Галузь знань _____12 – Інформаційні технології_____

Спеціальність _____123 – Комп’ютерна інженерія _____

НАУКОВА РОБОТА

СПОСІБ ГЕНЕРАЦІЇ ВИПАДКОВИХ ЧИСЕЛ НА ПЛАТФОРМІ ANDROID

Шифр Блокчейн

2020

Анотація
Наукової роботи під шифром: **Блокчейн**

Актуальність роботи. У сучасному світі майже всі володіють смартфоном і для більшості людей ця технологія стала важливою частиною їхнього життя. Смартфони мають велику кількість переваг такі як: робота в мережі мобільного зв’язку, підтримка WI-FI, Bluetooth, завдяки чому користувачі мають доступ до мережі Інтернет для завантаження та запуску сторонніх додатків. Більшість смартфонів мають вбудовані датчики: GPS, гіроскоп, акселерометр, камера, мікрофон та датчик освітленості. Для пристрою з таким числом користувачів у всьому світі важливо, щоб він був належним чином захищений. Зростаюча популярність смартфонів роблять смартфони постійною ціллю атак хакерів. Багато користувачів смартфонів зберігають особисті дані на своїх телефонах, тому проблеми з безпекою на такій платформі, як Android, мають суттєвий вплив на суспільство та економіку.

Нажаль смартфони на операційній системі Anroid мають проблеми з безпекою даних. Зловмисники можуть перехоплювати або змінювати з'єднання зі смартфоном. Протягом багатьох років досить багато криптографічних систем були знищенні атаками на їх генератори випадкових чисел (ГВЧ). Це серйозно впливає на безпеку і конфіденційність користувача смартфона.

Метою роботи є розробка способу генерації випадкових значень на основі потужного джерела випадковості. Даний спосіб може бути використаний на телефоні поруч з існуючими джерелами генерації

Задачі досліджень у роботі формулюються наступним чином:

Визначення датчиків або комбінації датчиків пристроїв підходять для отримання високої ентропійної випадковості.

Визначення способів отримання високої ентропійної випадковості з даних, отриманих з датчиків

Розробка способу отримання випадкових чисел на основі даних з датчиків

Розробка алгоритму для генерування випадкових чисел на основі пулу ентропії

Оцінка якості випадкових даних, отриманих з датчиків пристроїв для використання в середовищі користувача.

Об'єктом дослідження є способи та алгоритми генерації випадкових значень, які використовують датчики пристрою Android для ефективного отримання високої ентропії випадковості в криптографічних цілях.

Предметом дослідження є способи і алгоритми генерації випадкових значень на основі використання датчиків пристрою Android.

Методи досліджень базуються на основних положеннях теорії захисту інформації, теорії ймовірності і математичної статистики, системного і структурного

аналізу, генерації випадкових даних, аналізу якості ентропії і математичної статистики.

Наукова новизна. Розроблений спосіб для генерування випадкових значень на основі отримання пулу ентропії з датчиків Андроід-пристроїв, яка надсилається на вхід хеш-функції SHA-256. 256-бітовий вихід SHA-256 використовується як ключ для алгоритму Xsalsa20

Практична цінність отриманих результатів полягає::

1. Розроблений алгоритм для генерування пулу ентропії і створено додаткове джерело випадкових значень для використання на Android девайсах поруч з стандартним /dev/random.

2. Був розроблений додаток для Android пристроїв, який використовує . розроблений не блокуючий алгоритм генерування випадкових значень.

3. Проведений аналіз продуктивності роботи розробленого алгоритму.

Об’єм роботи складає 37 сторінок. Робота містить 17 рисунків, 1 таблицю, 34 бібліографічних джерела.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі майже всі володіють смартфоном і для більшості людей ця технологія стала важливою частиною їхнього життя. Смартфони мають велику кількість переваг такі як: робота в мережі мобільного зв’язку, підтримка WI-FI, Bluetooth, завдяки чому користувачі мають доступ до мережі Інтернет для завантаження та запуску сторонніх додатків. Більшість смартфонів мають вбудовані датчики: GPS, гіроскоп, акселерометр, камера, мікрофон та датчик освітленості. Для пристрою з таким числом користувачів у всьому світі важливо, щоб він був належним чином захищений. Зростаюча популярність смартфонів роблять смартфони постійною ціллю атак хакерів. Багато користувачів смартфонів зберігають особисті дані на своїх телефонах, тому проблеми з безпекою на такій платформі, як Android, мають суттєвий вплив на суспільство та економіку.

Нажаль смартфони на операційній системі Anroid мають проблеми з безпекою даних. Зловмисники можуть перехоплювати або змінювати з'єднання зі смартфоном. Протягом багатьох років досить багато криптографічних систем були знищенні атаками на їх генератори випадкових чисел (ГВЧ). Це серйозно впливає на безпеку і конфіденційність користувача смартфона.

Метою роботи є розробка способу генерації випадкових значень на основі потужного джерела випадковості. Даний спосіб може бути використаний на телефоні поруч з існуючими джерелами генерації

Задачі досліджень у роботі формулюються наступним чином:

1. Визначення датчиків або комбінації датчиків пристроїв підходять для отримання високої ентропійної випадковості.
2. Визначення способів отримання високої ентропійної випадковості з даних, отриманих з датчиків
3. Розробка способу отримання випадкових чисел на основі даних з датчиків
4. Розробка алгоритму для генерування випадкових чисел на основі пулу ентропії
5. Оцінка якості випадкових даних, отриманих з датчиків пристроїв для використання в середовищі користувача.

Об'єктом дослідження є способи та алгоритми генерації випадкових значень, які використовують датчики пристрою Android для ефективного отримання високої ентропії випадковості в криптографічних цілях.

Предметом дослідження є способи і алгоритми генерації випадкових значень на основі використання датчиків пристрою Android.

Методи досліджень базуються на основних положеннях теорії захисту інформації, теорії ймовірності і математичної статистики, системного і структурного аналізу, генерації випадкових даних, аналізу якості ентропії і математичної статистики.

Наукова новизна. Розроблений спосіб для генерування випадкових значень на основі отримання пулу ентропії з датчиків Андроід-пристроїв, яка надсилається на вхід хеш-функції SHA-256. 256-бітовий вихід SHA-256 використовується як ключ для алгоритму Xsalsa20

Практична цінність отриманих результатів полягає::

4. Розроблений алгоритм для генерування пулу ентропії і створено додаткове джерело випадкових значень для використання на Android девайсах поруч з стандартним /dev/random.

5. Був розроблений додаток для Android пристроїв, який використовує . розроблений не блокуючий алгоритм генерування випадкових значень.

6. Проведений аналіз продуктивності роботи розробленого алгоритму.

1 Аналіз особливостей генераторів випадкових чисел на платформі ANDROID

Ентропія визначаєтья як міра ступеня невизначеності випадкової величини. Якщо ξ- дискретна випадкова величина, задана на ймовірному просторі (Ω, F, P) і може приймати значення x_1, x_2, …, з розподілом імовірності $\{p_k : 1,2, \dots\}$, $p_k = P\{\xi = x_k\}$, то ентропія визначається за формулою:

$$H(\xi) = -\sum_{k=1}^{\infty} p_k \log p_k$$

Ентропія Шеннона в формулі забезпечує середню величину ентропії для незалежного розподілу випадкових величин.

Згідно з роботою [1] мірою ентропії для використання є менша ентропія, також відома як ентропія Реніі. Міні-ентропія визначається наступним чином:

$$H_\infty(X) = min_{x \in X}(-\log P_x(x)) = -\log\left(max_{x \in X} P_x(x)\right)$$

Мінімальна ентропійна міра ентропії рекомендується замість ентропії Шеннона, оскільки мінімальна ентропія розглядає найгірший випадок замість середнього випадку і є більш обмеженим випадком і кращої метрикою для вимірювання ентропії в криптографії.

Захищеність криптографічних систем залежить від секретних даних, які відомі авторизованим користувачам, але невідомі і непередбачувані для інших. Тобто сучасні криптографічні системи вимагають частої генерації випадкових значень. Для цього використовують генератори випадкових чисел (ГВЧ), які використовують джерела ентропії для генерації випадкових бітів даних.

Криптографічні атаки, які порушують роботу або використовують слабкі сторони ГВЧ, називаються атаками генератора випадкових чисел. Створення високоякісного ГВЧ завжди потрібне для забезпечення безпеки, а відсутність якісного ГВЧ призводить до вразливості до атак. Протягом останніх років було багато прикладів атак на ГВЧ криптографічних систем. Наслідки цих атак варюються між незначним витоком даних і серйозними проблемам для безпеки

Можна виділити два типи ГВЧ - генератори справжніх випадкових чисел і генератори псевдовипадкових чисел.

Генератори псевдовипадкових чисел (ГПВЧ) використовують односторонні функції для генерації ентропії. Ці алгоритми потребують введення ключа на основі джерела ентропії для створення довгих випадкових послідовностей даних.

Є багато прикладів ГПВЧ, такі як Fortuna, алгоритм Yarrow та конструкції, які використовують шифри, такі як AES-CTR, XSalsa20, ChaCha20. Перевага ГПВЧ полягає в тому, що вони можуть генерувати практично нескінченний потік випадкових

даних після того, як вони правильно налаштовані і вони не стикаються з проблемою блокування процесу оновлення ентропії. Недоліком є те, що використаний ключ в ГПВЧ є основною вразливістю і всі згенеровані дані стають передбачуваними.

Генератори справжніх випадкових чисел використовують джерело природної ентропії для генерації випадковості (наприклад, фоновий шум, атмосферний шум, шум електромагнітного апарату або космічне випромінювання).

Природна ентропія, як правило, дуже непередбачувана, недетермінована і важко вимірювана зовнішнім спостерігачем. Недоліком природної ентропії є те, що ГВЧ необхідно регулярно оновлювати свою ентропію, щоб залишатися непередбачуваним. Під час збирання додаткової ентропії ГВЧ блокується, доки не буде достатньо ентропії для задоволення параметрів.

В операційній системі Linux спеціальний символьний псевдопристрій /dev/random є прикладом цього процесу, він блокується, поки не накопичиться достатня ентропія. При читанні даних у пристрої /dev/random створюються тільки випадкові байти, що складаються з бітів шуму «хаотичного» пулу. У ядрі Linux «хаотичний» пул отримує ентропію з декількох джерел, в тому числі з апаратного генератора випадкових чисел процесорів Intel.

Пристрій /dev/random необхідний користувачам, наприклад, при створенні ключа шифрування, який передбачає тривале використання.

Якщо «хаотичний» пул спорожнів, то читання /dev/random блокується, поки необхідну кількість бітів в пулі не буде створено. Ця блокувальна поведінка може суттєво сповільнити роботу системи, коли виконуються дуже великі запити на випадковість.

Оцінка ентропії є життєво важливою частиною в побудові генератора псевдовипадкових чисел (ГПВЧ), тому що можливість дати точну оцінку кількості ентропії, що міститься в пулі ентропії, необхідна для досягнення певного рівня безпеки. Якщо точність оцінки ентропії ГПВЧ висока, це може дати кращі гарантії безпеки про непередбачуваність пулу ентропії. Це робить менш імовірним, що зловмисник зможе скомпрометувати випадковість системи ГПВЧ.

Однак важко дати хорошу оцінку ентропії, тому що в певному сенсі це означає, що потрібно "передбачити" непередбачуваність пулу ентропії. Було проведено багато досліджень на тему оцінки ентропії, але золоте рішення для повністю достовірної оцінки ентропії поки не знайдено.

У [14] доводиться, що складність оцінки мінімальної ентропії розподілу є класом складності, який вважається рівним NP-повній складності. Таким чином, проблема доведення того, що пул ентропії дійсно випадковий, обчислювально складний, тому замість цього оцінка повинна бути зроблена з використанням непрямих заходів.

На пристроях Android пул ентропії оновлюється кожен раз при перезавантаженні телефону, що часто достаньо для звичайного користувача. Крім того, пул ентропії може бути налаштований на оновлення після певних інтервалів, якщо компрометуючі атаки представляють серйозну загрозу. Реалізація рішень, запропонованих в документі, на пристроях Android знизить ефективність ГПВЧ на платформі, яка має справу з обмеженими ресурсами, що надаються пристроєм. Беручи до уваги ці моменти, було б недоцільно впроваджувати ці рішення в даний час.

Замість цього слід знайти рішення для забезпечення додаткової ентропії на пристроях Android, щоб внутрішній стан був набагато менш схильний до компрометуючих атак через генерацію з низькою ентропією в першу чергу. ГВЧ повинен постійно надавати користувачеві сильні випадкові початкові значення для використання криптографічних протоколів на пристрої, залишаючись при цьому максимально ефективним, використовуючи обмежені ресурси, пропоновані пристроєм.

Поточний стандарт генерації випадковостей на пристроях Android - /dev / random Linux kernel ГПВЧ. /dev / random був спочатку реалізований для Linux в 1994 році. /Dev / random ГПВЧ генерує пул ентропії з декількох джерел на апаратному рівні, таких як таймінги між синхронізацією клавіатури і синхронізація між перериваннями. Передбачається, що ці джерела ентропії недетерміновані і їх важко виміряти сторонньому спостерігачеві. Як тільки достатня випадковість буде змішана з пулом ентропії /dev/random, він може приймати запити на випадкові числа і надавати їх, приймаючи хеш SHA вмісту пулу ентропії.

Ядро Linux також надає другий ГПВЧ / dev / urandom. /dev /urandom ідентичний /dev /random за своєю функціональністю, з тією лише різницею, що /dev / urandom не блокується і не має обмежень на кількість запитів на випадкові числа, які він може прийняти. Оскільки вважається, що обчислювально неможливо отримати будь-яку корисну інформацію про вхід хеша SHA з його виведення, /dev/urandom як і раніше гарантує криптографічно сильну випадковість. Цього достатньо для багатьох додатків, але для додатків, що вимагають більш високої гарантії випадковості, рекомендується використовувати /dev/random. Причина цього в тому, що /dev / random поверне тільки максимальну кількість бітів випадковості, засноване на оцінці ентропії пулу ентропії, після того, як ця сума буде досягнута, через це ГПВЧ буде блокувати пристрій, поки він не поповнить свій пул ентропії.

/Dev / random ГПВЧ широко використовується більшістю додатків і зазвичай вважається безпечним. Тим не менш, /dev/random був підданий критиці через велику кількість досліджень, які стверджують, що він має вразливості. Навіть вихідний код /dev/random вказує на слабкість передбачуваності при запуску системи. При запуску системи послідовність дій передбачувана зловмисником, так як за цей час практично не відбувається взаємодії з користувачем. Це може призвести до непередбачуваності

бітів в пулі ентропії, що дозволить опуститися нижче мінімального порогу ентропії і створити вразливість.

Згідно [10] /dev / random містить вразливості в оцінці ентропії і функції внутрішнього змішування. Їх доказ вразливості в оцінці ентропії складається з двох способів обдурити оцінювача. По-перше, вони показують, що можна визначити розподіл нульової ентропії, який оцінювач оцінить високою ентропією. По-друге, вони показують, що можна визначити розподіл довільної високої ентропії, який оцінювач оцінить нульовою ентропією.

Подібним же чином вони демонструють, що можна визначити розподіл довільної високої ентропії, для якої функція змішування не збільшує ентропію внутрішнього стану. Ці атаки на Linux ГПВЧ показують, що вони не відповідають поняттю "надійності" безпеки, але все ще залишається не зрозумілим, чи призводять ці атаки до фактичних вразливостей, які можна використовувати на практиці.

Було висунуто ряд пропозицій по альтернативах ГПВЧ Linux. Однією з таких альтернатив є демон збору ентропії [17], який є демоном простору користувача і надає випадкові криптографічні дані. Будь-яка програма, яка потребує випадкових даних, може підключитися і запросити випадкові байти від нього. Він заповнює свій пул ентропії, збираючи випадковість вихідних даних програм системної статистики, таких як 'w', 'last' та 'vmstat'. Основна проблема з демоном полягає в тому, що він залежить від роботи досить завантаженої системи, щоб отримати сильні непередбачувані випадкові дані з цих статистичних даних. Це робить його менш придатним для використання на пристрої Android, так як ці пристрої можуть простоювати протягом тривалого періоду часу, що може призвести до того, що для створення пулу ентропії демону закінчиться випадковість.

Другий варіант пропонує Intel [18], де /dev/ random використовує інструкцію RDRAND. Інструкція RDRAND являє собою апаратний генератор випадкових чисел на кристалі Intel, який генерує випадкові біти з апаратного забезпечення і передає їх через AES, який потім використовується в якості початкового значення для CTR-DRBG DRNG. DRNG потім надає криптографічно безпечні випадкові числа додаткам, які запитують їх за допомогою інструкції RDRAND. Апаратні генератори випадкових чисел зазвичай забезпечують дуже високу ентропію, тому що їх генерація ентропії виконується на такому низькому рівні, що вона стає дуже непередбачуваною. Тим не менш, апаратний RDRAND Intel ГВЧ критикується, тому що він використовує власну реалізацію на кристалі, яка не може бути перевірена. Це не тільки означає, що базова функціональність реалізації не може бути перевірена на наявність дефектів або недоліків, але також можуть бути приховані бекдори в реалізації, які можуть змінити силу ентропії. Ці бекдори не виключені, як стало відомо в 2013 році, коли The New York Times опублікувала статтю [20], де вони стверджують, що Агентство

національної безпеки (АНБ) США працює разом з виробниками чіпів, щоб вставити бекдори в чіпах шифрування. Тсо відповів на це, сказавши: «Я дуже радий, що чинив опір тиску інженерів Intel, щоб не дозволити /dev/random покладатися тільки на інструкцію RDRAND».

У 2014 році було оновлено системний виклик getrandom() Linux, щоб блокувати за замовчуванням, коли пул ентропії не містить принаймні 128 біт ентропії [21]. До цього оновлення у додатків не було можливості дізнатися, чи відповідає випадковість, яку вони просили у /dev/urandom, мінімального порогу безпеки, тому що /dev/urandom завжди був неблокуючим і забезпечував випадковість, навіть якщо пул ентропії ще недостатньо був заповнений. Після цього оновлення, коли пул ентропії не містить принаймні 128 біт ентропії, будь-які запити випадковості, зроблений за допомогою /dev/urandom, будуть блокуватися до тих пір, поки не буде отримана достатня ентропія. Це оновлення значно поліпшило безпеку генератора випадкових чисел /dev / urandom, так як тепер він може надати гарантію безпеки випадковості, яка містить принаймні 128 біт ентропії. Тим не менш, це все ще означає, що Linux ГВЧ залежить від отримання даних від користувача, і це може вимагати користувачам Android-пристрою чекати накопичення достатньої ентропії.

2. Спосіб та алгоритм генерації ентропії на основі показників датчиків

2.1 Обгрунтування обраного способу генерації випадкових чисел

Android дозволяє отримати доступ до багатьох типів датчиків. Деякі з цих датчиків є апаратними, а деякі є програмними.

Апаратні датчики - це фізичні компоненти, вбудовані в телефон або планшет. Вони отримують свої дані шляхом безпосереднього вимірювання конкретних властивостей навколишнього середовища, таких як прискорення, напруженість геомагнітного поля або зміна кута.

Програмні датчики не є фізичними пристроями, хоча вони імітують апаратні датчики. Програмні датчики отримують дані від одного або декількох апаратних датчиків і іноді називаються віртуальними або синтетичними датчиками. Датчик лінійного прискорення і датчик сили тяжіння є прикладами програмних датчиків.

Платформа Android підтримує три категорій апаратних датчиків:

− Датчики руху. Ці датчики вимірюють зусилля прискорення і обертальні зусилля уздовж 3 осей. До цієї категорії відносяться акселерометри, датчики сили тяжіння, гіроскопи і датчики вектора обертання.

− Датчики навколишнього середовища. Ці датчики вимірюють різні параметри навколишнього середовища, такі як температура і тиск навколишнього повітря, освітленість і вологість. Ця категорія включає барометри, фотометри і термометри.

− Датчики положення. Ці датчики вимірюють фізичне положення пристрою. До цієї категорії відносяться датчики орієнтації і магнітометри.

Деякі пристрої на базі Android мають всі типи датчиків. Наприклад, більшість мобільних пристроїв і планшетів мають акселерометр і магнітометр, але менше пристроїв мають барометри або термометри. Крім того, пристрій може мати більше одного датчика даного типу. Наприклад, пристрій може мати два датчика сили тяжіння, кожен з яких має різний діапазон.

Платформа Android надає кілька датчиків[37], які дозволяють відстежувати рух пристрою.

Архітектури датчиків залежить від способу реалізації самого датчика:

− Гравітація, лінійне прискорення, вектор обертання, рух, лічильник кроків і датчики детектора кроків або апаратні або програмні.

− Датчики акселерометра і гіроскопа завжди апаратні.

Більшість пристроїв Android мають акселерометр, і більшість пристроїв тепер мають гіроскоп. Доступність програмних датчиків більш мінлива, тому що вони часто

покладаються на один або декілька апаратних датчиків для отримання даних Залежно від пристрою, ці програмні датчики можуть отримувати свої дані або з акселерометра і з магнітометра, або з гіроскопа.

На платформі Android передбачені два датчики[36], які дозволяють визначати положення пристрою: датчики геомагнітного поля та акселерометр. Платформа Android також пропонує датчик, який дозволяє визначити, наскільки близько обличчя до пристрою (відомий як датчик близькості). Датчик геомагнітного поля та датчик близькості є апаратними. Більшість виробників телефонів і планшетів включають датчик геомагнітного поля. Крім того, виробники телефонів зазвичай включають датчик близькості, щоб визначити, коли телефон тримається близько до обличчя користувача (наприклад, під час телефонного дзвінка). Для визначення орієнтації пристрою можна використовувати показання з акселерометра пристрою та датчика геомагнітного поля.

Датчики положення корисні для визначення фізичного стану пристрою в системі координат світу. Наприклад, можна використовувати датчик геомагнітного поля в поєднанні з акселерометром для визначення положення пристрою щодо магнітного Північного полюса.

Платформа Android надає чотири датчика[38], які дозволяють відстежувати різні властивості навколишнього середовища. Ви можете використовувати ці датчики для моніторингу відносної вологості навколишнього середовища, освітленості, тиску навколишнього середовища і температури навколишнього середовища поблизу пристрою на базі Android. Всі чотири датчика середовища є апаратними і доступні тільки в тому випадку, якщо виробник вбудував їх в пристрій. За винятком датчика освітленості, який використовується більшістю виробників пристроїв для управління яскравістю екрану, датчики навколишнього середовища не завжди доступні на пристроях.

Загалом, датчики використовують стандартну 3-координатну систему координат для вираження значень даних. Для більшості датчиків система координат визначається відносно екрану пристрою, коли пристрій утримується за умовчанням. Коли пристрій утримується за умовчанням, вісь X є горизонтальною і вказує на праву, вісь Y вертикальна та вказує вгору, а ось Z вказує на бік сторони екрану. У цій системі координати за екраном мають негативні значення Z. Ця система координат використовується такими датчиками:

- Датчик акселерометр.
- Датчик сили тяжіння.
- Датчик гіроскопа.
- Датчик лінійного прискорення.
- Датчик магнітного поля.

Існує ряд можливих типів датчиків на пристроях Android, які можна використовувати для отримання випадкових бітів для пулу ентропії. Кожен з датчиків має різні атрибути і відрізняється своєю надійністю при генерації випадкових даних. Деякі з цих датчиків були досліджені для їх надійності і рівень ентропії свої дані.

В статті [11] досліджено ентропійну стійкість акселерометрів і виявили, що вони забезпечують достатнє джерело випадковості навіть при нерухомості. Дослідники виконували різноманітні рухи і визначали значення мінімальної ентропії для кожного зразка руху впродовж 10 хвилин, як показано в таблиці 1.2 Значення мінімальної ентропії в таблиці- це кількість випадкових бітів, які можуть бути отримані з одного 30-бітного сенсора, а не з усієї вибірки розподілу. Результати показують, що навіть стаціонарний акселерометр може забезпечити випадковість з високою ентропією

Таблиця 1.2 – Оцінювання мінімальної ентропії.

Дослід	Мінімальна ентропія
Нерухомо №1	3.4
Нерухомо №2	3.6
В руці	10.8
Дуга	11.3
Падіння	9.1
Трикутник	11.0
Альфа	11.0
Переворот	11.7
Коло	11.4

Для того, щоб захистити користувачів, потрібно створити простий спосіб для шифрування даних Android-пристроїв. Це повинна бути відносно проста система також необхідно буде враховувати обмежену кількість ресурсів, які пристрій має з точки зору батареї, обчислювальної потужності і простору для зберігання даних. Якщо процес шифрування займає надто багато ресурсів, він буде заважати нормальній діяльності телефону користувача.

Вирішальним кроком для шифрування є генерація сильного випадкового початкове значення в якості вхідних даних для системи шифрування. В ідеалі генерація випадкового початкового значення буде відбуватися на пристрої з використанням обмеженої кількості обчислень, заряду батареї і місця для зберігання. В даній розробці представлений досить потужний та енергоефективний генератор випадковості, який буде працювати в просторі користувача. При завантаженні телефону Android додаток генерує пул ентропії від шуму даних датчиків телефону. Після того, як додаток сформував досить сильний пул ентропії, інші процеси на

телефоні Android можуть отримати від додатку випадковість будь-якого розміру. Перш ніж видати необхідні дані, аналізуються дані датчика і оцінюю необхідний час для запуску процесу генерації даних при завантаженні пристрою, перш ніж пул ентропії можна буде вважати досить сильним..

Мета алгоритму генерації випадковостей - забезпечити сильне джерело випадковості. Цей алгоритм може бути використаний на телефоні поруч з існуючими джерелами генерації, такі як /dev/random. Ця комбінація створить більш сильне і безпечне випадкове початкове значення для процесів шифрування на пристрої.

Для досягнення цієї мети потрібно в ході роботи необхідно вирішити наступні завдання:

1. Визначити датчики або комбінації датчиків пристроїв, які підходять для отримання високо ентропійної випадковості.
2. Дослідити як можна отримати високу ентропію випадковості з отриманих даних датчиків і який спосіб отримання випадковості найкраще підходить для цієї мети.
3. Провести оцінку якості ентропії отриманих даних з датчиків пристрою.
4. Розробити алгоритм для ефективного отримання випадкових даних з високою ентропією з датчиків пристроїв для використання в середовищі користувача.

2.2 Спосіб генерації ентропії на основі показників датчиків

Випадковість - дуже корисний ресурс для криптографічних додатків. Випадковість важлива для секретних ключів, адже ненадійне джерело випадковості може скомпрометувати найсильніший криптографічний протокол. Однак реальність така, що процедури отримання випадкових бітів (так званих генератори псевдовипадкових значень) часто не розроблені так добре, як могли б бути. Це небезпечно, так як проблема в безпеки цієї процедури призводить до вразливості всієї системи. Також, якщо ця процедура не має непохитного доказу своєї криптографічної стійкості, то немає ніякої надії на такі властивості для всієї системи.

Підхід до розробки моделі генерування випадкових значень(Рис. 2.1) складається з двох частин. По-перше, це аналіз даних датчика пристрою Android, чи можуть дані датчика містити достатню ентропію і чи можуть бути використані для цілей генерації випадковості. По-друге, розробка моделі, за допомогою якої може бути розроблений алгоритм генерації випадкових значень.

Рисунок 2.1–Схема моделі генерації випадкових значень

Хеш-функція-це відображення h вхідної послідовності з алфавіту A довільної довжини в послідовність з фіксованою довжиною A^n, де довжина n зазвичай коливається від 64 до 512. Для криптографічних цілей вибирається довжина принаймні 256 для n, щоб бути стійким до атак атакуючої сторони, довжина 64 занадто мала для цієї мети. Найбільш важливою властивістю хеш-функцій є те, що інвертувати процес відображення практично неможливо.

Криптографічна хеш-функція - це хеш-функція, яка може протистояти криптоаналітичним атакам. Для цього вона повинна володіти наступними властивостями [30]:

Хеш-функція h є односторонньою функцією, тобто майже для всіх виходів b обчислювально неможливо знайти вхід $a \in A$, при тому, що $b = h(a)$.

Хеш-функція h має слабку стійкість до зіткнень. Це означає, що для заданого значення a обчислювально неможливо знайти друге значення $a' \in A,\ a' \neq a,$, при тому, що $h(a) = h(a')$.

Хеш-функція h стійка до сильних зіткнень. Це означає, що обчислювально неможливо знайти пару значень $a,\ a' \in A,\ a' \neq a$, при тому, що $h(a) = h(a')$.

Потокові шифратори(Рис. 2.2), алгоритми шифрування, які шифрують потік введення тексту за допомогою псевдо-випадкових потоків ключів. Кожен біт отриманого відкритого тексту шифрується окремо за допомогою біта з потоку ключів.

Існує два типи потокових шифрів(Рис. 2.3):

1. У синхронних потокових шифраторах, потік ключів заснований виключно на випадковому введенні ключа;
2. В асинхронних потокових шифраторах результат зашифрованого тексту подається назад в потік ключів для наступного процесу шифрування[31].

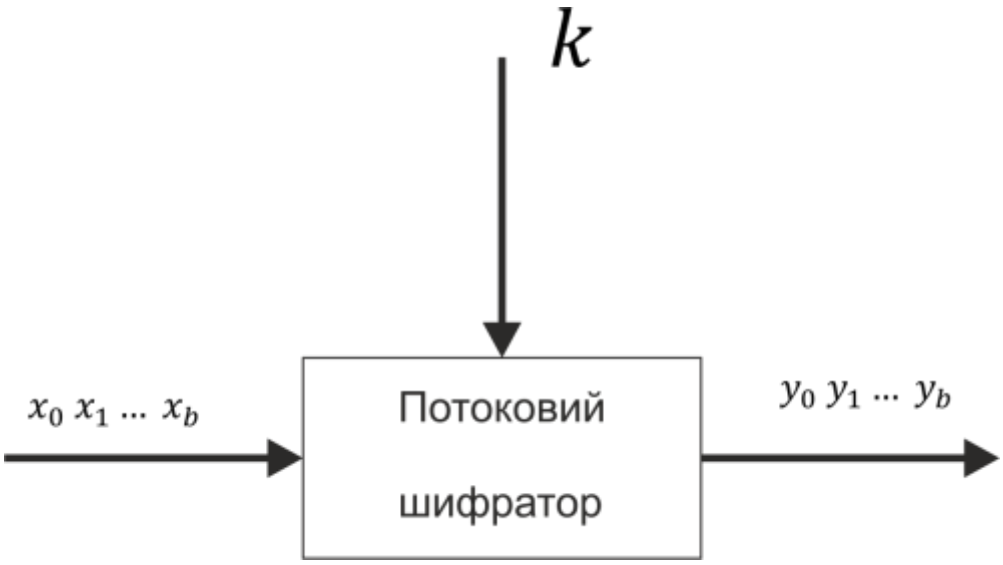

Рисунок 2.2– Принципи кодування b біт потоковим шифратором

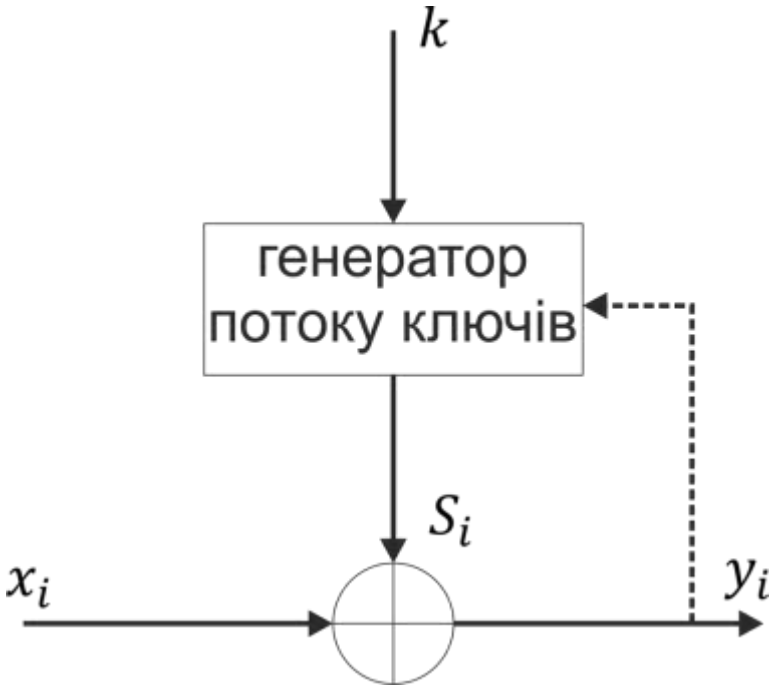

Рисунок 2.3– Синхронний та асинхронний потоковий шифратори

Через свої властивості потоковий шифратор добре підходить для ролі ГПВЧ для криптографічно стійкого генератора псевдовипадкових чисел (КГСГПВЧ) При використанні для шифрування, потоковий шифр об'єднує потік ключів і відкритий

текст для створення зашифрованого тексту. При використанні для генерації випадковості, потік ключів безпосередньо використовується в якості джерела випадковості. Якщо потоковий шифр використовується ключем з високою ентропією від генератора істинних випадкових чисел, результуючий вихідний потік може бути використаний в якості КГСГПВЧ з високою ентропією. Аналогічна конструкція буде використовуватися в алгоритмі, побудованому для цієї моделі, отримані дані датчика будуть об'єднані з інкрементним унікальним номером повідомлення, щоб служити ключем для потокового шифру Salsa20, який описаний у [7], після чого він може надавати випадкові дані за запитом з вихідного потоку.

Генератори випадкових чисел - це системи, що використовують джерело ентропії для генерації випадкових бітів даних. Можна виділити два типи ГВЧ: генератори справжніх випадкових чисел і генератори псевдовипадкових чисел.

Генератори істинних випадкових чисел використовують джерело природної ентропії для генерації випадковості (наприклад, фоновий шум, атмосферний шум, електромагнітний апаратний шум або космічне випромінювання). Природна ентропія, як правило, дуже непередбачувана, недетермінована і її важко виміряти сторонньому спостерігачеві. Недоліком природної ентропії є те, що ГВЧ необхідно регулярно оновлювати свій пул ентропії, щоб залишатися непередбачуваним. В процесі збору додаткової ентропії ГВЧ блокується до тих пір, поки не збере достатньо ентропії для задоволення попиту. /dev / random є прикладом цього процесу, як тільки він досягне межі ентропії, він буде блокуватися до тих пір, поки не буде зібрано достатню ентропію для поповнення пулу ентропії. Така поведінка блокування може сильно уповільнити роботу систем при виконанні дуже великих запитів випадковості.

Генератори псевдовипадкових чисел використовують односторонні функції для генерації ентропії. Ці алгоритми потребують вхідного початкового значення для генерації довгих випадкових послідовностей даних. Перевага ГПВЧ полягає в тому, що вони можуть генерувати практично нескінченний потік випадкових даних після правильного заповнення і не зіткнуться з проблемою блокування процесу поки оновлюється пул ентропії. Недоліком є те, що введення початкового значення ГПВЧ є серйозною вразливістю. Якщо початкове значення не достатньо стійке, тоді всі згенеровані дані скомпрометовані і стають передбачуваними. Тому важливо, щоб вихідні дані не були передбачені зовнішнім спостерігачем і мали високий рівень ентропії.

Криптографічно стійкий генератор псевдовипадкових чисел є ГПВЧ, адаптованими для використання в криптографії. КГСГПВЧ потрібно початкове значення високої ентропії для генерації стійкої випадковості, що робить його придатним для використання у криптографічних алгоритмах. Один із способів досягти цього - об'єднати генератор справжніх випадкових чисел і генератор псевдовипадкових

чисел для створення моделі КГСГПВЧ.

У цій моделі генератор справжніх випадкових чисел використовується для генерації початкового значення високої ентропії з недетермінованого джерела (наприклад, апаратного генератора випадкових чисел). Це початкове значення потім використовується в якості вхідних даних для побудови ГПВЧ для генерації великих обсягів випадкових даних. При правильному побудові ця модель має ту перевагу, що вона недетермінована, як генератор справжніх випадкових чисел, але вона як і раніше здатна генерувати великі обсяги даних без блокування після ініціалізації початкового значення ГПВЧ.

2.3 Алгоритми генерації пулу ентропії та генерування випадкових значень

На основі аналізу вибірок даних датчика пристрою Android можна стверджувати, що дані датчика можуть містити ентропію і можуть бути використані для цілей генерації випадковості. Для того щоб досягти цього, було побудовано алгоритм який може збирати і зберігати велику кількість даних з датчиків при різних обставинах. Результати цього аналізу дають оцінку кількості ентропії, яка може бути отримана завдяки даним, які генерують датчики пристрою Android. Також розроблено алгоритм, який реалізує даний спосіб генерації. Цей алгоритм демонструє, що можна використовувати дані датчика пристрою Android для ефективного отримання випадковості з високою ентропією. Якщо алгоритм адаптований для використання спільно з /dev / random, то отримані випадкові дані можуть бути використані в експлуатаційному середовищі в криптографічних цілях. Таке поєднання існуючого стандарту генерації випадковостей /dev/random і запропонованого способу генерації з використанням сенсорних даних посилить якість випадковості і зробить її більш стійкою до атак.

Загальна структура алгоритму наведена на рисунку 3.2. Алгоритм автоматично визначає, які датчики доступні для генерації даних на пристрої, на якому він використовується. Доступні датчики будуть періодично генерувати дані, і ці дані подаються в хеш SHA-256. Коли достатня ентропія була зібрана з даних датчика, хеш буде перероюляти всі дані в 256-бітний висновок. Цей 256-бітний вихід використовується в якості ключа для алгоритму XSalsa20 [24], поряд з 192-бітним інкрементним унікальним номером повідомлення. Отриманий потік з алгоритму XSalsa20 потім використовується в якості безперервного потоку випадковості, який може генерувати випадковість за запитом.

Подібно до Salsa20, Xsalsa20 захищений від атак часу і забезпечений власним 64-бітний лічильником блоків, для того щоб уникнути збільшення коливання після

кожного блоку.

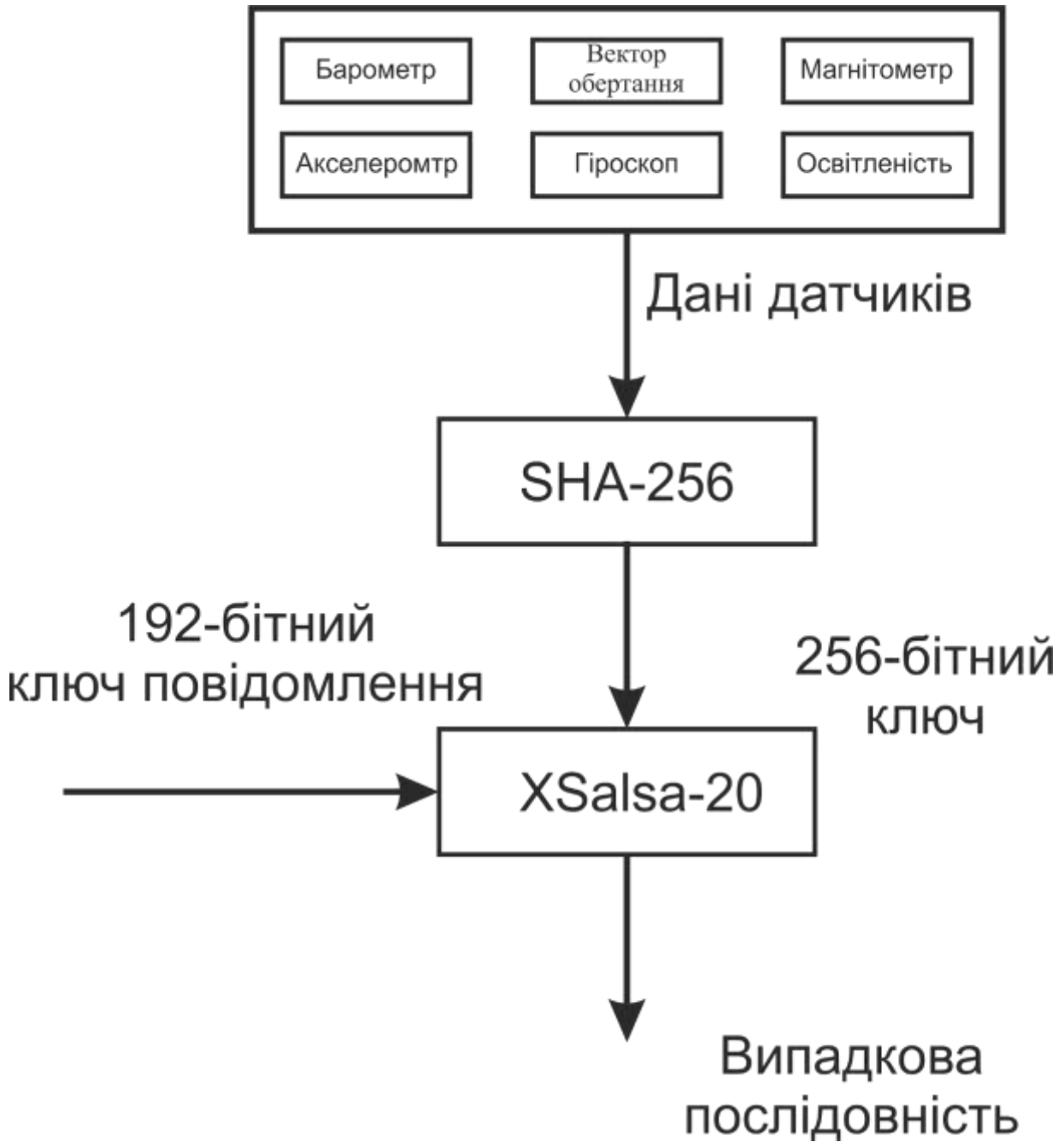

Рисунок 3.2–Схема реалізації алгоритму

Для взаємодії з сенсорами Android використовується Android Sensor Event API, який надає доступ до прослуховування подій датчика і генерування даних датчиків. Sensor Event API надає список всіх датчиків, які доступні на пристрої Android, які потім використовуються для генерації даних. API зберігає всі необроблені дані датчика, які він отримує в масиві значень, де кожне значення у масиві являє вісь датчика. Наприклад, датчик акселерометра забезпечує прискорення по осі x, осі y і осі

z. Швидкість, з якою API генерує дані датчика, встановлюється на саму коротку затримку в алгоритмі, який буде генерувати найвищий обсяг даних, можливий під час генерації пулу ентропії. Точна кількість даних, які генеруються, варіюється в залежності від пристрою, на якому працює алгоритм, але тестування показує, що в середньому буде близько 375.000 показань датчиків в хвилину.

Бібліотека Java Message Digest використовується для SHA-256, ця бібліотека надає методи, щоб оновити хеш з даними і перетворити в вихідні дані, як тільки пул ентропії буде заповнений. Цей хеш забезпечує 256-розрядний вивід і слідує за хеш-властивостями односторонньої функції і опирається колізії.

Для реалізації алгоритму XSalsa20 [24] використовується Spongy Castle API [25], цей API є різновидом Bouncy Castle Java Cryptography API [26], який адаптований для роботи на пристроях Android. Алгоритм XSalsa20 є модифікацією алгоритму Salsa20 з великим розміром (біт) унікального номера повідомлення для запобігання переповнення можливих значень номера повідомлення. Стандартний алгоритм Salsa20 має 64-бітний код, а алгоритм XSalsa20 має 192-бітний код. У звичайному випадку ця різниця не має значення, так як в сеансі не повинно бути більше ніж 264 запитів, але більший розмір унікального номера повідомлення робить алгоритм більш стійким до атак переповнювання. З меншим розміром унікального номера повідомлення атакуючий міг зробити велику кількість запитів переповнити пул номерів повідомлень, це могло змусити лічильник скинутися і використати номер повідомлення двічі хоча він повинен бути унікальним. Це порушить безпеку, тому що це розкриє вхідний ключ і поставить під загрозу всю систему. Алгоритм XSalsa20 автоматично збільшує номер повідомлення після кожного запиту, який він отримує, тому він ніколи не буде використовувати той же номер повідомлення двічі.

3 Розробка та тестування додатку

3.1 Розробка додатку для Android-пристроїв

Для розробки програмного забезпечення використаємо Android Studio яка розроблена фірмою Google. Для демонстрування роботи алгоритму була написана програма на мові Java, з використанням бібліотеки XSalsa 20, та SensorEvent API для роботи з сенсорами Android пристроїв. Головний екран додатку має наступний вигляд(рис.3.1). Цей інтерфейс призначений виключно для демонстрації зв'язку між криптографічним процесом і демоном генератора випадковостей

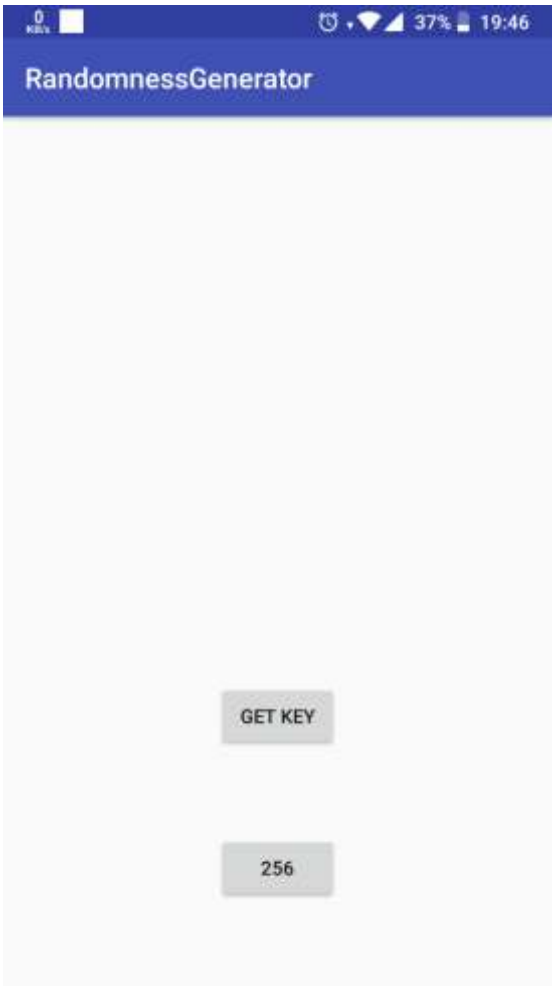

Рисунок 3.1 – Головний екран

На головному екрані можна задати кількість біт, яку хочемо отримати від генератора (рис.3.2). Для цього був розроблено діалогове вікно, яке з’являється при натисканні на нижню кнопку. Діалогове вікно складається з віджета прокрутки в якому за допомогою свайпів можна вибрати необхідну кількість біт, та зберегти це значення

за допомогою кнопки “SAVE”.

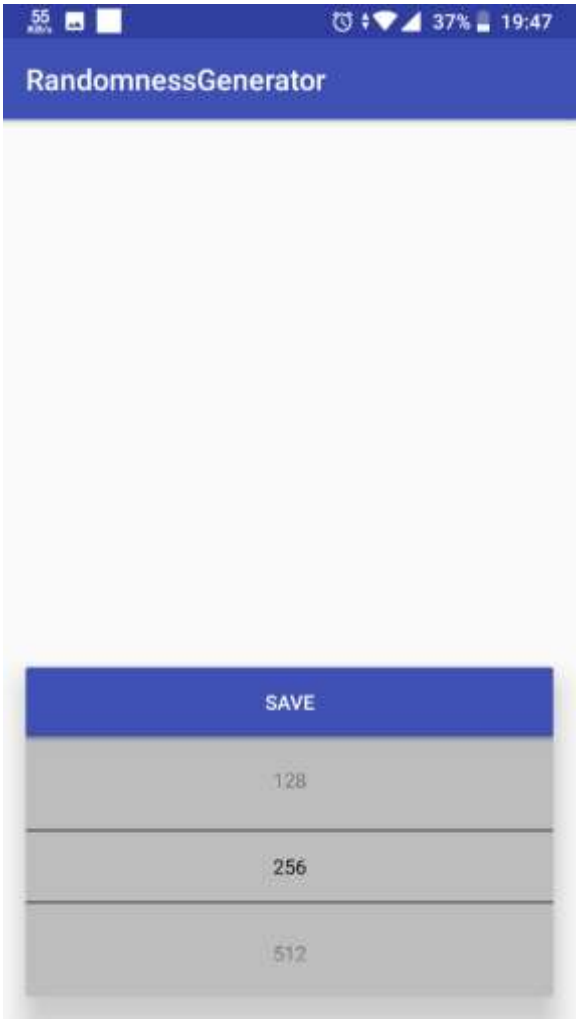

Рисунок 3.2 – Вибір кількості біт

Якщо пул ентропії заповнився не повністю, користувачу відобразиться ProgressBar, який буде демонструвати прогрес заповнення пулу ентропії(рис. 3.3). Коли результат буде згенерований він автоматично буде виведений на екран (рис 3.4).

Рисунок 3.3 – Вибір кількості біт

Рисунок 3.4 – Виведення результату

Як тільки завантажується операційна система програма отримує повідомлення

про цю подію і запускає сервіс для заповнення пулу ентропії. Даний сервіс працює у фоновому потоці поки операційна система не закінчить сесію. Щоб система Android не завершала сервіс потрібно вивести повідомлення про роботу сервіса на панель повідомлень (рис 3.5).

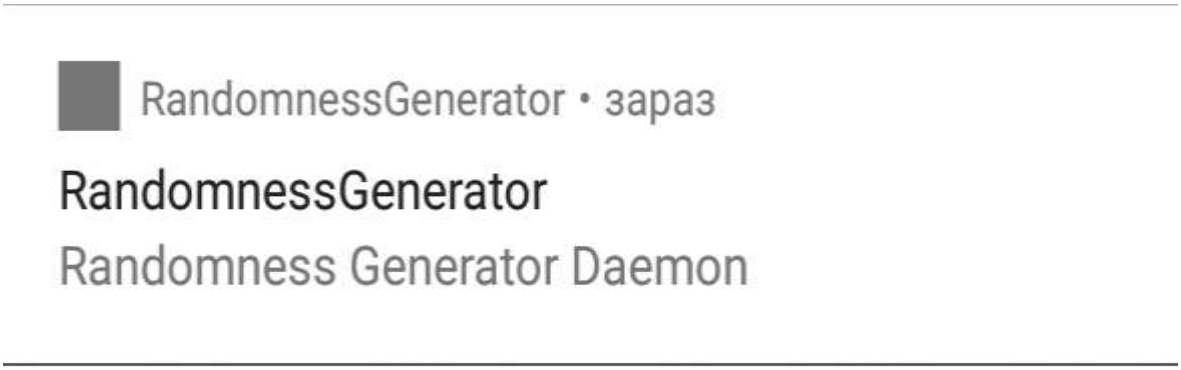

Рисунок 3.5 – Виведення повідомлення про роботу сервіса

3.2 Результати тестування алгоритму генерації

Для оцінки продуктивності алгоритму з точки зору якості статистичної випадковості був протестований обсяг вихідних даних з використанням тестів diehard [29]. Тести diehard — це набір статистичних тестів для вимірювання якості набору випадкових чисел. Вони були розроблені Джорджем Марсальей протягом декількох років і вперше опубліковані на CD-ROM, присвяченому випадковим числам. Разом вони розглядаються як один з найбільш строгих існуючих наборів тестів.

Дні народження (Birthday Spacings) — вибираються випадкові точки на великому інтервалі. Відстані між точками повинні бути асимптотично розподілені по Пуассону. Назву цей тест отримав на основі парадоксу днів народження.

Пересічні перестановки (Overlapping Permutations) — аналізуються послідовності п'яти послідовних випадкових чисел. 120 можливих перестановок повинні виходити зі статистично еквівалентною ймовірністю.

Ранги матриць (Ranks of matrices) — вибираються деяка кількість біт з деякої кількості випадкових чисел для формування матриці над {0,1}, потім визначається ранг матриці.

Мавпячі тести (Monkey Tests) — послідовності деякої кількості біт інтерпретуються як слова. Рахуються пересічні слова в потоці. Кількість «слів», які не з'являються, повинні задовольняти відомому розподілу. Назву цей тест отримав на

основі теореми про нескінченну кількість мавп.

Підрахунок одиничок (Count the 1's) — рахуються поодинокі біти в кожному з наступних або обраних байт. Ці лічильники перетворюється в "букви", і обчислюється кількість появи п’яти буквених "слів".

Тест на парковку (Parking Lot Test) — поодинокі кола випадково розміщуються в квадраті 100×100. Якщо окружність перетинає вже існуючу, потрібно спробувати ще. Після 12 000 спроб, кількість успішно "припаркованих" кіл повинно бути нормально розподілено.

Тест на мінімальну відстань (Minimum Distance Test) — 8000 точок випадково розміщуються в квадраті 10 000×10 000, потім знаходиться мінімальна відстань між будь-якими парами. Квадрат цієї відстані повинен бути експоненціально розподіллено з деякою медіаною.

Тест випадкових сфер (Random Spheres Test) — випадково вибираються 4000 точок в кубі з ребром 1000. У кожній точці поміщається сфера, чий радіус є мінімальною відстанню до іншої точки. Мінімальний обсяг сфери повинен бути експоненціально розподіллено з деякою медіаною.

Тест стиснення (The Squeeze Test) — 2^{31} множиться на випадкові числа в діапазоні [0,1) до тих пір, поки не вийде 1. Повторюється 100 000 раз. Кількість дійсних чисел необхідних для досягнення 1 має бути розподілено певним чином.

Тест пересічних сум (Overlapping Sum Test) — генерується довга послідовність дійсних чисел з інтервалу [0,1). В ній підсумовуються кожні 100 послідовних чисел. Суми повинні бути нормально розподілені з характерними середнім і дисперсією.

Тест послідовностей (Runs Test) — генерується довга послідовність на [0,1). Підраховуються висхідні і низхідні послідовності. Числа повинні задовольняти деякому розподілу.

Тест гри в кості (The Craps Test) - грається 200 000 ігор в кості, підраховуються перемоги і кількість кидків в кожній грі. Кожне число повинно задовольняти деякому розподілу.

Для тестування ГВЧ, тестами diehard було дано 10 мегабайт випадкового виходу. Цей вихід після цього випробуваний 16 методами випробування, які задають ρ-значення до оброблюваних даних, в підсумку 233 ρ-значень задано 16 методами випробування. Ці ρ-значення представляють статистичний розподіл даних. Система ГВЧ пройшла критерій незламності, якщо результуючі ρ-значення рівномірно розподілені по полю [0; 1]. Відступ від рівномірного розподілу вказує на те, що деякі тести виявили передбачувані закономірності в даних.

Графік на рис. 3.6 показує розподіл вихідних даних прототипу синім кольором, а ідеальний рівномірний розподіл-червоним кольором. Вісь y представляє ρ-значення, а вісь x являє відсортований розподіл всіх 233 результуючих ρ-значень від найнижчого

до найвищого. На графіку видно, що вихід прототипу пройшов випробування, так як його розподіл дуже близько до ідеального рівномірному розподілу. Слід очікувати, що результат є статистично випадковим після вилучення XSALSA20 ГПВЧ, але він служить підтвердженням того, що модель побудови прототипу реалізована надійно.

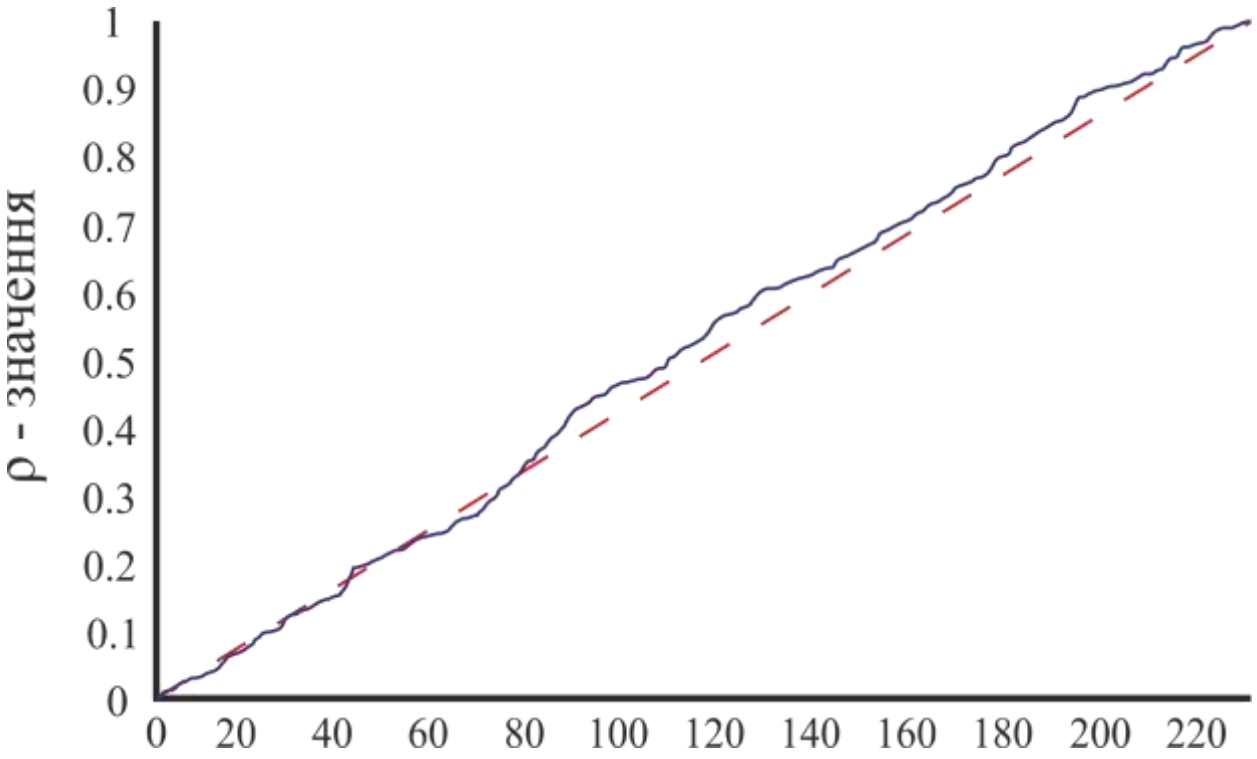

Рисунок 3.6 – Сортований розподіл вихідних даних a

3.3 Результати тестування програми для платформи Android

Продуктивність процесу генерації даних додатком на пристрої Android також враховується для загальної якості. Демон сервісу генерації даних, не повинен турбувати користувача пристрою, сповільнюючи або навіть блокуючи пристрій. Тому важливо, щоб процес генерації ентропії відбувався ефективно і не заважав користувачеві виконувати свої звичайні дії на пристрої. Продуктивність додатку можна оцінити за допомогою наступних критеріїв:

1. Тривалість генерації ентропійного пулу
2. Використання ресурсів ЦП
3. Використання ресурсів ОЗУ
4. Використання ресурсів акумулятора
5. Ентропійна якість

На рисунку 3.7 зображено використання додатком ЦП, завантаження ОЗУ, і використання ресурсів акумулятора під час генерації пулу ентропії. На рисунку 4.8 зображено використання ЦП, ОЗУ і акумулятора вже після того як було згенеровано

пул ентропії.

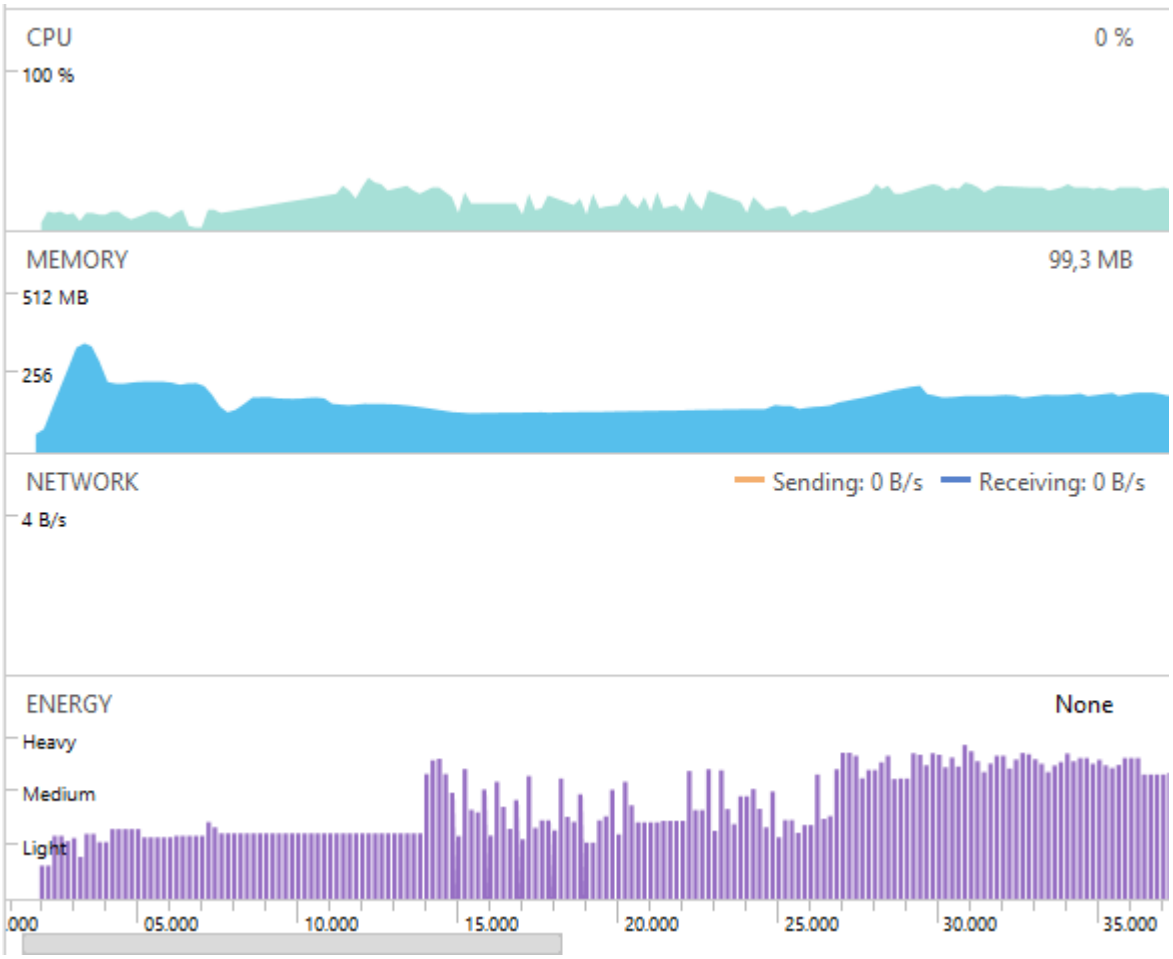

Рисунок 3.7 – Використання ресурсів пристрою при генерації пулу ентропії

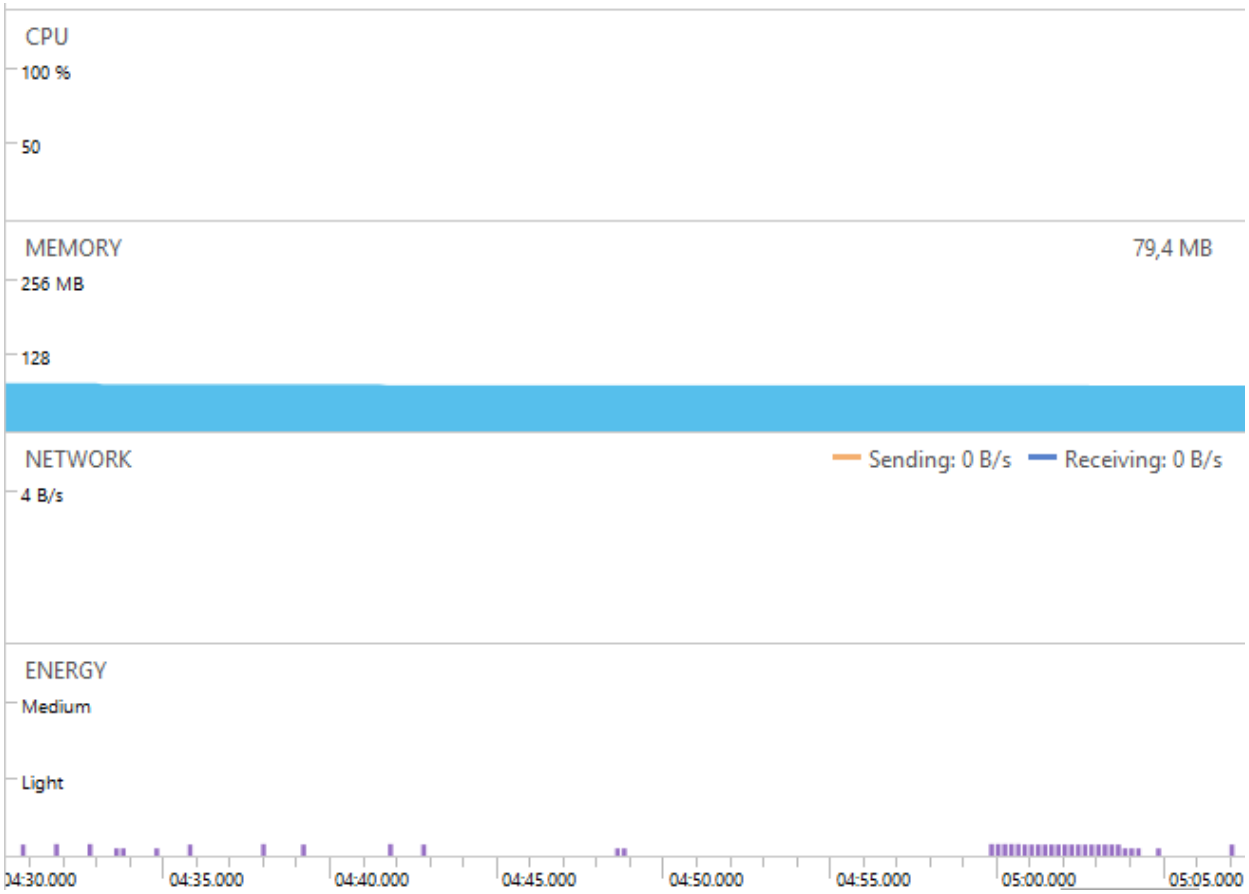

Рисунок 3.8 – Використання ресурсів пристрою після генерації пулу ентропії

Тривалість генерації ентропійного пулу

В даний час встановлено значення 60 секунд, але це значення має дуже консервативне значення, щоб дати максимальну гарантію безпеки. Ймовірно, це значення може бути зменшено до 5-10 секунд, не опускаючись нижче порога ентропії 256 біт, що не завдасть шкоди генерації стійких до атаки даних генератора.

Використання ресурсів ЦП

При генерації пулу ентропії завантаження ЦП може різко зрости(рис. 3.9). Але після того, як пул ентропії заповнений і демон працює стаціонарно, завантаження ЦП падає майже до 0%(рис.3.10). Коли випадковість запитується завантаження процесора йде трохи вгору, але це не повинно турбувати користувача

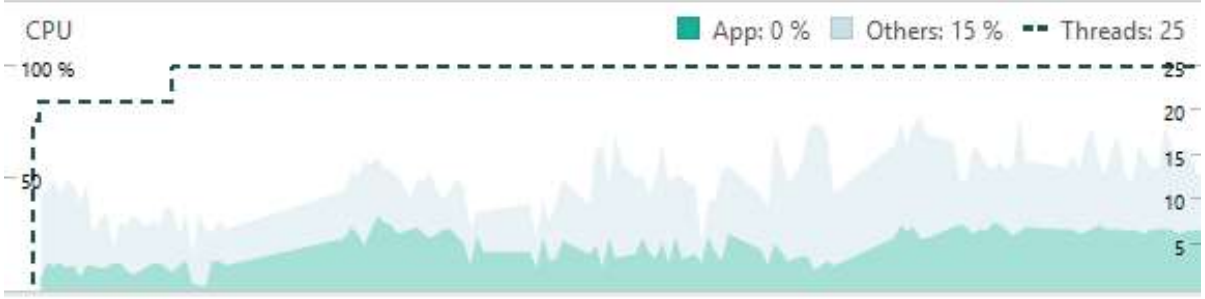

Рисунок 3.9 – Завантаження ЦП при генерації пулу ентропії

Рисунок 3.10 – Завантаження ЦП коли пул ентропії згенерований

Використання ресурсів ОЗУ

Використовує великий обсяг оперативної пам'яті під час створення пулу ентропії(рис. 3.11), з міркувань безпеки весь пул ентропії зберігається в оперативній пам'яті. Але після створення ентропії демон використовує набагато менші ресурси ОЗП(рис 3.12).

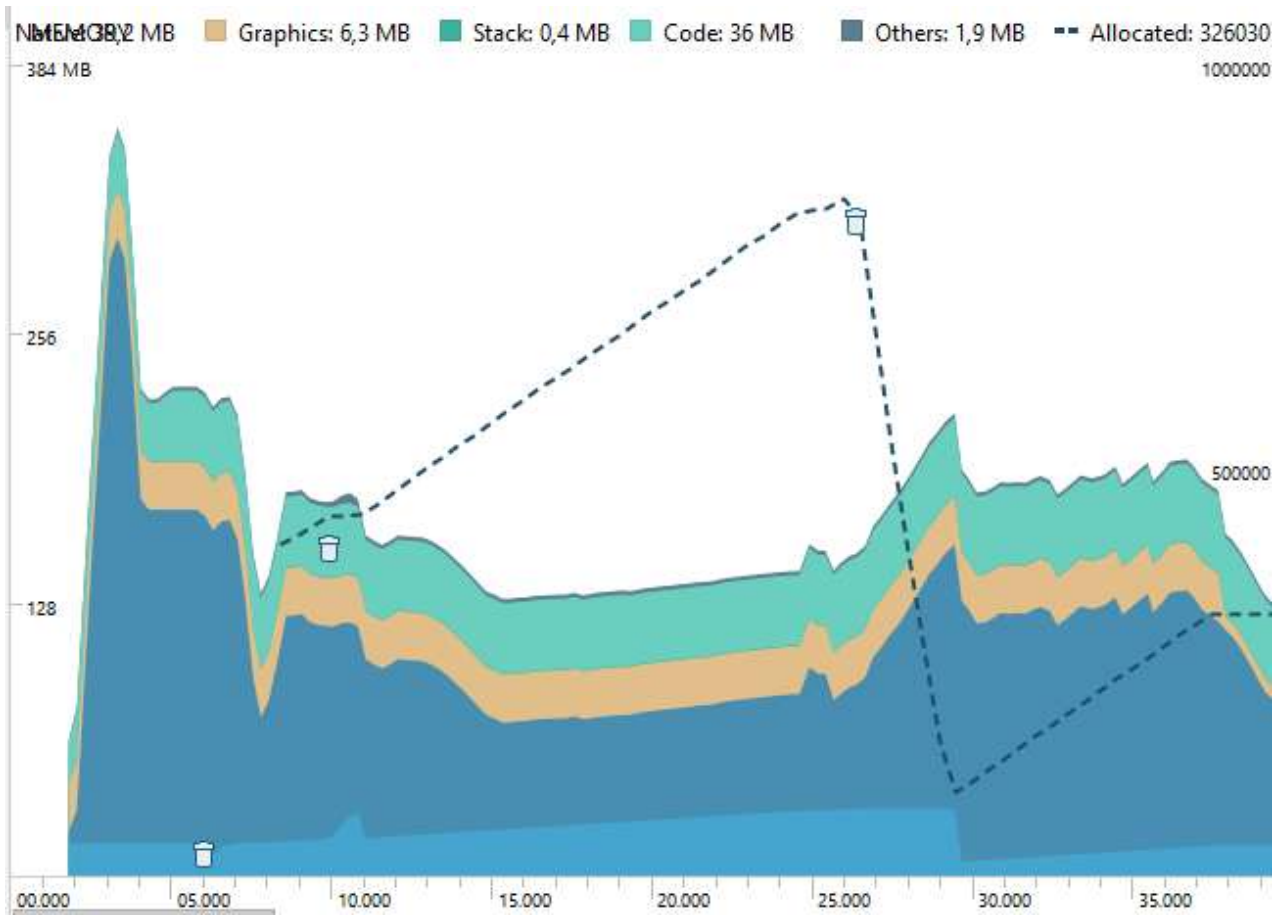

Рисунок 3.11 – Завантаження ОЗУ при генерації пулу ентропії

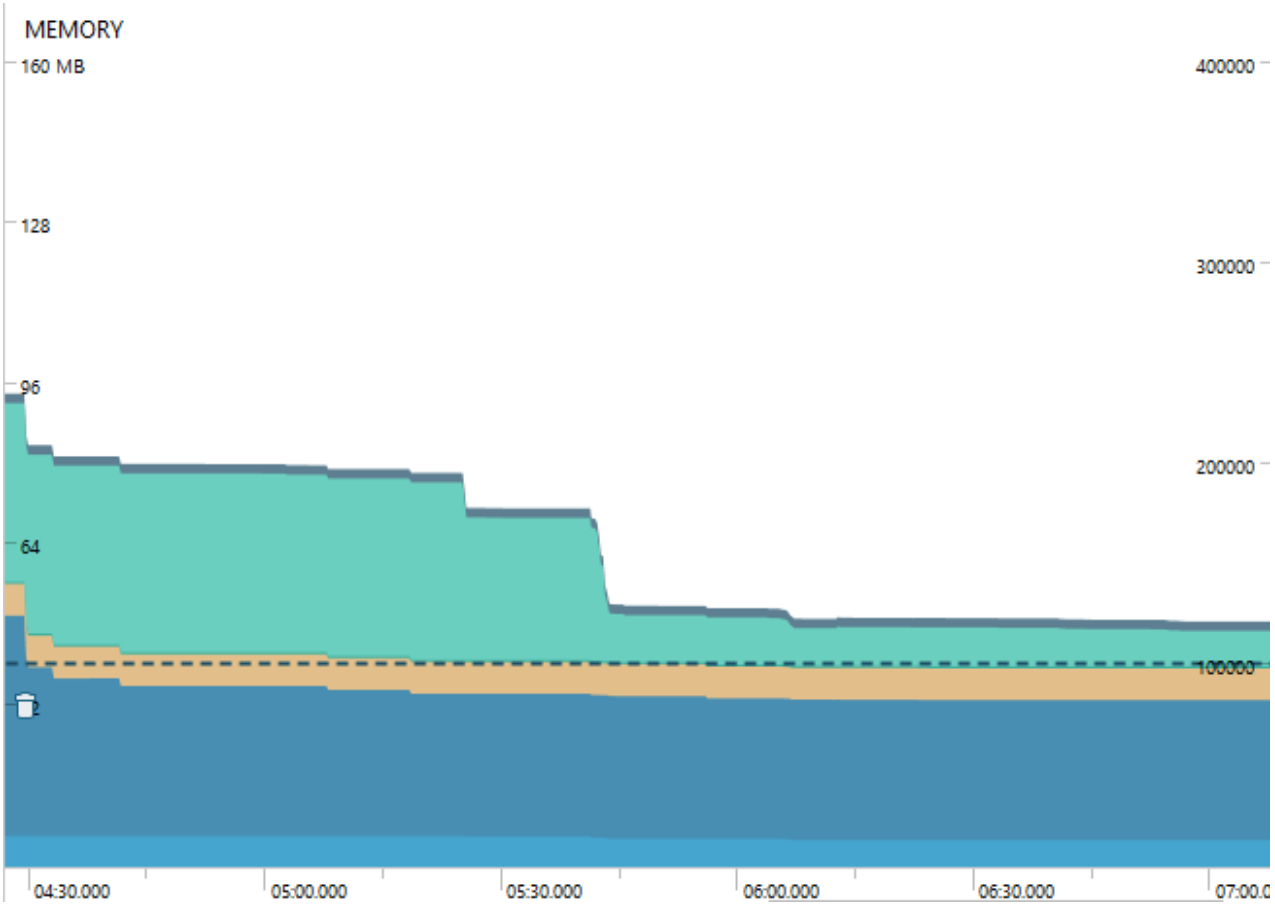

Рисунок 3.12 – Завантаження ОЗУ коли пул ентропії згенерований

Використання ресурсів акумулятора

Як і в випадку з ОЗУ, використовує велику кількість ресурсів акумулятора під час створення пулу ентропії(рис. 3.13). Але після створення ентропії демон майже не використовує ресурси акумулятора(рис 3.14).

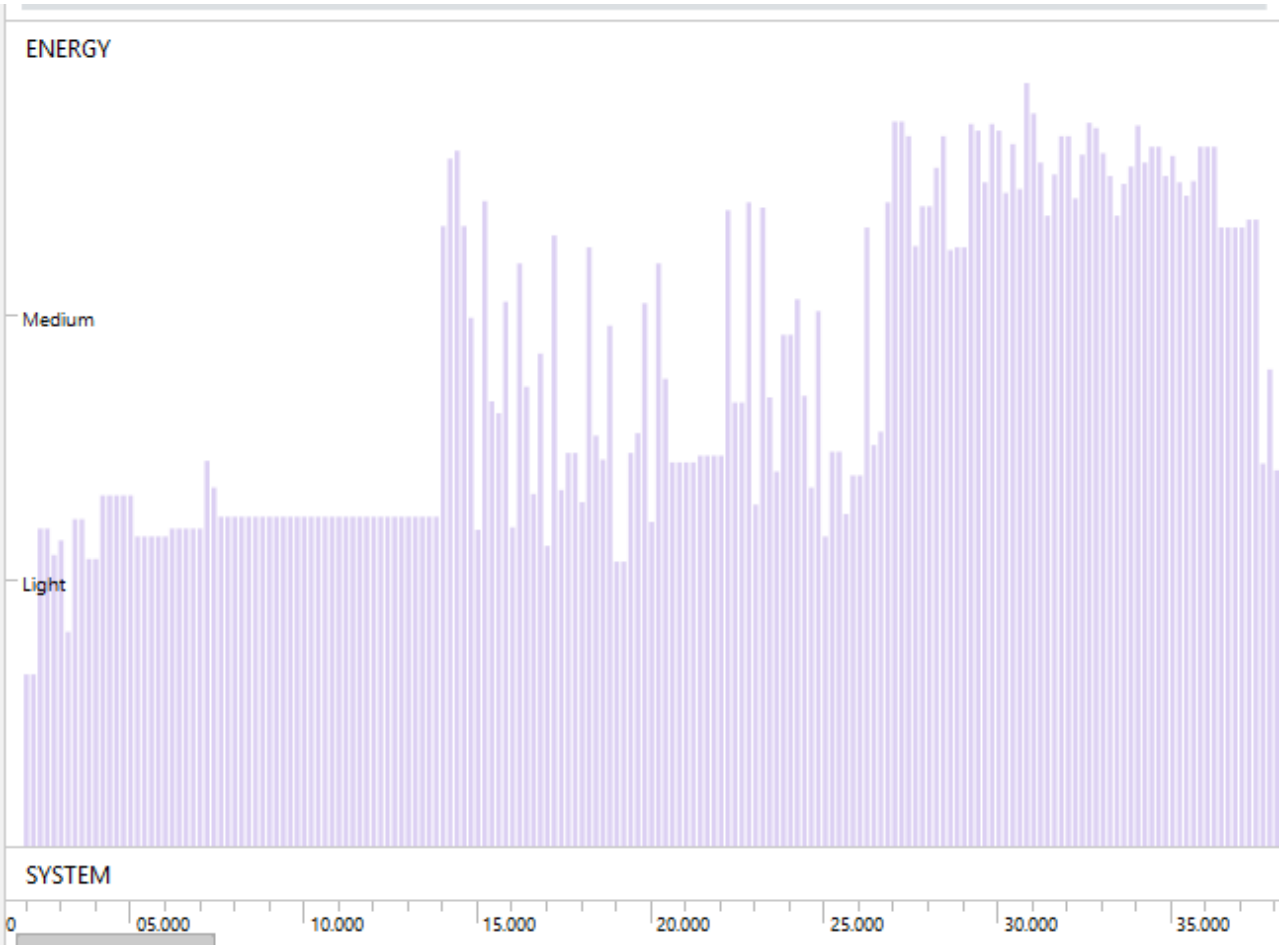

Рисунок 3.13 – Використання ресурсів акумулятора при генерації пулу ентропії

Рисунок 3.14 – Використання ресурсів акумулятора коли пул ентропії згенерований

Ентропійна якість

Тестування оцінки ентропії показує, що додаток може зібрати необхідні 256 біт ентропії протягом від 7 до 42 мілісекунд. Подальше тестування показує, що дані більшості датчиків мають дуже високу якість ентропії. Тестування виходу ГВЧ за допомогою тестів Diehard визначає, чи є вихід ГВЧ статистично випадковим.

Продуктивність по цим критеріям демонструє, що демон не буде турбувати користувача, сповільнюючи пристрій, так як він використовує обмежені ресурси з точки зору процесора, оперативної пам'яті і акумулятора, як тільки він згенерував пул ентропії. Демон буде використовувати більше ресурсів пристрою під час генерації пулу ентропії, але це необхідно тільки один раз під час завантаження пристрою протягом обмеженого часу.

ВИСНОВКИ

У даній роботі було досліджено, завдання для забезпечення сильного джерела випадковості, яке може бути використано додатком на Android пристрої сумісно з існуючим джерелом, такими як /dev/random. Це нове джерело випадковості має сприяти усуненню вразливостей в /dev/random в подіях з низькою ентропією. Ентропія не залежить від взаємодії з користувачем.

Для вирішення цього завдання було розробленно спосіб, який використовував дані датчиків пристрою для отримання даних з високою ентропією, які можуть бути використані в поєднанні з /dev/random для зміцнення пулу ентропії та зменшення кількості подій з низькою ентропією.

На основі спосіб був розроблені алгоритми і написана програма для використання в Android середовищі, яка генерував випадкові числа, використовуючи ентропію, зібрану з даних, які генерували датчики пристрою Android.

Результати показали, що зразки даних датчиків містять велику кількість непередбачуваності, особливо, якщо враховувати розряди десяткових значень.

Основні результати роботи :

1. Визначені основні проблеми генераторів випадкових значень. Показано, що розробка додаткового пулу ентропії є актуальною.

2. Проведений аналіз датчиків та комбінацій датчиків пристрою Android, для того щоб визначити, які підходять для отримання високо ентропійної випадковості.

3. Розроблений способі для генерування випадкових значень, що дозволяє заповнювати пул даних без взаємодії з користувачем. Даний алгоритм є не блокуючим, та надає можливість використання поруч з іншими генераторами.

4. Розроблений алгоритм для генерування випадкових значень, який можна використовувати поруч з іншими генераторами.

5. Була зробленна оцінка якості випадковості для результатів роботи алгоритму з використанням тестів Diehard продемонструвала, що вихід має майже ідеальний рівномірний розподіл. Таким чином, випадковість, створювана алгоритмом, може бути використана для роботи в криптографічних цілях, таких як генерація ключів для схем шифрування або генерація nonce для протоколів автентифікації.

6. Проведено аналіз роботи алгоритма, який показав високу продуктивність після закінчення процесу заповнення ентропійного пулу. Після заповнення пулу, генератор випадкових чисел може працювати постійно, він використовує обмежену кількість потужності процесора, об'єму оперативної пам'яті і ресурсів батареї.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 2.4bn smartphone users in 2017, says eMarketer | Mobile Marketing Magazine. [Електронний ресурс]. URL: https://mobilemarketingmagazine.com/24bn-smartphone-users-in-2017-says-emarketer (Дата звернення: дата звернення: 5.01.2020).
2. Mobile Operating System Market Share Worldwide | StatCounter Global Stats [Електронний ресурс]. URL: http://gs.statcounter.com/os-market-share/mobile/worldwide (Дата звернення:дата звернення: 5.09.2018).
3. Apps vulnerable to hacking, warns security company | Technology | The Guardian Stats [Електронний ресурс]. URL: https://www.theguardian.com/technology/2013/dec/11/app-hacking-commonplace-warns-security-company (Дата звернennядата звернення: 5.01.2020).
4. Luciano Bello and Maximiliano Bertacchini. Predictable PRNG in the vulnerable Debian OpenSSL package: the what and the how. 2008. URL: http://www.researchgate.net/profile/Luciano_Bello/publication/255569058_Predictable_PRNG_In_The_Vulnerable_Debian_OpenSSL_Package/links/542d3a370cf29bbc126d2223.pdf Predictable PRNG In The Vulnerable Debian OpenSSL Package - The What And The How [Електронний ресурс]. URL: https://www.researchgate.net/profile/Luciano_Bello/publication/255569058_Predictable_PRNG_In_The_Vulnerable_Debian_OpenSSL_Package/links/542d3a370cf29bbc126d2223.pdf (Дата звернення: дата звернення: 5.01.2020).
5. Random Number Bug in Debian Linux - Schneier on Security [Електронний ресурс]. URL: https://www.schneier.com/blog/archives/2008/05/random_number_b.html (Дата звернення: дата звернення: 5.01.2020).
6. Debian -- Security Information -- DSA-1571-1 openssl [Електронний ресурс]. URL: https://www.debian.org/security/2008/dsa-1571 (Дата звернення: дата звернення: 5.01.2020).
7. NVD - CVE-2008-0166 [Електронний ресурс]. URL: https://nvd.nist.gov/vuln/detail/CVE-2008-0166 (Дата звернення:дата звернення: 5.01.2020).еек
8. Auditing GitHub users' SSH key quality [Електронний ресурс]. URL: https://blog.benjojo.co.uk/post/auditing-github-users-keys (Дата звернення: дата звернення: 5.01.2020).
9. Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining Your Psand Qs: Detection of Widespread Weak Keys in Network Devices. In USENIX SecuritySymposium, pages 205{220, 2012. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices [Електронний ресурс]. URL:

https://factorable.net/weakkeys12.extended.pdf (Дата звернення: дата звернення: 55.01.2020).

10. Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergniaud, and Daniel Wichs.Security analysis of pseudo-random number generators with input:/dev/random is not robust. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. Security Analysis of Pseudo-Random Number Generators with Input: /dev/random is not Robust [Електронний ресурс]. URL: https://eprint.iacr.org/2013/338.pdf (Дата звернення: дата звернення: 5.01.2020).

11. Jonathan Voris, Nitesh Saxena, and Tzipora Halevi. Accelerometers and randomness: perfecttogether. In Proceedings of the fourth ACM conference on Wireless network security. ACM, 2011. Accelerometers and Randomness: Perfect Together [Електронний ресурс]. URL: https://info.cs.uab.edu/saxena/docs/wisec.pdf (Дата звернення: дата звернення: 5.01.2020).

12. Recovering Bitcoin private keys using weak signatures from the blockchain / Nils Schneider [Електронний ресурс]. URL: http://www.nilsschneider.net/2013/01/28/recovering-bitcoin-private-keys.html (Дата звернення: дата звернення: 5.01.2020).

13. Android Security Vulnerability [Електронний ресурс]. URL: https://bitcoin.org/en/alert/2013-08-11-android (Дата звернення: дата звернення: 5.01.2020).

14. Thomas Watson. The complexity of estimating min-entropy. Computational Complexity. The Complexity of Estimating Min-Entropy [Електронний ресурс]. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.9125&rep=rep1&type=pdf (Дата звернення:дата звернення: 5.01.2020).

15. Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too{optimal recovery strategies for compromised rngs. In Advances in Cryptology-CRYPTO 2014, 2014. How to Eat Your Entropy and Have it Too — Optimal Recovery Strategies for Compromised RNGs [Електронний ресурс]. URL: https://eprint.iacr.org/2014/167.pdf (Дата звернення: дата звернення: 5.01.2020).

16. Theodore Ts’o and Matt Mackall. random.c - A strong random number generator. random.c\char\drivers - kernel/git/stable/linux.git - Linux kernel stable tree [Електронний ресурс]. URL: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/char/random.c ((Дата звернення: дата звернення: 5.01.2020).

17. EGD: The Entropy Gathering Daemon [Електронний ресурс]. URL: http://egd.sourceforge.net/ (Дата звернення: дата звернення: 5.01.2020).

18. Intel. Intel Digital Random Number Generator (DRNG), 2012 Intel® Digital Random Number Digital Random Number Generator Generator(DRNG) [Електронний ресурс].

URL:
https://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf (Дата звернення: дата звернення: 5.01.2020).
19. Daniel J Bernstein. Entropy Attacks! The cr.yp.to blog, 2014.cr.yp.to: 2014.02.05: Entropy Attacks! [Електронний ресурс]. URL: http://blog.cr.yp.to/20140205-entropy.html (Дата звернення: дата звернення: 5.01.2020).
20. Jeff Larson Nicole Perlroth and Scott Shane. N.S.A. able to foil basic safeguards of privacy on web, 2013. N.S.A. Able to Foil Basic Safeguards of Privacy on Web - The New York Times [Електронний ресурс]. URL: https://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html (Дата звернення: дата звернення: 5.09.2018).
21. Linux-Kernel Archive: Re: [PATCH -v5] random: introduce getrandom(2) system call [Електронний ресурс]. URL: http://lkml.iu.edu/hypermail/linux/kernel/1407.3/01159.html (Дата звернення: дата звернення: 5.01.2020).
22. Генератор Геффа, Генератор «стоп - пошел», Чередующийся генератор «стоп - пошёл», Каскад Голлмана - Обеспечение защиты речевой информации в сетях мобильной связи [Електронний ресурс]. URL:
https://studbooks.net/2339913/tehnika/generator_geffa (Дата звернення: 5.01.2020).
23. Лінійний конгруентний метод — Вікіпедія [Електронний ресурс]. URL:
https://uk.wikipedia.org/wiki/ (дата звернення: 5.01.2020).
24. Daniel J Bernstein. The Salsa20 family of stream ciphers. In New stream cipher designs. Springer, 2008. The Salsa20 family of stream ciphers [Електронний ресурс]. URL:
http://cr.yp.to/snuffle/salsafamily-20071225.pdfhttp://cr.yp.to/snuffle/salsafamily-20071225.pdf (Дата звернення: дата звернення: 5.01.2020).
25. Spongy Castle by rtyley [Електронний ресурс]. URL:
https://rtyley.github.io/spongycastle/https://rtyley.github.io/spongycastle/ (Дата звернення: дата звернення: 7.01.2020).
26. Bouncy Castle. Bouncy castle crypto APIs. bouncycastle.org [Електронний ресурс]. URL: http://www.bouncycastle.org/http://www.bouncycastle.org/ (Дата звернення: дата звернення: 16.09.2018).
27. SensorEvent | Android Developers [Електронний ресурс]. URL:
https://developer.android.com/reference/android/hardware/SensorEventhttps://developer.android.com/reference/android/hardware/SensorEvent (дата звернення:Дата звернення: 11.01.2020).
28. Application Fundamentals | Android Developers [Електронний ресурс]. URL:
https://developer.android.com/guide/components/fundamentalshttps://developer.android.com/guide/components/fundamentals (Дата звернення: (дата звернення: 11.01.2020).

29. Robert G. Brown's General Tools Page URL: [Електронний ресурс]. http://webhome.phy.duke.edu/~rgb/General/dieharder.phphttp://webhome.phy.duke.edu/~rgb/General/dieharder.php (Дата звернення: дата звернення: 11.01.2020).
30. HCA van Tilborg. Fundamentals of Cryptology A Professional Reference and Interactive Tutorial [Електронний ресурс]. URL: https://hyperelliptic.org/tanja/teaching/cryptoI13/cryptodict.pdf (Дата звернення: дата звернення: 11.01.2020).
31. Christof Paar and Jan Pelzl. Understanding cryptography: a textbook for students and practitioners. Springer Science & Business Media, 2009.Understanding Cryptography [Електронний ресурс]. URL: http://khosach.info/store/msresource/it/book/Understanding-Cryptography.pdf (Дата звернення: дата звернення: 11.01.2020).
32. Niels Ferguson and Bruce Schneier. Practical cryptography, volume 23. Wiley New York, 2003. Practical Cryptography [Електронний ресурс]. URL: https://pdfs.semanticscholar.org/0627/1df8e235c6c5a722fb9ccd338addfe599983.pdf (Дата звернення: дата звернення: 11.01.2020).
33. John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator [Електронний ресурс]. URL: https://pdfs.semanticscholar.org/5c72/c84a8b29892855e55d3af1a57176bcfff5ee.pdf (Дата звернення: дата звернення: 20.09.2018).
34. Daniel J Bernstein. ChaCha, a variant of Salsa20. In Workshop Record of SASC, volume 8, 2008. ChaCha, a variant of Salsa20 [Електронний ресурс]. URL: http://cr.yp.to/chacha/chacha-20080128.pdf (Дата звернення: дата звернення: 11.01.2020).