відповідно вперед і назад накопичувальної суми.

Щоб застосувати всі тести, параметр n (довжина біт послідовностей) повинен бути більше 100000. Документація NIST STS рекомендує, щоб принаймні $k = \alpha - 1 = 100$ послідовностей були протестовані. Це також є відповідним значенням для тесту однорідності p-значень (принаймні 55 послідовностей повинні бути оброблені). Оскільки p-значення обробляються за допомогою деякої апроксимації, чим більше послідовностей буде протестовано, тим більш точні результати будуть отримані.

Проте варто зауважити, що деякі тести для отримання коректного результату вимагають мінімум $n = 10^6$ бітів для тестування.

Для формування «зерна» реалізованого алгоритму було відібрано популярний мікроконтролер ATmega328 з такими характеристиками:

- Тактова частота — 16 МГц;

- Flash-пам’ять — 32 Кб;
- SRAM — 2 Кб;
- EEPROM — 1 Кб.

Проте мікроконтролер використовувався лише для формування зерна, послідовність генерувалась на стаціонарному комп’ютері оскільки формування вибірки достатньої для тестів зайняло б дуже багато часу якби вона повністю формувалась на мікроконтролері. Проте самі значення, що генеруються є однаковими на будь-якому пристрої за умови однакових параметрів генератору та однакового «зерна».

Для проходження тестів було згенеровано m = 1000 послідовностей біт по n = 10^6 кожна, тобто загальний розмір набору даних для тесту $n > 10^9$. Початкове значення змінювалось після досягнення періоду. Значення отримані ГПВЧ приводились до діапазону (0;1) шляхом ділення результату на дільник m та перетворювались в 1 та 0 за правилом: якщо $x_n > 0.5$, до послідовності записується 1, інакше 0.

До всіх послідовностей та тестів був застосований рівень значущості $a = 0.01$, а всі інші параметри необхідні для кожного тесту вказані в табл. 1.

Під час оцінки якості ГПВЧ були використані рекомендації щодо інтерпретації результатів, з офіційної документації [10]. В документі пропонується дві стратегії ухвалення рішення, щодо проходження тестів на випадковість.

Згідно ***першої стратегії*** необхідно визначити частку послідовностей $P1$, які пройшли перевірку, тобто $p > a$, та порівняти її з нижньою межею довірчого інтервалу $P1_{THR}$.

$$P1 = \frac{\sum_{i=1}^{1000}(P\ value(i) \geq a)}{m},\ P1_{THR} = (1-a) \pm 3\sqrt{\frac{(1-a)a}{m}} = 0.9805607$$

Якщо для одного з 15 тестів значення $P1$ виходить за ці межі, вважається, що тест не пройдено.

Таблиця 1

Тестування ГПВЧ згідно з першою стратегією

Тест №	Назва тесту і його параметри	P1
1	Частотний монобітний	0.983
2	Частотний блочний (M=128)	0.986
3	Серій	0.986
4	Довгих серій одиниць (M=10000)	0.989
5	Рангу випадкової {0,1}-матриці	0.993
6	Дискретного перетворення Фур’є	0.987
7…154	Відповідності аперіодичних шаблонів, що не перекриваються (M=9, 148 шаблонів)	0.987 (mean)
155	Відповідності періодичних шаблонів, що перекриваються (M=9)	0.990
156	Лінійної складності (M=500)	0.992
157	Універсальний статистичний — Маурера (L=8, Q=2356)	0.989
158	Послідовності (M=16, $\nabla\psi^2{}_m\ (obs)$)	0.985
158	Послідовності (M=16, $\nabla^2\psi^2{}_m\ (obs)$)	0.990
160	Наближеної ентропії (M=10)	0.986
161	Накопичених сум (Прямий)	0.991
162	Накопичених сум (Зворотній)	0.993
163…170	Випадкових відхилень (x = –4,…,–1, 1,…,4)	0.991
171…188	Вигляду випадкових відхилень (x = –9,…,–1, 1,…,9)	0.990

Згідно **другої стратегії** розподіл P для кожного тесту повинен бути рівномірним на інтервалі [0,1].

$$x^2 = \sum_{i=1}^{10} \frac{(Ci - m/10)^2}{m/10},\ P2 = P(x^2) = igamc(\frac{9}{2}, x^2/2).$$

Якщо отримане значення в результаті тестування $P1 < 0.0001$, то вважається, що ГПВЧ тест не пройшов.

Для перевірки цієї стратегії значення P були розбиті на 10 підінтервалів C1-C1, з кроком 0.1. Результати наведені у табл. 2.

Таблиця 2

Тестування ГПВЧ згідно другої стратегії

Тест №	Назва тесту і його параметри	P2
1	Частотний монобітний	0.534146
2	Частотний блочний (M=128)	0.911413
3	Серій	0.573482
4	Довгих серій одиниць (M=10000)	0.441284
5	Рангу випадкової {0,1}-матриці	0.523526
6	Дискретного перетворення Фур'є	0.478839
7…154	Відповідності аперіодичних шаблонів, що не перекриваються (M=9, 148 шаблонів)	0.466079
155	Відповідності періодичних шаблонів, що перекриваються (M=9)	0.514791
156	Лінійної складності (M=500)	0.931185
157	Універсальний статистичний — Маурера (L=8, Q=2356)	0.213309
158	Послідовності (M=16, $\nabla\psi^2{}_m\,(obs)$)	0.979788
158	Послідовності (M=16, $\nabla^2\psi^2{}_m\,(obs)$)	0.735908
160	Наближеної ентропії (M=10)	0.350485
161	Накопичених сум (Прямий)	0.739918
162	Накопичених сум (Зворотній)	0.350485
163…170	Випадкових відхилень (x = –4,…,–1, 1,…,4)	0.448575
171…188	Вигляду випадкових відхилень (x = –9,…,–1, 1,…,9)	0.463232

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Розроблено алгоритм формування зародку для ГПВЧ з використанням фізичних величин — аналогового шуму та неініціалізованого простору оперативної пам'яті.

Згідно проведеному тестуванню, ГПВЧ з таким алгоритмом формування «зерна» пройшов всі статистичні тести. Таким чином підтверджена нульова гіпотеза, і можна вважати що згенерована послідовність може вважатись випадковою. Отже можна стверджувати, що реалізований алгоритм генерації випадкової послідовності ключа, може застосовуватись в розроблюваному методі шифрування повідомлень на пристроях з обмеженими обчислювальними ресурсами. Проте при розробці методу шифрування необхідно враховувати недоліки генератора, а саме початкове значення повинно знаходитись в секреті та жодне вихідне значення не повинно бути розкрито.

ISSN 2663 - 4023

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горбенко, І., Гулак, Г., Олійников, Р., Руженцев, В., & Михайленко, М., (2005). Аналіз властивостей алгоритмів блокового симетричного шифрування (за результатами міжнародного проекту NESSIE). *Міжнародна науково-практична конференція «Безпека інформації в інформаційно-телекомунікаційних системах». Тези доповідей,* 17–18.
2. Lugrin, T. (2023). One-Time Pad. *Trends in Data Protection and Encryption Technologies,* 3–6. https://doi.org/10.1007/978-3-031-33386-6_1
3. Katagi, M., & Moriai, S. (2012). Lightweight Cryptography for the Internet of Things. *Sony Corporation.*
4. Гулак, Г., & Ковальчук, Л. (2001). Різні підходи до визначення випадкових послідовностей. *Науково-технічний збірник «Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні», 3,* 127–133.
5. Kristinsson, B. (2011). Ardrand: The Arduino as a Hardware Random-Number Generator.
6. Elmestekawi, A., Tammam, A., & Issa, H. (2018). Unclonable key Generator Based on Chip signature and SRAM-PUF of ATmega328P chip. *2018 28th International Conference on Computer Theory and Applications (ICCTA),* 24–29. https://doi.org/10.1109/ICCTA45985.2018.9499169
7. Bagur, J., & Chung, T. (б. д.). *Arduino Memory Guide|Arduino Documentation*. Arduino Docs|Arduino Documentation. https://docs.arduino.cc/learn/programming/memory-guide.
8. Knuth, D. (1997). The art of computer programming (3rd ed.). *Addison Wesley*.
9. L'Ecuyer, P. (1999). Tables of linear congruential generators of different sizes and good lattice structure. *Math. Comput., 68,* 249–260. http://doi.org/10.1090/S0025-5718-99-00996-5
10. Rukhin, A., et al. (2010). NIST SP 800-22 Rev1. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. National Institute of Standards and Technology Special Publication 800-22 Rev1. https://doi.org/10.6028/NIST.SP.800-22r1a
11. Brown, R. (2004). Dieharder: A Random Number Test Suite, Version 3.31.1. *Webhome*. https://webhome.phy.duke.edu/~rgb/General/dieharder.php
12. L'Ecuyer, P., & Simard, R. (2007). TestU01: A C library for empirical testing of random number generators, *ACM Trans. Math. Softw., 33(4),* 1–40. https://doi.org/10.1145/1268776.1268777

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (22), 2023
ISSN 2663 - 4023

R. Chernenko
graduate student of the Department of Information and Cyber Security
Borys Grinchenko Kyiv University, Kyiv, Ukraine
ORCID 0000-0002-1439-961X
r.chernenko.asp@kubg.edu.ua

GENERATION OF PSEUDORANDOM SEQUENCES ON MICROCONTROLLERS WITH LIMITED COMPUTATIONAL RESOURCES, ENTROPY SOURCES, AND STATISTICAL PROPERTIES TESTING

Abstract. Traditional encryption algorithms cannot be implemented on Internet of Things (IoT) devices due to their constrained computational resources. This necessitates the search and development of cryptographic solutions for securing data processed and transmitted by such devices. When encrypting data on devices with limited computational resources, simple encryption algorithms based on elementary bitwise operations, such as bitwise modulo-2 addition (XOR), can be utilized since these operations execute in a single processor cycle and do not require complex computations. However, a drawback of such operations is their invertibility—knowing the encryption key enables easy decryption by applying the same operation to the ciphertext. Ensuring the reliability of such ciphers requires continuous generation of random encryption keys.This work explores the functionality of the linear congruential method for generating sequences of random numbers. Several entropy sources available on microcontrollers are presented for the initial generator value, along with proposed algorithms for collecting initial data from these sources. The use of noise from unconnected pins of the analog-to-digital converter is suggested as the primary entropy source, while the uninitialized area of the microcontroller's random-access memory serves as an additional source. A method for generating random sequences using the specified entropy sources is implemented and the algorithm's performance is evaluated, specifically the key characteristic—randomness of the encryption key. The NIST STS 800-22 test suite is employed for evaluation. In all tests, the random sequence generation algorithm demonstrated results confirming the hypothesis that the sequence can be considered random.

Keywords: Internet of Things; IoT; network security; constrained devices; encryption algorithms; PRNG; entropy source.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Horbenko, I., et al. (2005). Analysis properties of block symmetric encryption algorithms (according to the results of international of the NESSIE project). *International Scientific and Practical Conference "Safety of information in information and telecommunication systems". Abstracts of reports,* 17–18.
2. Lugrin, T. (2023). One-Time Pad. *Trends in Data Protection and Encryption Technologies,* 3–6. https://doi.org/10.1007/978-3-031-33386-6_1
3. Katagi, M., & Moriai, S. (2012). Lightweight Cryptography for the Internet of Things. *Sony Corporation.*
4. Hulak, H., & Kovalchuk, L. (2001). Different Approaches to Defining Random Sequences. *Scientific and Technical Collection "Legal, Regulatory, and Metrological Support for Information Security Systems in Ukraine",* 3, 127–133.
5. Kristinsson, B. (2011). Ardrand: The Arduino as a Hardware Random-Number Generator.
6. Elmestekawi, A., Tammam, A., & Issa, H. (2018). Unclonable key Generator Based on Chip signature and SRAM-PUF of ATmega328P chip. *2018 28th International Conference on Computer Theory and Applications (ICCTA)*, 24–29. https://doi.org/10.1109/ICCTA45985.2018.9499169
7. Bagur, J., & Chung, T. (б. д.). *Arduino Memory Guide|Arduino Documentation*. Arduino Docs|Arduino Documentation. https://docs.arduino.cc/learn/programming/memory-guide.
8. Knuth, D. (1997). The art of computer programming (3rd ed.). *Addison Wesley*.
9. L'Ecuyer, P. (1999). Tables of linear congruential generators of different sizes and good lattice structure. *Math. Comput., 68*, 249–260. http://doi.org/10.1090/S0025-5718-99-00996-5
10. Rukhin, A., et al. (2010). NIST SP 800-22 Rev1. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. National Institute of Standards and Technology Special Publication 800-22 Rev1. https://doi.org/10.6028/NIST.SP.800-22r1a

11. Brown, R. (2004). Dieharder: A Random Number Test Suite, Version 3.31.1. *Webhome*. https://webhome.phy.duke.edu/~rgb/General/dieharder.php
12. L'Ecuyer, P., & Simard, R. (2007). TestU01: A C library for empirical testing of random number generators, *ACM Trans. Math. Softw., 33(4),* 1–40. https://doi.org/10.1145/1268776.1268777