КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра

за спеціальністю 122 Комп’ютерні науки

на тему:

ОГЛЯД І ДОСЛІДЖЕННЯ КІЛЬЦЕВИХ ЦИФРОВИХ ПІДПИСІВ

Виконав студент 4-го курсу

Костенко Микола Олегович _____
(Підпис)

Науковий керівник:

професор

Анісімов Анатолій Васильович _____
(підпис)

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

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

«____» _____202_ р.,

Протокол№___

Завідувач кафедри

В. М. Терещенко _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи – 52 сторінки, 8 рисунків, 1 діаграма, 3 таблиці, 19 джерел, один додаток.

КІЛЬЦЕВІ ПІДПИСИ, ГРУПОВІ ПІДПИСИ, ЦИФРОВІ ПІДПИСИ, КРИПТОСИСТЕМА, ГЕШ-ФУНКЦІЯ, ЕЛІПТИЧНА КРИВА.

Об’єктом дослідження і розробки є кільцеві підписи, за допомогою яких користувач може підписувати повідомлення залишаючись анонімним. Предметом роботи є програмний засіб, який реалізує схему кільцевого цифрого підпису.

Метою роботи є використання програмного засобу для вивчення роботи зв’язних кільцевих підписів, і порівняння їх продуктивності.

Методи розробки: методи обчислення геш-функцій, аналіз схем підписів, методи створення приватних і публічних ключів. Інструменти розробки: безкоштовне інтегроване середовище розробки PyCharm, для мови розробки Python 3.10.

Результат роботи: проведено дослідження розвитку кільцевих підписів та їх типів. Здійснено огляд наявних застосувань кільцевих підписів та аналіз слабких сторін. Розроблено власну реалізацію кільцевого підпису на основі еліптичних кривих. Проведено порівняння швидкості генерування кільцевого підпису та його верифікації з використанням різних еліптичних кривих.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

RS – Ring Signature, кільцевий підпис;

RSA - Rivest, Shamir, Adleman - Рівест, Шамір, Адельман;

LSAG – Linkable Spontaneous Anonymous Group Signature, зв'язний спонтанний анонімний груповий підпис;

SLSAG – Short Linkable Spontaneous Anonymous Group Signature, короткий зв'язний спонтанний анонімний груповий підпис;

MLSAG – Multi-layered Linkable Spontaneous Anonymous Group Signature, багатошаровий зв'язний спонтанний анонімний груповий підпис;

LRS – Linkable Ring Signature, зв'язний кільцевий підпис

ECDSA - Elliptic Curve Digital Signature Algorithm

RingCT – Ring Confidential Transactions, кільцеві конфіденційні транзакції

ECC –Elliptic Curve Cryptography, еліптична криптографія;

ECDLP – Elliptic Curve Discrete logarithm problem, проблема дискретного лографмування в групі точок еліптичної кривої;

PoW – proof of work, алгоритм консенсусу;

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта дослідження. З розвитком інформаційних технологій, питання безпеки стало важливою складовою в інтернет-середовищі. Сучасні веб-додатки зберігають велику кількість інформації про користувачів, що викликає занепокоєння в суспільстві. Забезпечення анонімності і конфіденційності в інтернет мережі потребує дослідження та впровадження нових методів в сферах криптографії та інформаційної безпеки.

Розробка нових типів цифрових підписів дає можливість створювати більш захищені цифрові сервіси. Одиним із таких типів є цифрові підписи, які забезпечують автентичність та непорушеність даних, що дуже важливо в фінансових та державних установах. Оригінальну ідею кільцевих підписів було запропоновано в 2001 році трьома розробниками: Рональдом Рівестом, Аді Шаміром та Яель Тауманом. Головною метою було зробити встановлення підписанта практично неможливим. Наразі існує безліч різних підвидів цифрових підписів, головним застосуванням яких є анонімізація криптовалютних транзакцій.

Актуальність роботи та підстави для її використання. Більшість сучасних криптовалютних систем досі залишаються небезпечними як для користувачів, так і для потенційних інвесторів. Можливість відслідковування транзакцій зводить на нуль будь-яку анонімність в таких криптосистемах як Bitcoin, що в свою чергу дає можливість зловмисникам знаходити потенційних жертв.

Реалізація кільцевих підписів у криптосистемах є доцільною з огляду на їхні безпекові переваги в порівнянні з іншими типами підписів. Тому виникає потреба в їх дослідженні задля мінімізації ризиків та подальшого розвитку криптографічних систем. Аналіз існуючих протоколів дозволяє визначити ефективність захисту від атак з боку зловмисників, а отже сприяє їх вдосконаленню.

Мета й завдання роботи. Метою кваліфікаційної роботи є аналіз математичної основи кільцевих підписів та огляд існуючих реалізацій. Також завданням є власна реалізація кільцевого підпису на основі еліптичної кривої та порівняльний аналіз швидкодії на різних еліптичних кривих. Для досягнення цієї мети поставлено наступні завдання:

- Огляд математичного підгрунтя цифрових підписів.
- Огляд основних типів кільцевих підписів та сфер їх застосування.
- Розробка альтернативної реалізації кільцевого підпису на основі еліптичної кривої.
- Порівняння кільцевих підписів з різними еліптичними кривими.

Об’єкт, методи й засоби розробки. Об’єктом дослідження є механізм електронних кільцевих підписів. Об’єктом розробки є власна реалізація кільцевого підпису на основі еліптичної кривої.

Засобом розробки є інтегроване середовище розробки для мови програмування Python під назвою PyCharm, а також інтерпритатор мови Python. Вибрана мова програмування має велику кількість вбудованих та допоміжних бібліотек які прискорюють процес розробки. Незважаючи на низьку швидкість мови Python, вона прекрасно підходить для наукових цілей через свій зрозумілий синтаксис та простоту використання.

Можливі сфери застосування. Програмна реалізація кільцевого підпису може бути імплементована в існуючі криптовалютні системи, або для персонального захисту інформації. Результати порівняння кільцевих підписів на основі різних кривих можуть бути використані для оцінки їх ефективності в тих чи інших сферах використання цифрових підписів

1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО КІЛЬЦЕВІ ПІДПИСИ

1.1 Означення цифрового підпису

Цифровий підпис – це аналог письмового підпису який використовується для підтвердження цілісності та автентичності документів чи іншої цифрової інформації. Зазвичай він дозволяє ідентифікувати власника підпису та гарантує, що підписаний ним файл був не змінений після його підписання. Підписання нового документу передбачає генерацію унікального підпису який формується за допомогою криптографічних алгоритмів. Для цього процесу використовується приватний ключ який знає лише сам підписант та публічний ключ. Отриманий унікальний код в процесі підпису і є цифровим підписом.

Перевірка цифрового підпису відбувається за допомогою публічного ключа, який відповідає підписанту і є напряму пов’язаним з приватним ключем. В процесі перевірки застосовується криптографічний алгоритм для порівняння вихідних данних з файлом. Якщо підпис буде змінений, або не буде відповідати вхідним данним, то файл не пройде перевірку. Таким чином, підпис буде вважатись недійсним, що свідчить про його зміни після підписання.

1.2 Важливість цифрових підписів

Цифрові підписи дають змогу підписувати електронні документи. Відправник може електронно підписати документ і надіслати його одержувачу, уникнувши потреби підписувати та відправляти фізичну копію поштою. Також цифрові підписи можуть слугувати для автентифікації особистості в інтернеті, часто це використовується для підтвердження банківських транзакцій. Цифрові підписи особливо важливі у сферах фінансів, медицини та права, де цілісність та точність документів є дуже важливою. Вони забезпечують юридично обов’язковий спосіб підписання електронних документів і дозволяють запобігти шахрайству. Саме

можливість безпечного і надійного способу перевірки автентичності підписанта гарантує, що не було внесено жодних змін в документ. Тому цифрові підписи є настільки важливими і необхідними для забезпечення цілісності електронних транзакцій та документів, особливо в тих галузях, де це є критично необхідним.

1.3 Поява кільцевих цифрових підписів

В 2001 році на міжнародній конференції теорії та застосування криптографії та захисту даних було вперше продемонстровано ідею кільцевих підписів. В своїй роботі під назвою «How to Leak a Secret»[1] автори: Рональд Рівест, Аді Шамір та Яель Тауман розробили повноцінну математичну модель. Основою для неї стало дослідження Девіда Чапмена та Євгена Вана Хейста під назвою «Group Signatures». Початкова ідея застосування кільцевих підписів була викладена в дещо жартівливій формі, і її зміст повністю відповідає назві наукової праці. На думку авторів, кільцевим методом можна було б підписувати таємні державні документи для їх розкриття, адже якщо документ був підписаний групою довірених осіб з уряду – не виникло б питань щодо правдивості документу. З іншої сторони, інформатор залишався в безпеці, адже ідентифікувати підписанта майже неможливо.

В цьому уявному прикладі будь-хто із читачів чи журналістів міг би перевірити оригінальність документу. З іншої сторони, зберігалась анонімність джерела інформації, адже навіть в суді ніхто б не зміг видати справжнього інформатора.

1.4 Групові та кільцеві цифрові підписи

Оскільки ідея групових підписів з’явилась на десять років раніше ніж кільцеві підписи, вона заклала основи кодування та верифікація для кільцевих підписів[2]. Незважаючи на явну схожість, вони принципово відрізняються ступенем безпеки.

Групові підписи створені для контролю доступу і дають можливість підписати документ будь-кому, хто належить чітко визначеній групі. Головною особливістю є існування адміністратора який може додає людей до тої чи іншої групи і знає секретні ключі учасників. Це дає йому можливість повністю деанонімізувати підписанта. Такий функціонал є досить корисним для корпоративних цілей, де начальник відділу завжди може перевірити своїх підлеглих.

Рисунок 1 - схема управління груповими підписами

Для кращого розуміння схеми групових підписів автори дослідження наводять приклад з офісним принтером (рис. 1). Отже існує група співробітників які мають право користуватись принтером, для цього в кожного є секретний ключ виданий начальником. Перед тим як відправити документ, він підписується секретним ключем, а вже принтер перевіряє чи належить людина до групи яка має дозвіл на друк. Важливим аспектом є те, що зберігається анонімність робітника який надіслав документ на друк, а отже зберігається анонімність для інших співробітників які належать до цієї групи чи сторонньої людини. Якщо ж хтось порушує певні правила, начальник може спокійно з’ясувати яка людина за цим стоїть. Всього

Девідом Чаумом було представлено чотири різні схеми групових підписів. На рисунку 1 ми розглянули першу схему, в якій адміністратор має необмежені права. Завдяки тому, що він знає секретні ключи кожного користувача він може сам підписувати документи, й відкриття підпису неможливе без нього. Більш того, кожен секретний ключ є одноразовим. Вирішенням цієї проблеми було б використання сліпих ключів, де кожен користувач обчислював би свій секретний ключ на основі рандомного числа яке було б згенеровано адміністратором. В інших схемах роль адміністратора обмежена і дозволяє відкривати цифрові підписи без залучення адміністратора до процесу. Загалом, на відміну від першої схеми, де можна використовувати різноманітні несиметричні способи шифрування даних, інші схеми були розроблені на двох припущеннях:

- Задача факторизації для великого числа, отриманого методом добутку великих простих чисел є неможливою задачою.
- Задача обчислення дискретного логарифма за модулем великого простого числа є неможливою.

Варто зазначити, що навіть зараз ці задачі є майже неможливими, адже досі ніхто не зміг вирішити задачу факторизації числа RSA-2048, нагорода за яку становить близько двохсот мільйонів доларів США.

Ці задачі також використовуються для протоколів підтвердження чи заперечення. Згідно нашим припущенням, ці задачі є неможливими для розв’язання, а отже їх можна використовувати для протоколів наведених вище, оскільки лише справжній підписант зможе підтвердити свій підпис не вирішуючи задачу обчислення дискретного логарифму чи факторизації числа. Схожим чином працює і протокол заперечення. Таким чином для відкриття підпису потрібно перевірити всіх користувачів, і один із них не зможе заперечити факт підписання за допомогою протоколу заперечення.

В кільцевих підписах автори повністю прибрали ідею контролю над групою.

Таким чином, була відкинута роль адміністратора, який може відкривати

підпис чи додавати людину до групи. Кожен користувач за припущенням вже є частиною групи, він сам створює свій публічний та приватний ключ. Більш того, користувачі які є підписантами в кільцевих підписах можуть і не знати, що їх публічний ключ був використаний. Ці публічні ключи можуть бути взяті з будь-яких джерел. Як приклад – ви генеруєте публічний ключ для підписання електронних платежів в онлайн банкінгу і його може використати інший підписант аби створити кільцевий підпис з вашим публічним ключем.

1.5 Математична формалізація групових підписів

Як було зазначено в розділі 1.4 різні схеми групових підписів використовують різні схеми шифрування даних, більш того, деякі з них використовують додаткові припущення які були описані в минулому розділі. Для початку визначимо спільні позначення, які будуть використовуватись в описі різних схем. Надалі A – адміністратор який керує групою, але він не завжди виступатиме як вірифікатор підпису, тому позначимо V як функцію верифікації.

1.5.1 Перша схема

В першій схемі для N учасників і A використовується схема сліпих ключів, для унеможливлення підробки підпису адміністратором. Таким чином будь-який користувач a буде мати свій власний s_a секретний ключ, де $a \in [1; N]$ який не буде відомий A.

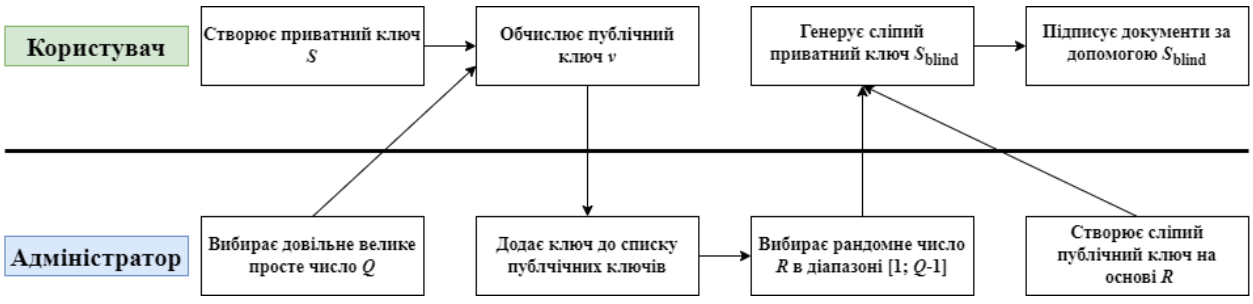


рисунок 2 - Створення сліпих підписів для першої схеми

Таким чином, кількість публічних ключів є лінійною та залежить від кількості користувачів. Вибір типу підпису є довільним і може

використовувати RSA[3], схему Ель-Гамаля[4] чи іншу несиметричну криптосистему. Для створення сліпих ключів s_{blind} та v_{blind} для кожного користувача потрібна взаємодія з *A*. Для прикладу може бути використана схема Ель-Гамаля, яка базується на складності обчислення дискретного алгоритму. Для числа *Q*, яке є довільним великим простим число, дивитись рис. 2 ми обчислюємо *g*, яке є первісним коренем мультиплікативної групи кільця лишків групи $\mathbb{Z}_Q$ за модулем *Q*. Тобто таке, що $g^{\varphi(Q)} \equiv 1 \bmod Q$, де $\varphi(x)=x\prod_{q|x}(1-\frac{1}{q})$ це функція Елейра[5] і *q* – просте число. Далі користувач а надсилає $v_a=g^{s_a} \bmod Q$ Адміністратору, що разом з *Q* та *g* буде публічним ключем користувача. В альтернативному випадку, адміністратор би сам здійснював створення приватного і публічного ключа і надавав би приватні ключі користувачам, але тоді б він міг би сам підписувати повідомлення. Для генерації сліпих ключів *A* генерує випадкове число R_a, де $R_a \in [1;Q\text{-}1]$ і обчислює новий сліпий публічний ключ $v_{blind} = g^{s_a R_a}$.
На основі R_a та v_{blind} користувач генерує $s_{blind}=s_a R_a (\bmod Q\text{-}1)$ який буде його сліпим приватним ключем. Надалі він буде використовуватись для підпису. Ця процедура створення ключів проводиться з кожним користувачем групи, аби він мав можливість підписувати документи.

Важливим є те, що підписані документи одного користувача будуть пов’язані лише сліпим ключем, що потребує повторної генерації сліпих ключів через певний проміжок часу, який визначає адміністратор. На відміну від сліпих ключів, ключ *s* кожного із користувачів буде статичним і на його основі наступні рази будуть генеруватись сліпі ключі.

Важливо зазначити, що роль адміністратора не є обов’язковою при використанні сліпих клчів, адже якщо кожен користувач матиме можливість самостійно доповнювати список публічних ключів групи, то данні обчислення зможуть проводитись локально користувачем і потім занесені в таблицю публічних ключів, розмір якої, як було зазначено вище, буде

лінійним і залежатиме від кількості користувачів і вона виконуватиме роль публічного ключа групи.

1.5.2 Друга схема

На відміну від попередньої схеми, в основі шифрування лежить перше припущення, наведене в розділі 1.4 про складність факторизації великих чисел, утворених добутком двох простих чисел. Отже для початку А генерує велике псевдопросте число N, для цього генеруються два великих простих числа p та q, де $N=p*q$. Генерація великих простих чисел відбувається за допомогою перевірки чисел ймовірнисними тестами простоти Міллера-Рабіна[6] та Бейлі-Померанца-Селфріджа-Уогстаффа. Проблема полягає в високій асимптотичній складності детермінових алгоритмів перевірки простоти, саме тому використання ймовірнисних тестів є виправданим. В якості N можна взяти одне із чисел RSA, бо вони задовольняють умовам схеми. Далі адміністратор вибирає односторонню функцію шифрування f, така функція робить зворотнє знаходження аргументу дуже складною задачею, для прикладу, ми можемо використовувати геш-функції сімейства SHA.

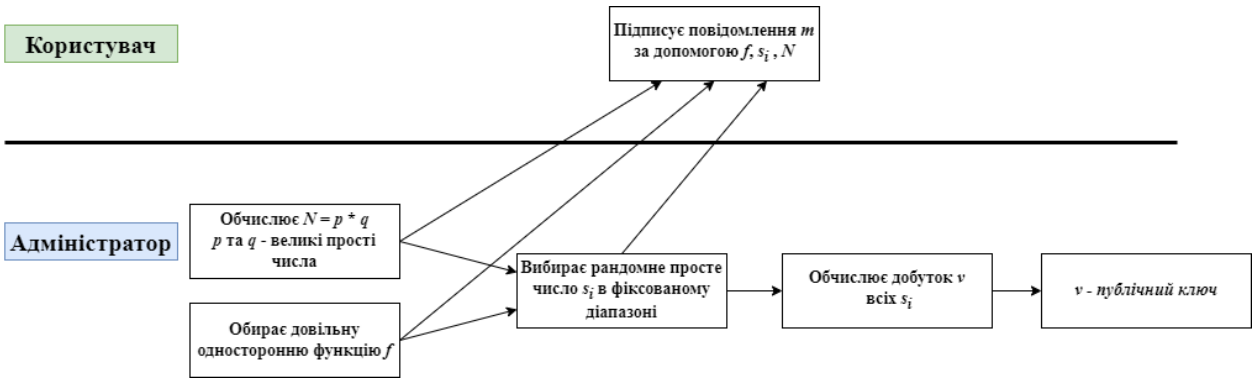


рисунок 3 - алгоритм створення ключів до другої схеми

Для кожного користувача із групи генерується велике просте ціле число s_i із інтервалу ($\sqrt{N}$;2$\sqrt{N}$-1). Публічний ключ групи $v = \prod_1^k s_i$, де k – кількість користувачів в групі, в цій схемі він поданий як добуток персональних простих чисел, на відміну від списку публічних ключів в розділі 1.5.1 .

В данному випадку s_i це приватний ключ який дозволяє не лише підписувати документи. Аби зменшити вплив адміністратора і дозволити самостійну

перевірку підписів, розроблено протокол підтвердження та заперечення. Завдяки цьому користувачі можуть самі довести один один одному чи є вони причетними до конкретного піпису чи ні. Підпис $sig(f, s_i, N, m)$, зображений на рис.3 формується шляхом обчислення $sig \equiv \left(f(m)\right)^{s_i} \bmod N$.

1.5.2.1 Протокол підтвердження

Однією з особливостей протоколу є максимальна безпека при перевірці. В його основі лежить припущення про неможливість обрахунку дискретного логарифму за модулем великого простого числа, адже в інакше протокол буде скомпрометований. Протокол належить до числа протоколів з нульовим розголошенням, і використовує бітову схему обов'язків для підтвердження. Завдяки цьому неможливо змінити обрахунки, і відповідь користувача буде правильна лише якщо він володіє приватним ключем.

Ідея протоколу полягає в тому, що якщо в нас є користувач J який заявляє, що знає значення s для вирішення рівняння $\alpha^s \equiv \beta \bmod N$, де α та $\beta \in \mathbb{Z}_N^*$, мультиплікативної групи утвореної за модулем великого простого числа N. Значення s виступає як приватний ключ, а a та B є загальновідомі. Ми припускаємо, що $s \in [a, a + B]$, де a та B є публічними. Тоді для підтвердження факту знання s, потрібно лише довести, що $s \in [a - B, a + 2B]$.

Опис ітерації протоколу:

1. Підтверджувач вибирає випадкове $r_1 \in [0, B]$, та обчислює $r_2 = r_1 - B$. Далі він обчислює значення $c_1 \equiv \alpha^{r_1} \bmod N$ та $c_2 \equiv \alpha^{r_2} \bmod N$ і надсилає їх в якості пари бітових рядків (bitString(c_1), bitString(c_1)).
2. Верифікатор вибирає випадкове значення $k \in \{0, 1\}$ і відправляє його підтверджувачу.
3. Залежно від значення k підтверджувач надішле:
   a. Якщо $k = 0$, то r_1 та r_2.
   b. Якщо $k = 1$, то надсилається r^*, що дорівнює значеню одного із виразів (r_1+s) чи (r_2+s), в залежності від того який із них належить інтервалу $[a, a+B]$. І таким чином відкриє відповідне повідомлення c^* із c_1 чи c_2 відповідно.
4. Верифікатор перевіряє повідомлення в залежності від k:
   a. Якщо $k = 0$, то він перевіряє, що $r_1 \in [0, B]$ та $r_1 - r_2 = B$, і що надіслана пара містить $a^{r_1} \bmod N$ та $a^{r_2} \bmod N$.
   b. Якщо $k = 1$, то він перевіряє чи $r^* \in [a, a+B]$ та чи задовольняє умові $\alpha^{r^*} \equiv \beta c^*$ одне із значень в парі, тобто $\alpha^{r^*} \equiv \beta \alpha^{r^*} \bmod N$.

В протоколі роль верифікатора є здебільшого пасивною, він надсилає лише один біт інформації в змінній *k*, це аналогічно підкиданню монетки. Зрозуміло, що коли $k = 0$ ми не отримуємо корисної інформації про *s*, а отже вірогідність *p* правдивості припущення, про знання приватного підпису підтверджувачем після однієї успішної ітерації протоколу складатиме $p = \frac{1}{2}$ і з кожною новою успішною ітерацією буде підвищуватись. Таким чином $p = 1 - \left(\frac{1}{2}\right)^i$, де *i* – кількість ітерацій протоколу.

Аби підтвердити підпис проілюстрований в розділі 1.5.2 користувачу який виступатиме в ролі підтверджувача потрібно довести, що він знає *s* яке задоволняє $sig \equiv (f(m))^{s_i} \bmod N$ для $sig(f, s_i, N, m)$, де sig та $f(m) \in \mathbb{Z}_N^*$, що $s \in (\sqrt{N}, 2\sqrt{N}\text{-}1)$ і $s|v$. Для зручності позначимо $M = f(m)$. Для підтвердження $sig \equiv (M)^{s_i} \bmod N$ і $s \in (\sqrt{N}, 2\sqrt{N}\text{-}1)$ використаємо ітеративний протокол наведений в цьому розділі. Після цього, для підтвердження того, що *s* є дільником *N* можна використати той факт, що підтверджувач знає *s*.
Аби зберігалось нульове розголошення використовується модифікована бітова схема обов'язків де верифікатор випадковим чином вибирає число $r \in [1, N]$ і надсилає повідомлення підтверджувачу $y \equiv (sig)^r \bmod N$.
У відповідь підтверджувач надсилає повідомлення $x \equiv (y)^{\frac{v}{s}}$.
Далі верифікатор надсилає *r* користувачу і відкриває його повідомлення.
Таким чином, лише якщо $\frac{v}{s} \in \mathbb{N}$, то $x \equiv ((M^s)^r)^{\frac{v}{s}} \bmod N$, тобто $x \equiv M^{vr}$, а отже користувач підтвердив підпис. В основі цього алгоритму лежить припущення про факторизацію великих чисел із розділу 1.4, яким і є публічний ключ *v*, який утворений добутком великих простих чисел.

1.5.2.2 Протокол заперечення

Іноді постає більш складне питання пошуку справжнього підписанта. Для цього може використовуватись протокол заперечення, який дає можливість користувачу довести іншим свою непричетність. Він також належить до протоколів з нульовим розголошенням і використовує алгоритми описані в минулих дослідженнях Девіда Чаума[5].

1.5.3 Третя схема

Ідея цієї схеми схожа з другою схемою, але в основі використовується припущення про складність факторизації великих чисел, утворених добутком великих простих чисел. Більш того, ця схема є сумісною з протоколами

заперечення і підтвердження описаних в розділі 1.5.2. В протоколі використовується схема модифікована схема RSA для генерації приватних ключів. Кожен користувач обирає два великих простих числа p та q і обчислюється число N їх композиції як на рис.4. Незмінною залишається одностороння функція, в якості якої, до прикладу, може бути довільна функція з сімейства геш-функцій SHA-2.

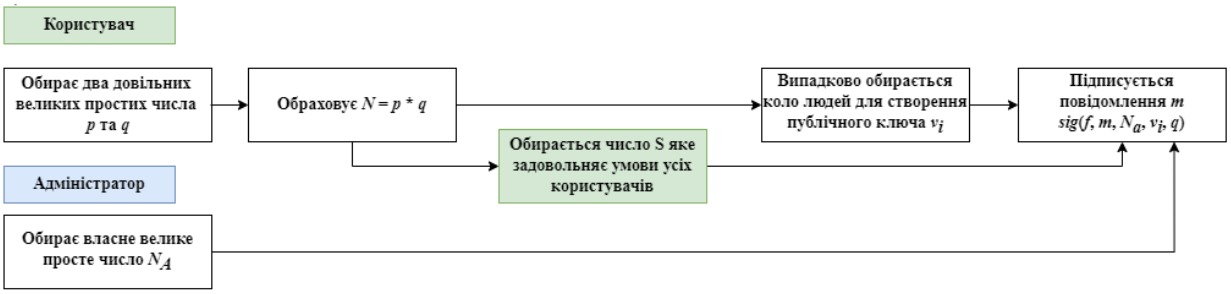


Рисунок 4 - алгоритм підписання повідомлення в третій схемі

Отримане число N вноситься кожним користувачем в публічну таблицю, яка в подальшому буде потрібна іншим користувачам в групі для генерації власних підписів. Після цього обирається число S, яке б задовольняло умовам:

- $q \in \left[\sqrt{S}, 2\sqrt{S}\text{-}1\right]$
- $p > 4\sqrt{S}$

Для кожного користувача. Це потрібно для сумісності з протоколами наведеними в розділах 1.5.2.1 та 1.5.2.2.

Генерація цифрового підпису *sig* для повідомлення *m* потребує перед початком вибрати випадковим чином підгрупу користувачів разом з підписантом з групи. Генерується публічний підпис v_i який є добутком всіх N із публічного списку які належать користувачам підгрупи. Таким чином $sig(f, m, N_a, v_i, q)$ представляє собою пару значень $\{(f(m))^q \bmod N_a, v_i\}$. Легко помітити, візуальну схожість цифрового підпису з минулою схемою. В обох з них публічні ключі сформовані шляхом добутку приватних ключів, але на відміну від минулої схеми, публічний ключ v_i буде різний для різних повідомлень. Це забезпечується випадковим вибором користувачів, які використовуються для підписання. З цього випливає очевидна теза, що збільшення кількості користувачів в групі збільшує унікальність v_i .

Для підтвердження авторства підпису, користувачу потрібно довести, що змінна q, використана для підпису документу, відповідає умові $q \in \left[\sqrt{S}, 2\sqrt{S}\text{-}1\right]$ та те, що p є дільником публічного ключа v_i . Власне для цього

можна використовувати протокол підтвердження з розділу 1.5.2.1 без його модифікацій і протокол заперечення для відповідного випадку.

1.5.4 Четверта схема

Четверта схема використовує друге припущення з розділу 1.4 про складність обчислення дискретного логарифму для створення цифрового підпису. Як і в третій схемі, група не є початково обмеженою, що дає змогу додавати нових користувачів в процесі. Це є незрівнянною перевагою, адже в реаліях використання групових підписів група може збільшуватись за рахунок нових користувачів. В перших двох схемах довелось би заново проходити процес ініціалізації групи, створюючи персональні ключі, що було б дуже великим дискомфортом і потребує прямого втручання адміністратора на всіх етапах цього процесу. Тому можливість гнучко модифікувати систему групового підпису в третій та четвертій схемі без потреби для повторної ініціалізації та зміни ключів кожного користувача є прикладом розумного застосування несиметричних криптосистем.

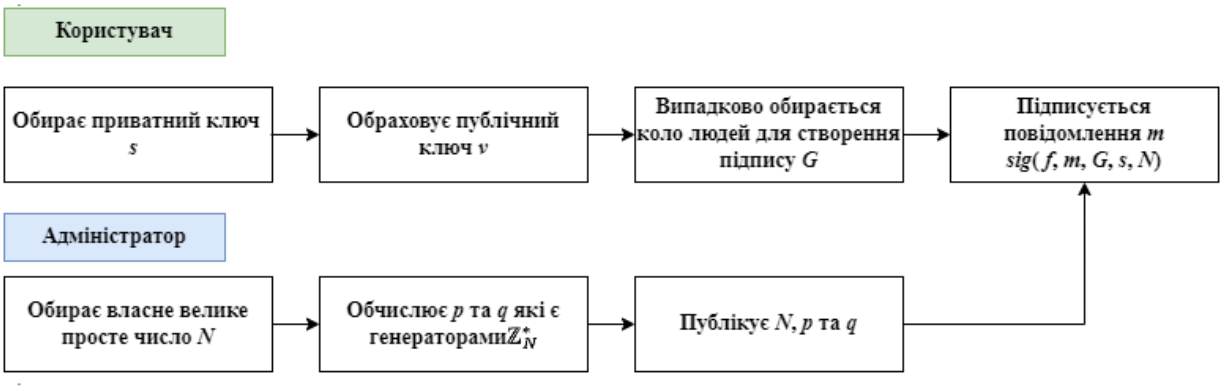


рисунок 5- алгоритм підпису повідомлення в четвертій схемі

Адміністратор на початку ініціалізації вибирає довільне велике просте число N по анології з минулими схемами і знаходить числа p та q, які є генераторами мультиплікативної групи кільця лишків за модулем N. Ці змінні є публічними і використовуватимуться для підпису повідомлень. Кожен користувач обирає своє власне велике просте число s яке надалі уде його приватним ключем. На його основі формується публічний ключ $v \equiv p^s \bmod N$. Підпис генерується схожим чином, як це описано в розділі 1.5.3, але має декілька відмінностей. Випадковим чином вибирається підгрупа людей G із групи користувачів і формується список лінійної довжини із $k_i \in G$. А повідомлення шифрується $S \equiv \left(f(m)\right)^s \bmod N$. Таким чином $sig(f, m, G, s, N) = sig(G, S)$. Очевидно, що протокол підтвердження з розділу 1.5.2.1 не може бути використаий без модифікацій. Важливим є те,

що для підтвердження підпису користувачу потрібно довести, що його публічний ключ належить до списку відкритих ключів групи G без розкриття будь-якої інформації про себе та приватний ключ.

В модифікованому протоколі підтвердження користувач вбирає довільні значення $r_1, r_2, r_3 \in [1, N\text{-}1]$ та $r_1^* \ldots r_G^* \in [1, N\text{-}1]$ і обчислює значення: $a \equiv \left(\frac{p}{f(m)}\right)^{r_1} q^{r_2} \bmod N$, $B \equiv (f(m))^{r_3} \bmod N$. Створюється G_r яка є випадковим чином відсортована група G і обчислюються $\gamma_i \equiv v_i q^{r_i^*} \bmod N$, для $i \in G_r$.

Далі ідентично до протоколу в розділі 1.5.2.1 використовується бітова схема забов'язань в якій підтверджувач надисилає обчисленні данні верифікатору, а той в свою чергу надсилає у відповідь випадкове значення нуля чи одиниці. В залежності від цього верифікатору надсилаються оригінальні значення r_1, r_2, r_3, r_i^* та G_r якщо біт дорівнював нулю, або r_1+s, r_2+s, r_3+s, та G_r^i. Зрозуміло що в першому випадку верифікатор просто перевірить правильність, а в другому перевірить чи виконуються умови:

- $BS \equiv (f(m))^{r_1+s} \bmod N$
- $a\gamma_{G_r^i} \equiv Sq^{r_2+s} \left(\frac{p}{f(m)}\right)^{r_3+s} \bmod N$

На цьому головні відмінності між протоколами закінчуються. Більше детально про це описано в дослідженні [3]. Протокол заперечення із розділу 1.5.2.2 є сумісний з цією схемою і не вимагає його модифікації для застосування.

1.6 Висновки щодо групових підписів

Дослідження групових підписів показало, що ідея, запропонована Девідом Чаумом дозволяє забезпечити конфіденційність та безпеку при обміні інформаціє в групі. Більш того, групові підписи є досить гнучким інструментом і прекрасно підходять для корпоративних потреб, адже мають можливість адміністрування. Наявність протоколів перевірки для більшості схем дає змогу оптимізувати цю процедуру і робить її децентралізованою. Головні переваги групових підписів:

1. Збільшення рівня безпеки серед користувачів, шляхом використання криптографічних схем в групових підписах
2. Простота реалзіації на практиці і можливість модифікації схем підписання з розподілом ролей.
3. Можливість підписання документа від імені групи задля підвищення довіри до документу в групі.

З іншої сторони, групові підписи мають безліч недоліків. Серед них є застарілі схеми шифрування, які вимагають використання ключів великої довжини, адже з розвитком інформаційни технологій, стає все складніше. Тому сучасні групові підписи дуже відрізняються від оригінальних схем представлених в 1991 році. Більш того, пошук підписанта повідомлення може бути досить ресурсозатратним, адже вимагає використання протоколу заперечення велику кількість разів. Іншою проблемою може бути існування адміністратора, адже він відіграє важливу роль у групових підписах і може поставити під загрозу безпеку всієї групи, якщо буде отримано доступ до його данних.

Зрештою, поява групових підписів відкрила новий підхід для забезпечення безпеки, і стала грунтом для створення кільцевих підписів, а імплементація нових модифікацій протоколів дає можливість розширювати сфери використання.

2. Огляд кільцевих підписів

В першому розділі було коротко описано ідею кільцевих підписів, а також формалізовано ідею та схеми групових підписів. Оглядаючись назад, головною ідеєю створення кільцевих підписів була саме ідея анонімності підписанта. В кільцевому підписі є власне підписант S, в якого є s – секретний ключ і v_s – публічний ключ. Позначимо людей, публічні ключі v_i яких використовуються для генерації підпису як N_i. Розглядаючи третю та четверту схему групових підписів в підрозділах 1.5, можна сказати, що вони можуть бути названі першими кільцевими підписами, адже підгрупа користувачів виступає в ролі кільця, в якому використовуються їх публічні ключі і приватний ключ підписанта. Та кільцеві підписи не потребуються статичної групи чи адміністратора, складних протоколів формування ключів та управління групою. Кільцеві підписи використовують спрощену схему, яка дозволяє максимально покращити анонімність підписанта. Схеми описані в наступних розділах були спеціально розроблені для кілець великих розмірів, адже таким чином визначення підписанта стає все більш складною задачею.

2.1 Формалізація кільцевих підписів

Оскільки на відміну від групових підписів, в кільцевих підписах не використовуються протоколи підтвердження чи заперечення авторства. Ідея кільцевих підписів припускає, що підписавши повідомлення, далі можливо лише підтвердити його незмінність використовуючи протокол з нульовим

розголошенням. Більш того, ми припускаємо, що публічні ключі v_i користувачів N_i пов'язані з використанням односторонніх функцій з відповідними секретними ключами s_i які нам не є відомі. Оскільки отримати ключі v_i не так і складно, користувач із N_i може навіть не здогадуватись, що ми використали його публічний ключ без відома. З іншої сторони, публічні ключі використовуються лише як спосіб ускладнення пошуку підписанта серед інших людей в кільці. Таким чином це не несе загрози персональним даним будь-кого із кільця окрім підписанта. Головними є два протоколи:

1. Підписання повідомлення, що в процесі створює кільцевий цифровий підпис $\text{sig}(m, s, v_s, v_1, v_2, \ldots v_i)$.
2. Верифікації оригінальності повідомлення m кільцевим підписом $\text{sig}(m, s, v_s, v_1, v_2, \ldots v_i)$. Протокол належить до класу протоколів з нульовим розголошенням і лишу підтверджує чи заперечує факт того, що повідомлення m було не зміненим.

Як було зазначено вище, користувачі можуть не знати, що їх публічний ключ було використано, без їх відома для створення кільцевого підпису, але й підписант може нічого не знати про них. В цьому випадку інформація чи взаємодія з іншими користувачами не є потрібною, це працює в обидва боки. На відміну від групових підписів, які використовують певні криптографічні припущення в своїй схемі, й існують лише через алгоритму складність певних обчислень, кільцеві підписи є стійкими навіть у випадку існування необмежених обчислювальних ресурсів. Навіть маючи в такому випадку будь-яку кількість повідомлень, підписаних однаковим s, персона S залишатиметься нерозгаданою. Таким чином алгоритм забезпечує анонімність на абсолютно новому рівні!

Важливим також є простота цих протоколів, незважаючи на те, що кільцеві підписи розраховані на велику кількість публічних ключів в кільці. В оригінальних схемах, основаних на RSA та несиметричному шифруванню під час підписання та верифікації відбувається не більше двох операцій піднесення числа в степінь по модулю певного числа на одного користувача.

2.2 LSAG підписи

В 2004 році група китайських вчених запропонувала нову ідею підписів під назвою Linkable Spontaneous Anonymous Group Signature[7]. Дослідження використовувало вже існуючі напрацювання в цій темі Девіда Чаума та дослідження Крамера про SAG підписи. Назва підпису висвітлює основні ідеї схеми:

1. Зв’язність, різні цифрові підписи, які були створені одним і тим же користувачем будуть пов’язані, таким чином, можливо перевірити чи були два різних документи підписані однією особою.
2. Спонтанність, формально під час створення підпису використовуються публічні ключі інших людей, за рахунок цього, незважаючи на зв’язність підписів, задача індентифікації підписанта є дуже складною. Група є спонтанною і в ній відсутня роль адміністратора, вона не є завчасно визначеною, а формується лише під час підписання.
3. Анонімність, ця ідея формується з зазначених вище ідей і природи безпекової моделі.

Загалом, сфера використання цих підписів не сильно відрізняється від звичайних групових підписів. Це в першу чергу використання у процесі електронних виборів. Групою в такому випадку будуть виборці, які мають завчасно зареєструватись до проведення основного етапу. Далі кожен виборець буде підписувати свій бюлетень LSAG підписом. Важливою деталлю в цьому є зв’язність підписів, таким чином електронні бюлетені людини з виборів мера і виборів президента будуть пов’язані між собою.

Питання безпеки не обмежується лише анонімністю підписанта. Захист від підробки підпису є не менш важливим. Автори дослідження використовують термін екзистенційної непідробності[8] для опису степеню захищенності. Адже декілька користувачів можуть створювати свої підписи використовуючи однакову групу, як у випадку з голосуванням. В цьому випадку, маючи повідомлення M і всі публічні ключі vi користувачів групи неможливо підробити підпис певної особи, адже для цього потрібно знати конкретний приватний ключ, який напряму зв’язаний з одним із публічних ключів vi. Отже маючи RO(V, M), де $V = \{v_1, v_2, \ldots v_i\}$ це список публічних ключів, а RO – ring oracle, що створює на основі цих данних підпис σ для якого sig(V, M, σ) = 1. Тоді довільний алгоритм Alg, який працює за поліноміальний час, використовуючи RO таким чином, що $\mathrm{Alg}^{\mathrm{RO}}(V) \rightarrow (V, M, \sigma_{Alg})$ задовольнить умову sig(V, M, σ_{Alg})=1 лише з надзвичайно низькою ймовірністю.

2.2.1 Анонімність LSAG

В схемі підпису є лише один приватний ключ, який і забезпечує відмінність між підписами з однаковою групою і текстом. Забезпечення секретності цього приватного ключа дає можливість гарантувати анонімність підписанту. Оскільки LSAG схема забезпечує цю анонімність, то формально якщо існує алгоритм Alg який використовуючи дані (V, M, S_k) та sig(V, M, σ),

згенероване користувачем i, де $S_k = \{s1, s2, \dots sk\} \subset S$ це підмножина приватних ключів людей, таких, що $Vk \subset V$, за поліноміальний час може:

$$\Pr[Alg(V, M, Sk, \sigma) \to i] \begin{cases} \in \left(\frac{1}{n\text{-}k} - \frac{1}{\mathrm{F}(j)}, \frac{1}{n\text{-}k} + \frac{1}{F(j)}\right), \\ \text{за умови } s_i \notin Sk \text{ та } 0 \le k < n\text{-}1 \\ > 1 - \frac{1}{F(j)} \end{cases}$$

для будь-якого довільного поліноміального $F(j)$.
Важливим є те, що користувач може довести іншим факт того, що підпис належить йому лише шляхом розкриття свого приватного ключа. Загалом, можна сказати, що імовірність знаходження справжнього підписанта складаю 1/n, де n це кількість користувачів в групі, незалежно від того, чи має він список приватних ключів, чи ні.

2.2.2 Зв'язність LSAG

Нехай в нас є список $V = \{v_1, v_2, \dots v_i\}$ публічних ключів, тоді два LSAG підписи зв'язні за умови існування алгоритму L, який дає однозначну відповідь на питання зв'язності двух підписів. Для цього L має виконуватись за поліноміальний час з наступними умовами:

$$\Pr[L(L, M_1, M_2, \sigma_1, \sigma_2) = 1 : i = j] \le f(p),$$

i

$$\Pr[L(L, M_1, M_2, \sigma_1, \sigma_2) = 0 : i \ne j] \le f(p)$$

для дуже великих p та будь-яких користувачів i та $j \in \{1, \dots. n\}$, повідомлень $M1$, $M2$ та відповідних $\sigma_1 \leftarrow \mathrm{sig}(V, M_1, s_i)$ і $\sigma_2 \leftarrow \mathrm{sig}(V, M_2, s_j)$. В данному випадку f є нехтовною функцією[9] для числа p, $f(x)$:N→Z , така що для будь-якого натурального числа c існує ціле $\mathrm{Z_c}$, що $x > \mathrm{Z_c}$ і $|f(x)| < \frac{1}{x^c}$.

2.2.3 Протоколи LSAG

В цьому розділі ми визначимо змінні для опису протоколів підписання та верифікації. Отже в нас є $G = \langle g \rangle$ яке є групою за модулем великого простого числа p, аналогічно до того, що було в розділі 1.5.2.
Далі обчислимо $H_1: \{0,1\}^* \to \mathbb{Z}_p$ та $H_2: \{0,1\}^* \to G$, де H_1 та H_2 це певні статичні криптографічні геш-функції. В кожного користувача $i \in \{1, 2\dots. n\}$ серед групи користувачів розміру n є свій публічний ключ $\mathrm{v_i}$ та секретний ключ v_i. Тоді $V = \{v_1, \dots v_n\}$ це список всіх публічних ключів.

2.2.3.1 Генерація підпису

Для підписання повідомлення $M \in \{0,1\}^*$ яке може бути представлено як послідовність біт, користувач і має виконати наступні кроки:

1. Обрахувати $h = \mathrm{H}_2(V)$ та $k = h^{s_i}$
2. Вибрати довільне число z $\in_R \mathbb{Z}_p$ і обрахувати $c_{i+1}=\mathrm{H}_1(V,k, M, g^z, h^z)$
3. Для кожного $j=[1, i\text{-}1]\cup[i+1, n]$, де $j \in$ N обирається $x_j \in_R \mathrm{Z_p}$ і обраховується $c_{j+1}=\mathrm{H}_1(V,k, M, g^{x_j}v_j^{c_j}, h^{x_j}k^{c_j})$
4. Обрахувати $x_i=z\text{-}s_ic_i \bmod q$

Таким чином послідовність $\sigma_V(M) = (c_1,x_1,\ldots,x_n,k)$.

2.2.3.2 Верифікація підпису

Верифікатора маючи LSAG підпис $\sigma_V(M) = (c_1,x_1,\ldots,x_n,k)$, повідомлення M та список публічних ключів V може верифікувати підпис, для цього потрібно виконати наступні кроки:

1. Обрахувати $h = H_2(V)$ і для $j = 1,\ldots,n$, порахувати $z_j=g^{x_i}v_j^{c_j}$ і $z_j^*=h^{x_i}k^{c_j}$. Тоді $c_{j+1}= \mathrm{H}_1(V,M, k, z_j,z_j^*)$ за умови $j < n$.
2. Перевірити чи $c_1 =^? \mathrm{H}_1(V, M, k,z_j,z_j^*)$. Якщо це вірно, то верифікація успішна, в іншому випадку висновок негативний.

Таким чином проходить верифікація повідомлення з використанням LSAG підпису, яка дозволяє впевнитись в цілісності та незмінності повідомлення *M*.

2.2.4 Коцепції вдосконалення LSAG

В розділі 2.2 було описано простий приклад використання LSAG підписів в електронних виборах. Подібні схеми проведення виборів в три етапи були запропоновані і раніше, як результат досліджень японських вчених практичного застосування криптографії для проведення масових виборів представлених в 1992 році. В цих схемах вибори складаються з реєстрації, голосування та підрахунку голосів. Під час реєстрації, користувачі надсилають пусті бланки, які не можуть бути відслідковані. Це досягається використанням сліпих ключів, ми вже розглядали їх в деяких схемах групових підписів, детальніше про їх створення описано в дослідженні. На етапі голосування кожен користувач надсилає свій бюлетень в зашифрованій формі в центр підрахунку голосів, загалом, в даному випадку їх може

існувати довільна кількість, адже це не грає ролі. Звісно, є різні варіанти перевірки бюлетенів. Важливим фактором є довіра користувачів до системи, тому в деяких схемах електроних виборів бюлетені після голосування стають публічними, аби людина могла перевірити, чи був зарахований її голос, чи ні. Зазвичай в таких схемах користувач після перевірки надсилає потрібний ключ чи певний параметр який дозволяє відкрити бюлетень, і таким чином голос зараховується. Ці міри потрібні для запобігання масових фальсифікацій, більш того, це значно підвищує довіру до виборів серед електорату.

Очевидним мінусом є складність подібної системи, вона вимагає багатьох дій від користувача, більш того, кількість інформації, яка піддається обробці значно вища, навіть велика кількість центрів обробки бюлетенів не допоможе, адже пара десятків мільйонів користувачів дуже легко зламає навіть підготовлену систему. Завдяки використанню LSAG підписів можна оптимізувати систему, зменшивши кількість етапів до двох. Залишається лише фаза голосування і підрахунку, без фази реєстрації. Ідея оптимізації полягає в тому, що прибравши фазу реєстрації, системі не доведеться обробляти десятки мільйонів пустих бюлетенів на початку. Далі наведено приклад цієї схеми:

1. Надійне і перевірене джерело генерує повідомлення. Для простоти можна взяти лише два M_{yes} та M_{no} , як у випадку референдуму. І далі ці повідомлення публікуються. Звісно може бути різна кількість повідомлень *M,* якщо ми будемо проводити президентські вибор, то будуть згенеровані n повідомлень, кожне із яких буде відповідати певному із n кандидатів. Таким чином, користувачу не потрібно видозмінювати бюлетень в залежності від кандидата, він просто буде обирати потрібний готовий йому бюлетень.
2. Кожен користувач Генерує LSAG підпис для повідомлення M_{yes} чи M_{no} і списку публічних ключів *V*.
3. Далі просто обраховується кількість голосів для кожного повідомленя.

Окремо важливо зазначити, що існує ймовірність того, що користувач захоче створити декілька підписів для одного кандидата, чи для декількох. Ці дії можна буде відстежити завдяки зв’язності LSAG підписів на відміну від SAG. Тому можна буде протидіяти подібним схемам.

Описана вище схема має забезпечувати анонімність підписанта бюлетеню під час виборів. Для оптимізації цього процесу, була прибрана реєстрація виборців, це означає, що алгоритм має компенсувати ті механіки, закладені в початкових схемах виборів. Вперш за все, для забезпечення анонімності мають використовуватись захищені канали зв’язку. Більш того, після

отримання підпису у виборчому центрі, він буде доданий до публічного бюлетеню виборців того чи іншого кандидата, при цьому жодні інші персональні данні не будуть розголошені. Це потрібно для того, аби кожен міг перевірити, що жодна людина не проголосувала декілька разів, використовуючи алгоритм описаний в розділі 2.2.2 . Для забезпечення більш ніж одних виборів можна використовувати різні схеми обрахунку геш-значення $h = H_2(V)$ для кожних виборів, просто модифікуючи геш-функцію H_2.

В описі стандартних схем голосування присутній такий етап, як відкриття бюлетенів, який напряму є відкриттям підписів з допомогою користувача. Як ми знаємо з першої схеми групових підписів, існування такого адміністратора є вразливою частиною в схемі і лише погіршує безпеку подібних систем. Тому в LSAG схемі виборів ця фаза відкриття підписів відсутня, користувачу не потрібно відкривати його, підпис вже має всі потрібні можливості для забезпечення чесного голосування.

Важливою частиною є те, що після голосування в підписанта не залишається жодних довідок чи просто файлів, які б підтвердили його авторство підпису. Це захищає підписанта від підкупу його голосу, адже жодна інша людина не зможе перевірити, чи справді він голосував за певного кандидата чи ні. Це ще одна перевага LSAG підписів. Користувач має лише обрати кандидата, все інше буде зроблено машиною, включаючи генерацію випадкових чисел, обрахунок геш-функцій та інших частин алгоритму. Загалом, схема може бути модифікована для підтримки вже інуючих виборчих протоколів в різних країнах, як наприклад вибір президента в США, де відбувається вторинне голосування від сенаторів кожного штату.

2.2.5 Розвиток LSAG

Через рік після виходу оригінального дослідження, було випущене нове дослідження від попередніх авторів, в якому вони вдосконалили LSAG підписи[10]. Власне в цьому дослідженні автори переосмислюють ідею LSAG, адже насправді вона є яскравим представником кільцевих підписів. В LSAG виконуються наступні пункти кільцевих підписів:

1. Відсутність адміністратора групи
2. Анонімність підписанта
3. Використання публічних ключів інших користувачів

На відміну від кільцевих підписів представлених Девідом Чаумом в 1991 році [1], LRS – Linkable ring signature, продовження LSAG, є зв’язок між

підписами, які були створені однією людиною. Це дуже важлива особливість, яка відкрила нові способи використання даної технології.

2.3 LRS підписи

В новій ітерації вже існуючої технології LSAG була переглянута безпекова модель схеми. Для цього було переглянуто ідею зв'язності підписів, з оглядом на майбутні перспективи використання. З новою схемою LRS покращився ще й захист самого підписанта, завдяки врахуванню практичних вразливостей перед хакерськими атаками. До всього іншого, додалась нова характеристика LRS, non-slanderability. Дослівний переклад терміну означає неможливість наклепу. Іншими словами, зв'язність двох підписів неможливо підробити аби підставити когось.

Із минулих розділів ми знайомі із цією характеристикою LSAG підписів, але в LRS схему було покращено, адже для імплементації в криптосистеми це є однією з найнеобхідніших безпекових потреб, захист від підробки. Довгий час, факт безпечності захисту від підробки в LRS залишався ніким не доведеним. Лише в дослідженні 2021 року цей факт було підтверджено[8].

За означенням, захист від підробки гарантує, що створення підпису, який був би зв'язаний з іншим підписом, згенерованим іншою людино, за допомогою алгоритму зв'язування, представленого в LRS та LSAG є неможливим.

2.4 RCT протокол

На основі LRS було створено окремий протокол RCT – Rind confidential transaction[11], він був створений для використання в криптосистемах і був імплементований в децентралізовану анонімну криптосистему Monero, в якій схожим чином до криптовалюти Bitcoin працює система нагородження користувачів в процесі майнінгу криптовалюти.

Процес майнінгу в різних криптосистемах відрізняється між собою, але в цілому використовується алгоритм PoW – proof of work. В таких системах є набір невиконаних транзакцій, які потрібно включити до нового блоку. Крім того, у блоку міститься попередній хеш (криптографічний ідентифікатор)

попереднього блоку. Майнери спробують знайти певне число, відоме як Nonce (подвійного використання велике число), таке, що коли його включено до блоку даних, отриманий хеш, застосований до всього блоку даних, задовольняє певну вимогу або має певну властивість, наприклад, починається з певного числа нулів.

В процесі хешування майнери послідовно випробовують різні значення Nonce і обчислюють хеш-функцію, наприклад, SHA-256 для всього блоку даних, який включає Nonce. Вони перевіряють, чи отриманий хеш задовольняє умови PoW.

Для перевірки блоку, якщо майнер знаходить такий Nonce, при якому отримується хеш, який задовольняє умови PoW, то він оголошує про успіх і виробляє новий блок, який містить цей хеш і всі невиконані транзакції в результаті посилаються на цей блок. В результаті майнер отримує нагороду.

2.4.1 Досвід використання LRS в RCT

При розробці блокчейну в криптосистемі Monero розробники вирішили використовувати при підписанні транзакції одноразові ключі, адже це мало б покращити анонімність користувачів. Для цього за основу було взято схему LSAG підписів, описану в розділі 2.2 . Це був абсолютно новий підхід, адже всі інші криптосистеми для анонімізації використовували так звані міксери. Міксери - це алгоритми які здійснювали безліч транзакцій між проміжними користувачами, розділяючи початкову суму на безліч малих шматочків. Це робило задачу викриття користувача значно складнішою. Чим більше транзакцій робив міксер, тим надійнішою була передача криптовалюти, і тим більше була комісія самого міксера.

Одним із ідейних мінусів міксера є централізоване управління, якщо воно буде скомпроментоване, то всі транзакції здійснені через міксер також будуть скомпроментовані.

Завдяки відсутності таких мінусів, RS є серйозним конкурентом, саме тому він обраний CryptoNote. Єдиною вразливістю RS є блокчейн аналіз на основі кількості відправленої криптовалюти. Очевидно, що при здійснені транзакції певна сума криптовалюти переміщується з одного гаманця, на інший. Таким чином, хакеру потрібно лише знайти в блокейні транзакцію, найближчу транзакцію яка містила таку ж кількість криптовалюти. Більшість існуючих допоміжних алгоритмів міксування є занадто об'ємними і негативно б вплинули на швидкість обробки транзакцій, що в свою чергу призвело б до збільшення комісії.

Велика кількість допоміжних транзакцій при такому підході призводить до того, що отримувачу для того, аби витратити цю валюту доведеться взаємодіяти з ними усіма, і буквально збирати всю отриману криптовалюту назад до купи. Більш того, оригінальний протокол кільцевих підписів від CryptoNote[12] створював підпис з кільцем заданого розміру, таким чином, при однакових обставинах, різниця в розмірі кільця могла створювати величезну різницю в захисті. Цей недолік був викликаний тим, що кількість підписів різного розміру була неоднорідна і умовно для $sig_i(v_i, A_i)$ була менша кількість потенційних підписів $sig_j(v_j, A_j)$, де v – публічний ключ, а $A_i=A_j$ – розмір кільця. Таким чином, потенційна безпека була вищою за реальну і залежала від вибраного A.

2.4.2 MLSAG підписи

Для виправлення недоліків RS підписів, описаних в попередньому розділі було створено MLSAG – Multilayered LSAG підписи для використання в RingCT протоколі. MLSAG також є кільцевим підписом, як і LSAG. Завдяки MLSAG, протокол RingCT дає можливість приховувати не лише відправника і отримувача, а й суму транзакції. Для створення MLSAG

підпису використовується модифікований LSAG алгоритм, який оптимізує використання пам’яті.

Генерація ключа модифікованого LSAG: Для списку публічних ключів V обирається певний індекс i такий, що $v_i \in V$ і створюються два ключі. Першим це пара значень $\{x, P\}$ і публічний ключ I. $xP=v_i$, де P це точка на еліптичній кривій, а x це одноразовий ключ для підпису. Тоді $I=xH_P(v_i)$, де H_p це геш-функція яка повертає точку на еліптичній кривій.

Підпис повідомлення M відбувається за допомогою згенерованих ключів. Для цього обираються два випадкових значення s та w_j такі, що $j \neq i, j \in \{1,\dots n\}$ такі, що s та $w_j \in \mathbb{Z}_q$, в полі еліптичної кривої. Далі обчислюються значення: $L_i=sP$, $R_i=aH_p(P_i)$, $c_{j+1}=h(M,L_i,R_i)$ в данному випадку функція h це геш-функція яка повертає значення в полі $\mathbb{Z}_q$. Далі для всіх i по модулю n обраховуються значення:

$$L_{j+1}=s_{i+1}H_p+c_{i+1}v_{i+1},$$

$$R_{i+1}=s_{i+1}H_p(v_{i+1})+c_{i+1}\cdot I,$$

$$c_{i+2}=h(M,L_{i+1},R_{i+1})$$

Обрахунки продовжуються доки не дійдемо до:

$$L_{j-1}=s_{i-1}H_p+c_{i-1}\ v_{i-1},$$

$$R_{i-1}=s_{i-1}H_p(v_{i-1})+c_{i-1}\cdot I$$

$$c_i=h(M,L_{i-1},R_{i-1})$$

Таким чином ми отримуємо список $C=\{c_1, c_2,\dots,c_n\}$. Далі визначимо $w_i=s-c_i\cdot x \bmod l$, де l це порядок еліптичної кривої, звідси отримуємо, що $s=w_i+c_i\cdot x \bmod l$, тоді:

$$L_i=sP=w_iP+c_ixP=sP+c_iv_i$$
$$R_i=sH_p(v_i)=w_iH_p(v_i)+c_iI$$

$$c_{i+1}=\mathrm{h}(M,L_i,R_i)$$

Таким чином, підписом буде:

$$\sigma(\mathrm{I}, c_1, w_1,w_2,\ldots,w_n)$$

на відміну від оригінального підпису від CryptoNote:

$$\sigma(\mathrm{I}, c_1, c_2,\ldots c_n, w_1,w_2,\ldots,w_n)$$

Така оптимізація економить ледь не половину розміру підпису!
Більш того, процес верифікації є досить простим, потрібно обрахувати всі R_i, L_i та c_i таким же чином, як це було зроблено під час створення підпису. Далі він перевіряє наступні рівності:

1. $c_{n+1}=c_1$
2. $c_{i+1}=\mathrm{h}(M, L_i,R_i)$ для всіх $i \in \{1,\ldots,n\}$

Якщо всі рівності виконуються – виріфікаціяє вважається успішною. Ця оптимізація була розроблена Коатару Сузукі в співпраці з його колегами, і представлена в 2002 році, за два роки до появи LSAG. Вона була імплементована лише в 2017 з переходом Monero на RingCT.

Загалом, на данний момент MLSAG є одним із найкращих кільцевих цифрових підписів які взагалі існують, як було сказано раніше, він прийшов на заміну більш простій реалізації кільцевих підписів, яка використовувалась в криптосистемі Monero до того, як було знайдено критичний баг, який міг повністю зруйнувати криптосистему, детальніше це буде описано в наступних розділах. Детальніше дізнатись про MLSAG та протокол, що його використовує RingCT можна в джерелі[11].

2.5 TRS підписи

Ідея TRS – traceable ring signature[13] дуже схожа з LSAG підписами, за основу були взяті групові підписи. Автори хотіли зробити відстеження підписів більш анонімним на відміну від групових підписів, де відстеження

підпису дає можливість деанонімізувати підписанта. Ідея генерації ключів в TRS була взята із досліджень Девіда Чаума про сліпі підписи[]. В концепті TRS немає ні адміністратора групи, ніяких завчасно визначених протоколів для обміну секретною інформацією серед користувачів групи, і подібних речей, притаманних груповим підписам. Таким чином, після створення сліпого підпису, його було б неможливо відстежити навіть його підписанту, більш того, важливим фактором є неможливість повторної підробки підпису. Це означає, що маючи сліпий підпис, неможливо на його основі згенерувати ще одни TRS.

Одна із найдраматичніших характеристик TRS підписів є так звана “double-spending”, що означає, що у разі підписання користувачем двох однакових сліпих підписів, він буде відстежуваний. Вона зіграла важливу роль у розробці криптосистеми Monero і ледь не зруйнувала довіру користувачів до кілцьевих підписів.

Для забезпечення такого особливості як відстеження підписів існує окремий тег $L=(id, V)$, де V це список публічних ключів учасників кільця і id це змінна для індентифікації, детальніше про її роль в протоколах верифікації буде описано пізніше. Після підписання повідомлення M за допомогою TRS підпису, створюється тег L, за допомогою якого можна здійснити верифікацію підпису будь-якій людині, без загрози бути деанонімізованим. Існування тегу є ключовою відмінністю між TRS та LSAG підписами. Для цього введені наступні вимоги до безпекових обмжень:

1. Публічне відстеження, будь-хто, хто підписав два різних повідомлення однаковим підписом матиме тег, за допомогою якого будь-хто може відстежити їх, тобто перевірити, чи були вони підписані однією людиною. Для цього потрібно мати мати повідомлення, підпис та тег.
2. Зв’язність тегів, кількість пов’язаних підписів за допомогою одного тега не може бути більша за кількість користувачів в кільці.

3. Анонімність, допоки користувач не підпише два повідомлення одним підписом з прив’язкою до тегу, його особистість буде неможливо відрізнити від будь-якого іншого користувача з кільця. Більш того, два підписи, з прив’язкою до різних тегів неможливо пов’язати між собою. Більш того, неможливо розпізнати навіть факт підписання двох різних підписів однією особою, якщо вони не прив’язані до одного тегу.
4. Невинність – якщо корисистувач не підписував два документи з прив’язкою до тегу, то його буде неможливо звинуватити в зворотньому, адже інша людина не зможе згенерувати TRS з прив’язкою до тегу окрім користувача. Ця схема буде працювати навіть якщо всі інші користувачі в кільці будуть скомпроментовані, окрім підписанта.

Ці міри захисту допомагають проти напівкільцевої атаки, а централізоване управління зведене до мінімуму. Єдиною централізованою системою залишається список публічних ключів кільця. Для побудови ефективного і простого TRS дослідники пропонують свою концептуальну схему, в основі якої лежать проблеми складності обчислення дискретного логарифму і припущення про проблему вибору Діффі-Хелмана.

2.5.1 Порівняння TRS аналогами

Автори дослідження пропонують свій погляд RS. Найбільша схожість є між LSAG(LRS) та TRS. Обидві схеми пропонують варіанти реалізації зв’язування окремих підписів між собою.На відміну від TRS підписів, RS підписи описані в дослідженнях [12, 14] вразливі до атаки в якій користувача можна звинуватити в у “double spending”. Наступні дослідження LSAG привели до створення SLSAG – short LSAG [15]. Особливість SLSAG в порівнянні з TRS є скорочення кількості комунікацій для створення підпису, завдяки цьому при великій кількості користувачів в кільці це дозволяє отримати кращу оптимізацію ніж в TRS. За ці переваги SLSAG розплачується рівнем безпеки, бо для реалізації схеми використовується довірений об’єкт,

що є потенційно небезпечним. Схема є вразливою до випадків, коли об’єкт є скомпроментований. Таким чином реалізація TRS є простіша і безпечніша за SLSAG.

2.6 Огляд кільцевих підписів в CryptoNote

В 2013 році анонімний користувач під нікнейном Nicolas van Saberhagen запропонував свою криптосистему побудовану на кільцевих підписах. Існуюча криптосистема Bitcoin хоч і була прогресивна, та не забезпечувала анонімності, частково цю тему було згадано в розділі 2.4.1 . Проблемими Bitcoin є його статичність та неможливість імплементації оновлень допоки всі користувачі мережі не здійснять локальне оновлення. Саме безпекові недоліки в Bitcoin стали причиною для розробки нового способу підписання транзакцій. Нещодавні новини все більше підтверджують тезу, про недостатню безпеку Bitcoin: “Невідомий хакер використав особливості як відбувається підпис і формування блокчейну в мережі Bitcoin для індентифікації 986 гаманців, що належали головному розвід управлінню(ГРУ), службі ззовнішньої розвідки(СЗР) та федеральній службі безпеки(ФСБ)”[16]. Більш того, хакер витратив кількість криптовалюти, еквівалентну трьомстам тисячам доларів. Використання кільцевих підписів в CryptoNote мало вирішити наведені вище проблеми в майбутній криптосистемі. Для цього проведення транзакцій має бути прихованим від сторонніх очей.

Питання відслідковування транзакцій в данному випадку є наріжним каменем, на якому побудована данна схема. Для повної анонімності криптосистема має відповідати критеріям:

1. Невідслідковність, що означає, що вірогідність авторства має бути рівновірогідно для всіх потенційних підписантів.
2. Незв’язність, що означає, що будь-які дві транзакції створені одним користувачем будуть не зв’язні.

На відміну від з’язних підписів в CryptoNote використовують одноразові адреси, завдяки чому транзакції відповідають критеріям описаним вище. Таким чином, адреса кожної транзакції є унікальна і створюється на основі данних отримувача і випадкових данних, згенерованих відправником.

2.6.1Одноразові кільцеві підписи

В основі кільцевих підписів в CryptoNote лежать одразу декілька типів різних кільцевих підписів. Загалом, як і в звичайних RS протокол містить стандартні функції генерації ключів, підпису і його верифікації. Більш того, є окрема булева функція зв’язності, яка визначає присутність зв’язку між ними:

1. Генерація ключів, яка на виході видає пару (P, x) і публічний ключ I.
2. Функція підписання повідомлення M використовує список публічних ключів V і пару $\{P, x\}$ для генерації підпису σ і кільця.
3. Верифікація повідомлення M, кільця R та σ видає відовідь на питання, чи було оповідомлення змінено чи ні
4. Функція перевірки зв’язності.

2.6.2 Формалізація схем підпису і верифікації

1. Генерація ключів. Для цього вибирається довільний публічний ключ $x \in [1, q\text{-}1]$ і обраховуються два значення, $I = xH_p(P)$ і $P = xG$, де I це публічний ключ який зветься ключем-образом і v_i це публічний ключ.
2. Підписання повідомлення M одноразовим кільцевим підписом відбувається за допомою протоколу з нульовим розголошенням. Із списку публічних ключів V обирається список $V^* \subset V$, далі з використанням пари значенння $\{P, x\}$ і ключа I. Виділимо публічний ключ $P_s \in \{P,x\}$ та V^*, це публічний ключ, що відпоідає приватному ключу x. Далі користувач обирає $\{s_i \mid i=\{0, 1,\ldots, n\}\}$ та $\{w_i \mid i=\{0, 1,\ldots, n\}, i \neq s\}$ і виконує наступні обчислення:

$$L_i=\begin{cases}s_iG, & \text{якщо } i=s\\ s_iG+w_iP_i & \text{якщо } i\neq s\end{cases}$$

$$R_i=\begin{cases}s_i\mathrm{H_p}(P_i) & \text{якщо } i=s\\ s_i\mathrm{H_p}(P_i)+w_i\mathrm{I} & \text{якщо } i\neq s\end{cases}$$

Наступним кроком обраховується константа:

$$c=\mathrm{H_s}(M, L_1, L_2,\ldots,L_n,R_1, R_2, \ldots,R_n)$$

Далі для остаточного результату обраховуються значення:

$$c_i=\begin{cases}w_i, & \text{якщо } i\neq s\\ c-\sum_{i=0}^{n} c_i \bmod q & \text{якщо } i=s\end{cases}$$

$$r_i=\begin{cases}s_i, & \text{якщо } i\neq s\\ s_s-c_sx \bmod q & \text{якщо } i=s\end{cases}$$

Таким чином отримуємо RS $\sigma=(\mathrm{I}, c_1,c_2,\ldots,c_n, r_1, r_2, \ldots, r_n)$

3. Верифікація відбувається шляхом зворотніх обчислень:

$$\begin{cases}L_i^*=r_iG+c_iP_i\\ R_i^*=r_i\mathrm{H_p}(P_i)+c_i\mathrm{I}\end{cases}$$

Тоді підпис буде верифіковано якщо виконуватиметься наступна рівність:

$$\sum_{i=0}^{n} c_i=^{?}\mathrm{H_s}\left(M,L_0^*,L_1^*, \ldots,L_n^*, R_0^*, R_1^*, \ldots,R_n^*\right) \bmod q$$

4. Перевірка зв’язності відбувається перевіркою I. Якщо I використовувався до цього в інших підписах, то підписи будуть зв’язані. Таким чином, при повторному використанні I підписи пов’язуються між собою і для перевірки підписант за допомогою відомих коефіцієнтів r_i та c_i доводить, що він знає такий x, що як мінімум для одного P_i виконується рівність:

$$\mathrm{H_p}(P_i)=\mathrm{I}\cdot x^{-1}$$

Таким чином ніхто не може відновити публічний ключ за допомогою ключа-образу для індентифікації користувача. Важливим при цьому є

те, що неможливо підписати два різні повідомлення з однаковим приватним ключем *x*, але з різними I. Саме тому вони йдуть в парі.

2.6.3Вразливість кільцевих підписів в CryptoNote

Як було зазначено вище в описі криптосистеми, вона була побудована з використанням еліптичних кривих. В якості найбільш перспективної еліптичної кривої була вибрана ed25519. Данна еліптична крива задоволняє стандарти безпеки ECDLP – elliptic curve discrete-logarithm problem, а також стандарти ECC – elliptic curve cryptography. Важливою особливістю цієї еліптичної кривої, в порівннянні з багатьма існуючими еліптичними кривиє є те, що кофактор кривої h = 8. Ця особливість призвела до серйозної діри в безпеці.

Для безпеки приватного ключа використовується його ключ-образ I, описаний в минулих розділах. Він використовується як для верифікації повідомлення, так і для запобігання такому явищу як “double spending”. Головною ж проблемою стало те, що в CryptoNote ключ-образ це просто точка на еліптичній кривій. З розділу 2.6.2 ми знаємо, що верифікація відбувається шляхом скалярного множення I, тобто точки на еліптичній кривій ed25519. Завдяки тому, що кофактор еліптичної кривої дорівнює восьми, відписант може підбирати точки на еліптичній кривій, додаючи дуже незначні і маленьки точки на еліптичній кривій до I. Цим він може контролювати значення скалярного добутку і обходити заборону на “double-spending”.

Іншими словами – користувач може безлімітно подвоювати його гроші, просто через те, що на еліптичній кривій існують точки, які є еквівалентними при використанні в протоколі верифікації. Ця вразливість зачіпала не тільки криптосистему Monero, а й інші криптосистеми, які використовували Monero як основу[14]. Це зайвий раз нагадує важливість комплексних досліджень

при проектуванні схем кільцевих підписів, а й електронного голосування та удь-яких інших криптографічних сфер.

3. Програмна реалізація LRS

Для реалізації LRS підпису використаємо дані, описані в розділах 2.2 та 2.3 . Зазначені схеми кільцевих підписів використовують еліптичні криві для пришвидшення процесу шифрування, і для кращого криптографічного захисту. В якості середовища для розробки було обрано мову Python 3.10, адже вона вже має безліч потрібних для розробки бібліотек і є найкращим вибором для різноманітних наукових досліджень.

3.1 Підхід до розробки

Важливим аспектом розробки кільцевого підпису є можливість використання різних еліптичних кривих, з можливістю дослідження швидкодії функцій підпису та верифікації з використанням різних еліптичних кривих. Для цього був використаний ECDSA алгоритм генерації ключів[17]. Безпека ECDSA ключів базується на проблемі складності обчислення дискретного логарифму в групі точок еліптичної кривої. Це робить ці ключі більш захищеними ніж ті, які створені на основі RSA чи простого дискретного логарифму[18]. Таким чином, 256 бітний ECDSA ключ буде більш захищений ніж навіть 2048 бітне RSA число.

Загалом, ECDSA є стандартом індустрії та використовується в таких протоколах як openSSL, Cryptolib та багатьох інших. Для розробки на мові python було використано інтегроване середовище розробки PyCharm, адже це значно спрощує написання коду і відстеження помилок, а внутрішня підтримка контролю версій та вбудований debugger зменшують час, на їх пошук. PyCharm дозволяє створювати окрему віртуальну оболочку, завдяки

чому дуже легко керувати пакетами і тестувати готовий алгоритм підпису[19].

3.2 Основні програмні компоненти

Програму можна поділити на дві частини:

1. Створення ECDSA підписів
2. Створення кільцевого підпису

Для створення ECDSA підписів в програмі є класи:

1. ECDSA – це клас який обчислює приватний і публічний ключ.
2. EllipticCurve – це клас який визначає взаємодію з еліптичними кривими.
3. Curves – це файл який містить в собі характеристики еліптичних кривих.
4. Gen – це клас-декоратор який дає змогу легко генерувати потрібну кількість ключів.
5. Keys – це файл який містить характеристики ключів і дає змогу взаємодіяти з ними.

Для створення кільцевого підпису є два класи:

1. LinkableRingSignature – це клас який реалізує всі функції LRS підписів
2. Point – це допоміжний клас який містить характеристики точки на еліптичній прямій і дозволяє легко з ними взаємодіяти.

3.3 Вибір еліптичних кривих

Для створення ECDSA підписів вибрано 6 еліптичних кривих, кофактор яких дорівнює одиниці, таким чином підпис не буде вразливим до “double spending” і буде відповідати заявленим характеристикам безпеки.

Цими кривими є NIST192p, NIST224p, NIST256p, NIST384p, NIST521p та SECP256k1. Завдяки різній довжині ключів, можна оцінити швидкість роботи алгоритму в різних ситуаціях, більш того, ми можемо порівняти швидкість конкурентних еліптичних кривих SECP256k1 та NIST256p на практиці. В цей список не ввійшла крива ed25519 з причин, описаних в розділі 2.6.3, незважаючи на її високу захищеність в інших сферах криптографії.

3.4 Генерація ключів

Важливим аспектом є те, що існує потреба в зберіганні публічних ключів для повторного використання і вивантаження їх з оперативної пам’яті комп’ютера для інших обчислень. Більш це потрібно для обміну публічними ключами з іншими людьми, завдяки чому можна будувати децентралізовану систему з кільцевим підписом. Також це має досить вагоме практичне значення при тестуванні, адже дозволяє зберегти данні, чи використати їх для бенчмарку на іншій електронно-обчислювальній машині з метою оцінки оптимізації алгоритму підпису і порівняння швидкодії на різних платформах.

В якості файлу для зберігання ключів використовується розширення “.pem”, це розширення файлу дозволяє безпечно надсилати його іншим користувачам, адже ключ може шифруватись для забезпечення його цілісності. В моєму випадку використовується хеш-функція сімейства SHA для обчислення хеш-значення публічного ключі і його подальшого шифрування. Як закодований публічний ключ можна побачити на рисунку 6.

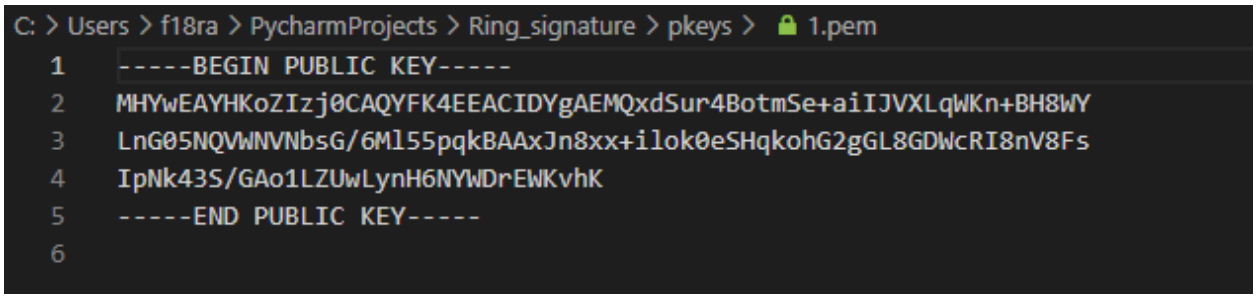


Рисунок 6 - приклад публічного ключа збереженого в .pem

Загалом, для кільцевого підпису потрібна велика кількість публічних ключів, тому для цього і використовується клас gen, який зберігає всі публічні ключі в .pem.

```
C:\Users\f18ra\PycharmProjects\Ring_signature\venv\Scripts\python.exe C:/Users/f18ra/PycharmProjects/Ring_signature/main.py
All generation time for 100 keys: 0.0940854549407959
All generation time for 100 keys: 0.09008264541625977
All generation time for 100 keys: 0.09108233451843262
All generation time for 100 keys: 0.10709691047668457
All generation time for 100 keys: 0.09608721733093262
Average time for key generation: 0.0956869125366211

Process finished with exit code 0
```

Рисунок 7 - приклад генерації 100 публічних ключів

Загалом, нижче приведена таблиця 1 середньої швидкості генерації 100 публічних ключів і їх збереження в файли для різних еліптичних кривих в секундах, приклад тестування генерації ключів наведений на рисунку 7. Детальна інформація по кожній кривій описана нижче в таблиці 1.

Таблиця 1 – час витрачений для генерації ключів

	NIST192p	NIST224p	NIST256p	NIST384p	NIST521p	SECP256k1
1	0.094085	0.110100	0.131119	0.214193	0.417379	0.129117
2	0.090082	0.098088	0.110100	0.185175	0.324294	0.121109
3	0.091082	0.098089	0.110100	0.182157	0.361328	0.116105
4	0.107096	0.098088	0.109098	0.198180	0.333303	0.109098
5	0.096087	0.1030936	0.111100	0.211199	0.314293	0.111100
μ	0.095686	0.101492	0.114303	0.198181	0.350119	0.117506

Згідно таблиці 1, де μ - середній час, різниця у швидкості між NIST256p та SECP256k1 при створені ключів мінімальна, але NIST256p загалом стабільно показує швидший результат генерації середньому на 3-5%. Також можна помітити, що швидкість генерації NIST192p є на 15% вищою за NIST256p, але враховуючи той факт, що розмір ключа NIST192p на 25% менший, і є не

таким захищеним - сенсу практичного використання NIST192p майже немає. NIST521p показує найдовший час генерації, але загалом забезпечує найкращу безпеку серед усіх шести еліптичних кривих.

3.5 Генерація і Верифікація підписів

Окремим пунктом є швидкість швидкість підписання і верифікації підпису, вона обернено пропорційно залежить від розміру кільця і довжини ключа. Оскільки python є мовою, що інтерпритується, а отже швидкість обчислень буде меншою ніж на мовах C та C++. Через це, обчислення відбуваються довше і впираються в технічні обмеження стаціонарних комп’ютерів. Для кращої репрезентативності тестів, кількість публічних ключів в кільці була обмежена до 10, але кількість ітерацій збільшена.

```
signature generation time:0.9929101467132568
verification time:        1.0229296684265137 Verificated
signature generation time:0.9959053993225098
verification time:        1.028935194015503 Verificated
signature generation time:0.9909005165100098
verification time:        1.0249316692352295 Verificated
signature generation time:0.998908519744873
verification time:        1.061964750289917 Verificated
signature generation time:1.0609650611877441
verification time:        1.0819826126098633 Verificated
signature generation time:1.0869958400726318
verification time:        1.091984748840332 Verificated
signature generation time:1.082984447479248
verification time:        1.0539579391479492 Verificated
signature generation time:1.020927906036377
verification time:        1.0419394969940186 Verificated
signature generation time:1.0229222774505615
verification time:        1.0539653301239014 Verificated
signature generation time:1.00191068649292
verification time:        1.0359416007995605 Verificated
======================================================
Medium signature time:  1.0255330801010132
Medium verification time:  1.049853301048279

Process finished with exit code 0
```

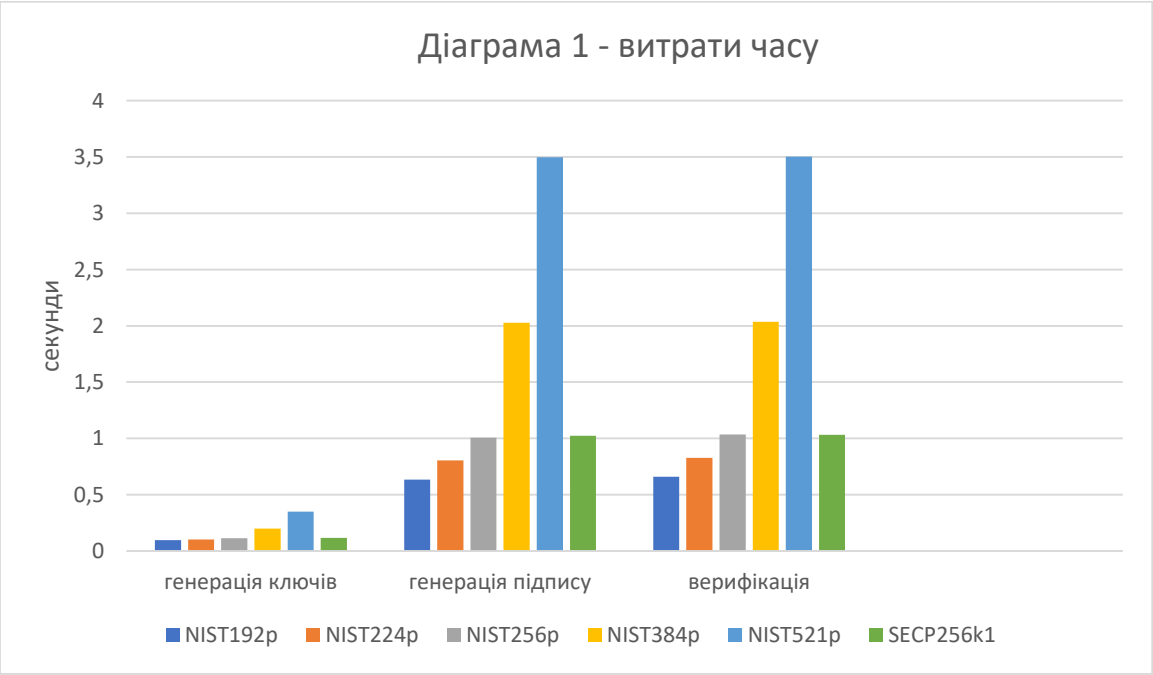
Рисунок 8 - приклад бенчмарку створення та верифікації кільцевого підпису

Нижче будуть наведені дві таблиці, перша описує швидкість генерації підпису, а друга – швидкість верифікації підпису, значення в таблицях вказані в секундах. Цей параметр є більш важливим ніж генерація ключів, адже ключі генеруються один раз, а кожен новий підпис вимагає більше складних обчислень.

Таблиця 2 – генерація підпису

	NIST192p	NIST192p	NIST256p	NIST384p	NIST521p	SECP256k1
1	0.637579	0.804739	0.996906	2.084894	3.516195	1.089998
2	0.635577	0.806733	0.993910	2.010827	3.486168	0.997906
3	0.631574	0.804731	0.991909	2.023839	3.501182	1.002911
4	0.635569	0.793721	1.006922	1.992803	3.471155	0.989897
5	0.635577	0.807741	1.044942	2.018842	3.511191	1.037950
μ	0.635175	0.803533	1.006918	2.026241	3.497178	1.023732

Загалом, результати в таблиці 2 показують досить схожу картину з результатами тесту генерації ключів в таблиці 1. Зберігається однакове співвідношення між швидкістю генерації підписів і ключів. Нижче наведено діаграму яка використовує дані з таблиць 1, 2 та 3 для ілюстрації середнього часу, який витрачається програмою для обрахунку наступних дій:



Ця діаграма також підтверджує той факт, що верифікація в LRS відбувається за однаковий час, що і генерація підпису. Таким чином, це твердження було підтверджено на практиці з використанням різних еліптичних кривих. Також можна виділити факт однакової швидкості обрахунків з використанням

NIST256p та SECP256k1. Загалом, питання вибору між цими двома кривим досі залишається відкритим.

Таблиця 3 – верифікація підпису

	NIST192p	NIST224p	NIST256p	NIST384p	NIST521p	SECP256k1
1	0.655596	0.825750	1.018925	2.063876	3.520191	1.021928
2	0.659599	0.827752	1.020920	2.043857	3.486168	1.029936
3	0.661601	0.833757	1.066962	2.035850	3.518205	1.029935
4	0.665612	0.821747	1.017917	2.007832	3.481163	1.016924
5	0.655596	0.833757	1.054959	2.020836	3.503175	1.0589549
μ	0.659601	0.828553	1.035936	2.034450	3.50178	1.031536

3.6 Результати реалізації

На основі огляду існуючих переваг і недоліків кільцевих підписів було реалізовано LRS, який використовує ECDSA для створення приватних та публічних ключів, адже це забезпечує вищий рівень безпеки на відміну від RSA. В результаті отриманих тестів на шести різних еліптичних кривих, вдалося оцінити швидкість створення підпису, генерації ключів та верифікації.

Варто зазначити, що NISTP256p та SECP256k1 показали однаково непоганий результат і є кращими за NIST192p та NIST224p, адже вони не можуть компенсувати низький захист високою швидкістю виконання. В той

же час, NISTP256p та SECP256k1 це стандарти індустрії які й досі використовуються в криптосистемах, як наприклад Bitcoin. З іншої сторони, існують NIST384p та NIST521p які дають кращий захист, і тому користувач може використати їх, коли для нього надзвичайно важлива безпека.

ВИСНОВКИ

Досліджуючи історію кільцевих підписів, було розглянуто безліч різноманітних алгоритмів кільцевих підписів, починаючи від самих простих прикладів представлених в якості групових підписів, і закінчуючи сучасним MLSAG підписом, який використовується криптосистемах. Під час огляду кільцевих алгоритмів було висвітлено безпекові переваги кільцевих підписів, й проблеми з якими приходиться боротись розробникам.Були розглянуті найпоширеніші алгоритми кілцьевих підписів:

1. LSAG підписи
2. LRS
3. TRS
4. MLSAG підписи

Проведене дослідження вразливості кільцевого підпису в CryptoNote є важливим для подальшого розвитку протоколу RingCT, який перейшов на вдосконалену версію і тепер використовує MLSAG підпис.

Реалізація власного кільцевого підпису на основі досвіду CryptoNote є досить простою і ефективною у використанні, адже використовує ECDSA для генерації менших і більш надійних ключів ніж RSA чи криптосистеми, в основі яких лежить проблема обчислення дискретного логарифму. Незважаючи на низьку швидкість мови Python в порівнянні з мовами C та C++, кільцевий підпис є цілком захищеним і може використовувати різні еліптичні криві для генерації ключів.

Загалом, кільцеві підписи поступово розвиваються ще з 1991 року, й до сих пір. Лише не так давно з’явились криптосистеми з підтримкою кільцевих підписів. Більш того, схеми електронного голосування з використанням кільцевих підписів також не стоять на місці і прогресують разом з розвитком нових схем кільцевих підписів. Використання еліптичних кривих в кільцевих підписах дало ідеї друге дихання, і зараз ми можемо спостерігати успішне використання MLSAG в криптосистемі Monero. На мою думку, кільцеві

підписи чекає ще більший успіх ніж сьогодні, адже вони поєднують в собі простоту обчислень і надзвичайний рівень захисту.

СПИСОК ДЖЕРЕЛ

1 Ronald L. Rivest, A. S. (2001). How to leak a secret. In C. Boyd (Ed.), *Advances in Cryptology — Asiacrypt 2001* (pp. 552-565). Berlin: Springer-Verlag. Retrieved from https://people.csail.mit.edu/rivest/pubs/RST01.pdf

2 Heyst, D. C. (1991). Group signatures. In D. Davies (Ed.), *Advances in Cryptology — Eurocrypt '91* (pp. 257–265). Berlin: SpringerVerlag. Retrieved from https://link.springer.com/content/pdf/10.1007/3-540-46416-6_22.pdf

3 LAKE, J. (2021, Березеня 18). *What is RSA encryption and how does it work?* Retrieved from https://www.comparitech.com/: https://www.comparitech.com/blog/information-security/rsa-encryption/

4 Desmedt, Y. (2011). ElGamal Public Key Encryption. *Encyclopedia of Cryptography and Security.* Boston: Springer.

5 Patrick Corn, C. H. (n.d.). *Euler's Totient Function*. Retrieved from brilliant: https://brilliant.org/wiki/eulers-totient-function/

6 Stanford. (n.d.). *The Miller-Rabin Test*. Retrieved from crypto.stanford.edu: https://crypto.stanford.edu/pbc/notes/numbertheory/millerrabin.html

7 Wang, H. P. (2004). Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups. *Australasian Conference on Information Security and Privacy. 3108.* Berlin: Springer. Retrieved from https://eprint.iacr.org/2021/1406.pdf

8 Liu, V. K. (2021). Non-Slanderability of Linkable Spontaneous Anonymous Group Signature (LSAG). Retrieved from https://eprint.iacr.org/2021/1406

9 Bellare, M. (1997). A note on negligible functions. Retrieved from https://eprint.iacr.org/1997/004

10 Liu, J. W. (205). : Linkable ring signatures: Security models and new schemes. *Computational Science and Its Applications - ICCSA 2005. 3481*, pp. 614–623. Berlin: Springer.

11 SHEN, N. (2015). Ring Signature Confidential Transactions for Monero. Retrieved from https://eprint.iacr.org/2015/1098

12 Saberhagen, N. v. (2013, Жовтня 17). *CryptoNote v 2.0*. Retrieved from CryptoNote: https://cryptonote.org/whitepaper.pdf

13 Suzuki, E. F. (2006). Traceable Ring Signature. (pp. 1-10). Cryptology ePrint Archive. Retrieved from https://eprint.iacr.org/2006/389

14 Spagni, l. a. (2017, травень 17). *Disclosure of a Major Bug in CryptoNote Based Currencies*. Retrieved from GetMonero:

https://www.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html

15 Tsang, P. W. (2005). : Short linkable ring signatures for e-voting, e-cash and attestation. *ISPEC 2005. 3439*, pp. 48–60. Berlin: Springer.

16 Nelson, D. (2023, квітень 27). *Russian Bitcoin Wallets Allegedly Exposed by Apparent Hacker*. Retrieved from coindesk: https://www.coindesk.com/business/2023/04/27/russian-bitcoin-wallets-allegedly-exposed-by-apparent-hacker/

17 Gitbook. (n.d.). *ECDSA: Elliptic Curve Signatures*. Retrieved from https://cryptobook.nakov.com/: https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages

18 Hankerson, D. M. (2011). Elliptic Curve Discrete Logarithm Problem. *Encyclopedia of Cryptography and Securit.* Boston: Springer.

19 Schnorr., C. P. (1991). Efficient signature generation for smart cards. *J. Cryptology*, 239–252.

ДОДАТОК А

```python
import ECDSA
import time

class generator:
    def __init__(self, curve, number, path=None):
        self.curve = curve
        self.number = number
        self.path = path

    def key_gen(self, index):
        start = time.time()
        pkeys = []
        for (i) in range(self.number):
            key = ECDSA.PrivateKey(self.curve)
            if i == index:
                ECDSA.PrivateKey.write_pem(key, "private" + str(i) + ".pem")
            pkey = ECDSA.PublicKey.fromPrivateKey( key)
            if self.path is not None:
                ECDSA.PublicKey.write_pem(pkey, self.path + str(i) + ".pem")
            else:
                pkeys.append(pkey)
        if pkeys is not []:
            end = time.time()
            print("All generation time for "+str(self.number)+ " keys: " + str(end - start))
            return pkeys
```

рис. 1 - клас генерації ключів

```python
    def ring_signature(siging_key, key_idx, M, y, G=Curve.generator, hash_func=hash):

        n = len(y)
        c = [0] * n
        r = [0] * n
        H = LRS.H2(y, hash_func=hash_func)
        I = H * siging_key

        u = ecdsa.util.randrange(ecdsa.curve.order)
        c[(key_idx + 1) % n] = LRS.H1([y, I, M, G * u, H * u], hash_func=hash_func)

        for i in [i for i in range(key_idx + 1, n)] + [i for i in range(key_idx)]:
            r[i] = ecdsa.util.randrange(ecdsa.curve.order)

            a = (G * r[i]) + (y[i] * c[i])
            b = (H * r[i]) + (I * c[i])

            c[(i + 1) % n] = LRS.H1([y, I, M, a, b], hash_func=hash_func)

        r[key_idx] = (u - siging_key * c[key_idx]) % ecdsa.curve.order
        return (c[0], r, I)
```

рис. 2 - функція генерації підпису

```python
def verify_ring_signature(message, y, c_0, s, I, G=curve.generator, hash_func=hash):
    n = len(y)
    c = [c_0] + [0] * (n - 1)

    H = H2(y, hash_func)

    for i in range(n):
        a = (G * s[i]) + (y[i] * c[i])
        b = (H * s[i]) + (Y * c[i])

        if i < n - 1:
            c[i + 1] = H1([y, I, message, a, b], hash_func)
        else:
            return c_0 == H1([y, I, message, a, b], hash_func)

    return False
```

рис. 3 - функція верифікації підпису