КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

КІЛЬЦЕВИЙ ПІДПИС

Виконав студент 2-го курсу магістратури
Чемерис Павло Олександрович _____
(підпис)

Науковий керівник:
доктор фіз.-мат. наук, професор
Анісімов Анатолій Васильович _____
(підпис)

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

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри
теорії та технології програмування
«___»_____ 2021 р.,
протокол №____

Завідувач кафедри
проф. М. С. Нікітченко _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 53 сторінки, 3 ілюстрації, 35 джерел посилань.

ЕЛЕКТРОННИЙ ЦИФРОВИЙ ПІДПИС, КІЛЬЦЕВИЙ ПІДПИС, ГРУПОВИЙ ПІДПИС, ПОРОГОВИЙ КІЛЬЦЕВИЙ ПІДПИС, ПРОСТЕЖУВАНИЙ КІЛЬЦЕВИЙ ПІДПИС, ECDSA, RSA.

Об’єктом роботи є процес захисту та верифікації непідробності інформації. Предметом роботи є програмний засіб для створення електронного цифрового підпису.

Метою роботи є дослідження схем кільцевих підписів та створення зручного програмного забезпечення для створення та верифікації цих підписів.

Методи розроблення: шифрування за допомогою алгоритму RSA, шифрування за допомогою алгоритму ECDSA, хешування алгоритмами SHA256 та SHA1, методи створення кільцевого, порогового та простежуваного кільцевого підписів. Інструменти розроблення: мова програмування Python, допоміжні бібліотеки зі збіркою захищених хеш-функцій та різних алгоритмів шифрування – PyCrypto та fastecdsa.

Результат роботи: досліджено різні схеми та модифікації кільцевих підписів, реалізована схема кільцевого підпису та створено зручний інтерфейс на мові Python для її використання, реалізована схема порогового кільцевого підпису зі власною модифікацією для більш зручного використання, реалізована схема простежуваного кільцевого підпису на базі еліптичних кривих.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ЕЦП – електронний цифровий підпис;

RSA – абревіатура від прізвищ Rivest, Shamir і Adleman – криптографічний алгоритм з відкритим ключем;

API – Application Programming Interface, прикладний програмний інтерфейс;

IEEE – Institute of Electrical and Electronics Engineers, світовий лідер в області розробки стандартів з радіоелектроніки, електротехніки та апаратного забезпечення обчислювальних систем і мереж;

SHA – Secure Hash Algorithm, захищений алгоритм хешування;

MD5 – Message Digest 5, 128-бітний алгоритм хешування;

IDLE – Integrated Development and Learning Environment, інтегроване середовище розробки і навчання на мові Python;

ECDSA – Elliptic Curve Digital Signature Algorithm, алгоритм з відкритим ключем для створення цифрового підпису визначений в групі точок еліптичної кривої;

ВСТУП

Електронний цифровий підпис (ЕЦП) – важливий розділ криптографії, який широко використовується у повсякденному житті. Завданням електронного цифрового підпису є ідентифікація особи, що підписала документ та підтвердження цілісності даних в електронній формі.

ЕЦП накладається за допомогою секретного ключа та за правовим статусом прирівнюється до власноручного підпису, тому має широке застосування від юридичних галузей до криптовалют, та є ефективним інструментом забезпечення інформаційної безпеки [1].

Кільцевий підпис – це механізм реалізації електронного підпису, за допомогою якого можна переконатися, що документ підписаний одним з членів списку потенційних підписантів, але при цьому не розголошується особистість автора. Для накладення підпису підписанту не потрібно вимагати дозвіл, сприяння або допомогу з боку включених до списку осіб – використовуються лише відкриті ключі всіх членів списку і власний закритий ключ [2].

Кільцевий підпис був розроблений Рональдом Рівестом, Аді Шаміром і Яелью Тауманом та опублікований у 2001 році на міжнародній конференції Asiacrypt. В порівнянні з груповим підписом, кільцеві підписи не мають групових менеджерів, процедур налагодження, відсутні процедури відкликання та немає координації. Оригінальна стаття авторів має назву “How to leak a secret”, оскільки кільцевий підпис дає змогу забезпечити витік групової інформації, зберігши при цьому анонімність, а також вирішити багато інших задач в системах з групами осіб [3].

На даний момент розроблена велика кількість алгоритмів створення кільцевого підпису, а також створено різні модифікації, що дозволяють вирішувати більш вузьконаправлені, але не менш важливі проблеми захисту інформації та анонімності.

Важливими модифікаціями кільцевого підпису є наступні:

– *порогові кільцеві підписи*: на відміну від стандартного порогового підпису, коли t з n користувачів беруть участь у дешифруванні повідомлення, даний варіант кільцевого підпису зобов'язує t користувачів співпрацювати в процесі підписання. Для цього t учасників $(i_1, i_2, ..., i_t)$ повинні обчислити деяку функція над повідомленням m, подавши t закритих і n відкритих ключів на вхід $(m, S_{i_1}, S_{i_2}, ..., S_{i_t}, P_1, P_2, ..., P_t)$ [4].

– *пов'язані кільцеві підписи*: пов’язаність підписів дає можливість визначити, чи були створені два кільцевих підписи однією людиною (за допомогою одного і того ж секретного ключа), але без можливості визначення особи, тобто зберігається анонімність. Даний тип підпису може застосовуватись в автономній системі електронних грошей [5].

– *простежуваний кільцевий підпис*: має властивість пов'язаності підписів, проте при повторному використанні може розкриватися відкритий ключ підписувача. Такий протокол дозволяє реалізувати системи таємного електронного голосування, які дозволяють поставити анонімно тільки один підпис, а також створювати транзакції в криптовалютах, що розкривають учасника, який провів транзакцію двічі [6].

Метою даної роботи є створення програмного засобу для використання схем кільцевого, порогового та простежуваного кільцевого підписів, оскільки у вільному доступі є лише публікації цих схем, але немає готових рішень, які б інші користувачі могли б використати чи інтегрувати у свої системи.

Для досягнення мети робота буде розділена на наступні етапи:

– дослідження існуючих схем кільцевих підписів
– дослідження практичних застосувань в різних сферах
– оптимізація алгоритмів створення та верифікації підписів
– реалізація цих алгоритмів на одній з мов програмування

РОЗДІЛ 1. КІЛЬЦЕВИЙ ПІДПИС

1.1 Сучасні схеми

Перша схема кільцевого підпису була запропонована Рональдом Рівестом, Аді Шаміром і Яелью Тауманом. Запропонована схема лягла в основу багатьох інших алгоритмів та визначає кільцевий підпис наступним чином: нехай існує n потенційних підписантів, тоді кільцевий підпис визначений двома функціями:

- ***ring-sign***$(m, P_1, P_2,...,P_n, s, S_s)$ створення кільцевого підпису σ для повідомлення m, використовуючи публічні ключі $P_1, P_2,...,P_n$, зі секретним ключем S_s s-ого підписанта, що є справжнім володарем підпису.
- ***ring-verify***(m, σ) функція підтвердження, перевіряє чи повідомлення m з підписом σ (що включає в себе всі публічні ключі) підписане одним з підписантів, повертає *true* або *false*.

Оскільки було введено поняття кільцевих підписів, у літературі було запропоновано ряд нових схем кільцевих підписів. Шахам і Вотерс [7] запропонували першу ефективну схему кільцевого підпису на основі білінійних груп. Ця схема є анонімною щодо повної ключової експозиції та незламною щодо корупції серед групи. Кар [8] запропонував схему онлайн/офлайн-сигналу, засновану як на обчислювальних проблемах Diffie-Hellman, так і k-CAA. Схема задовольняє неоднозначність підписувача і дозволяє виявити неправильну поведінку. Ван та інші співавтори [9] запропонували нову концепцію “identity-based quotable ring signature”, яка

може бути використана для створення нових кільцевих підписів на підрядках початкового повідомлення використовуючи початковий кільцевий підпис початкового повідомлення. Схема ґрунтується на білінійному спарюванні складеного порядку і є надійною, оскільки базується на протоколі Діффі-Геллмана. Зенг та інші автори запропонували ефективну не інтерактивну схему відрікального кільцевого підпису і довели свою безпеку в стандартній моделі [10].

Щоб зрозуміти принцип роботи кільцевого підпису та перейти до дослідження його модифікацій, в даній роботі було досліджено та реалізовано схему, запропоновану Рональдом Рівестом та іншими.

1.2 Застосування в різних сферах

Окрім тривіального застосування схеми для підписання документів та повідомлень, кільцевий підпис знайшов застосування в хмарних технологіях та криптовалютах.

В криптовалюті Bytecoin використовується система CryptoNote, що в свою чергу базується на кільцевих підписах [11]. Криптовалюта ShadowCash для забезпечення анонімності відправника транзакцій, використовує простежуваний кільцевий підпис [12].

Серед хмарних технологій, було запропоновано схему для обміну даними в хмарі на базі кільцевого підпису [13] та схема для збереження конфіденційності в громадському аудиті [14] .

1.3 Теоретичні відомості

Нижче представлений опис головних понять, що використовують в схемі кільцевого підпису.

1.3.1 RSA

RSA – криптографічний алгоритм, розроблений Рівестом, Шаміром та Адлеманом. Надійність алогритму базується на обчислювальній складності задач факторизації великих цілих чисел, тому він може використовуватися в задачах шифрування та створення цифрового підпису [15].

В цифровому підписі кожен підписант має RSA публічний ключ $P = (n,\ e)$, який визначає параметри односторонньої функції:

$$f(x) = x^e\ (mod\ n)$$

та секретний ключ d, що використовується як "таємний вхід" до односторонньої функції. Це означає, що лише підписант може ефективно порахувати значення f^{-1}.

1.3.2 Розширення односторонньої функції в RSA

В схемі кільцевого підпису використовуються публічні ключі всіх членів групи, тому значення функцій $f_i(x)$ у різних підписантів матимуть різний розмір, навіть, якщо всі n_i матимуть однакову кількість бітів. Це ускладнює комбінування власних підписів всіх підписантів, тому було вирішено розширити односторонню функцію RSA так, щоб всі значення функцій мали однакову кількість бітів.

Нехай b таке ціле число, що 2^b більше за всі n_i. Тоді для будь-якого b-бітного числа m знайдемо такі невід'ємні q_i та r_i, що виконується:

$$m = q_i * n_i + r_i, \text{ де } 0 \leq r_i < n_i.$$

Звідси маємо:

$$g_i(m) = \{q_i * n_i + f_i(r_i), \text{ якщо } (q_i + 1) * n_i \leq 2^b$$
$$g_i(m) = m, \text{ інакше}$$

Якщо вибране b буде недостатньо великим числом, то в деяких випадках m залишиться незмінним. Тому, зазвичай b вибирають на 160 бітів більшим ніж всі n_i [3].

1.3.3 Хеш-функція

Хеш-функція – функція перетворення вхідного масиву даних будь-якої довжини в масив даних фіксованого розміру. У випадках, коли розмір результуючого масиву менше, ніж вхідного, з’являються “колізії” - явище коли результат функції для двох різних вхідних масивів однаковий. Тому, при створенні хеш-функції її параметри підбирають таким чином, щоб зменшити кількість колізій [16].

Для хеш-функцій $h(x)$, що використовуються в криптографії також має виконуватися незворотність: для заданого значення хеш-функції m має бути обчислювально неможливо знайти блок даних x, для якого $h(x) = m$.

1.3.4 Комбінуюча функція

Комбінуюча функція $C_{k,\,v}(y_1, y_2, ..., y_n)$ – функція, що приймає на вхід ключ k, початкове значення v та довільні b-бітні числа $y_1, y_2, ..., y_n$. Комбінуюча функція є суперпозицією вкладених функцій E_k (функція

симетричного шифрування або хеш-функція, залежна від k), та повертає b-бітне число z.

Також, комбінуюча функція має наступні властивості:

1. Для кожного s, $1 \leq s \leq n$, і для будь-яких фіксованих значень y_i, $i \neq s$, функція $C_{k,\ v}$ має взаємно однозначне відображення з y_s в z.
2. Для кожного s, $1 \leq s \leq n$, заданого b-бітного числа z, та для всіх y_i, окрім y_s, можливо ефективно знайти таке b-бітне число y_s, що $C_{k,\ v}(y_1,\ y_2, ..., y_n) = z$.
3. Неможливо підібрати параметри для всіх вхідних даних не маючи секретного ключа. Тобто, для заданих k, v, z та $y_1, y_2, ..., y_n$ неможливо знайти такі $x_1, x_2, ..., x_n$ що: $C_{k,\ v}\left(y_1 = g_1(x_1),\ y_2 = g_2(x_2), ...,\ y_n = g_n(x_n)\right) = z$, не знаючи "таємного входу" до односторонніх функцій $g_1(x_1),\ g_2(x_2), ..., g_n(x_n)$.

1.4 Схема підпису

В основі кільцевого підпису лежить комбінуюча функція

$$C_{k,\ v}(y_1,\ y_2, ..., y_n) = v = E_k(y_r \oplus E_k(y_{r-1} \oplus E_k(y_{r-2} \oplus E_k(...\oplus E_k(y_1 \oplus v)...))))$$
$$y_1 = g_1(x_1),\ y_2 = g_2(x_2), ...,\ y_n = g_n(x_n)$$

- $\oplus$ – бітова операція XOR
- g_i – розширення одностороння функція RSA: $f_i(x) = x^{e_i} (mod\ n_i)$, i-го члена групи
- $k = h(m)$– значення хеш-функції застосованої до повідомлення m

- E_k – залежна від kфункція симетричного шифрування (може також бути хеш-функцією)
- $v,\ x_1,\ x_2, ..., x_n$ – деякі випадкові числа.

Тоді, якщо учасник групи з номером s хоче створити кільцевий підпис, отримаємо наступну схему кільцевого підпису:

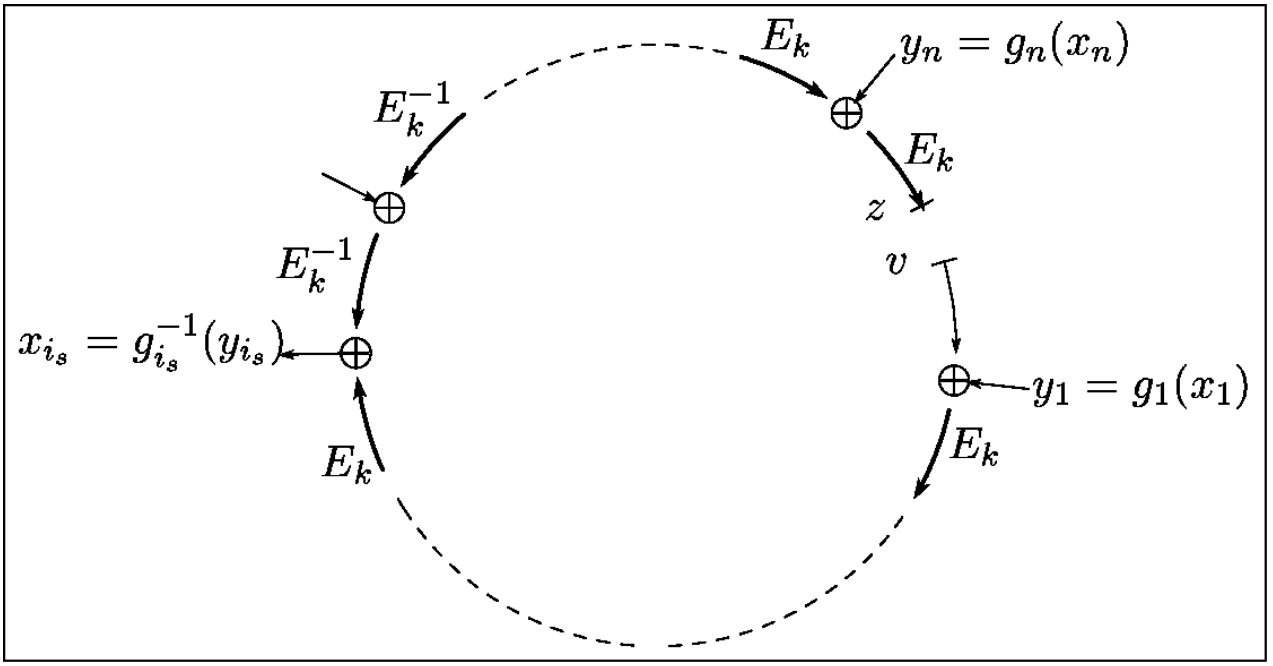

Рисунок 1 – Схема кільцевого підпису

За даною схемою зрозуміло, що неможливо підібрати випадковим чином такі числа $v,\ x_1,\ x_2, ..., x_n$, що задовільняють комбінуючу функцію, оскільки на кожному кроці рахується значення односторонньої функції $y_i = g_i(x_i)$, тому не знаючи "таємного входу" до хоча б однієї з функції, неможливо створити комбінуючу функцію, що "починається" на "закінчується" випадковим значенням v. Також зрозуміло що комбінуюча функція зберігає в собі повідомлення, яке неможливо підробити, оскільки E_k – функція симетричного шифрування або хеш-функція.

1.5 Алгоритм створення підпису

Задано повідомлення m, яке потрібно підписати, номер підписувача s, його секретний ключ S_s та множина відкритих ключів всіх можливих підписантів $P_1, P_2, ..., P_n$. Створення підпису відбувається наступним чином:

1. Вибирається ключ: підписувач рахує симетричний ключ k, як хеш-функцію від повідомлення m: $k = h(m)$.
2. Вибираються x_i: підписувач випадковим чином генерує x_i $(1 \le i \le n,\ i \ne s)$ для всіх інших підписантів, окрім себе, та рахує:
$$y_i = g_i(x_i)$$
3. Рахується число v_s: підписувач випадковим чином генерує b-бітне число u та рахує $v_s = E_k(u)$.
4. Послідовно від $s+1$ до n рахується: $v_{i+1} = E_k(v_i \oplus y_{i+1})$, та зберігається$v = v_n$.
5. Послідовно від 1 до $s-1$ рахується: $v_1 = E_k(v_n \oplus y_1)$, $v_{i+1} = E_k(v_i \oplus y_{i+1})$
6. Рахується x_s: з наступних рівностей маємо:
$$v_s = E_k(u) = E_k(v_{s-1} \oplus y_s)) => u = v_{s-1} \oplus y_s => y_s = u \oplus v_{s-1}$$
$$x_s = g_s^{-1}(y_s) => x_s = g_s^{-1}(u \oplus v_{s-1})$$
використовуючи секретний ключ, який є “таємним входом” до односторонньої функції g_s, підписувач знаходить потрібне йому x_s.
7. Повертається кільцевий підпис: підпис над повідомленням m визначений кортежем $(P_1, P_2, ..., P_n, v, x_1, x_2, ..., x_n)$.

Асимптотичний час – $O(n)$.

1.6 Алгоритм верифікації

Для заданого повідомлення m та кільцевого підпису $(P_1, P_2, ..., P_n, v, x_1, x_2, ..., x_n)$, маємо наступний алгоритм верифікації:

1. Рахується y_i: для всіх x_i $(1 \leq i \leq n)$ перевіряючий рахує y_i:

$$y_i = g_i(x_i)$$

2. Рахується k: для повідомлення m перевіряючий рахує ключ шифрування:

$$k = h(m)$$

3. Верифікація підпису: перевіряється чи y_i задовольняють рівності:

$$C_{k,\, v}(y_1, y_2, ..., y_n) = v$$

Якщо це так, підпис вважається дійсним, інакше – ні.

Асимптотичний час – $O(n)$.

РОЗДІЛ 2. ПОРОГОВИЙ КІЛЬЦЕВИЙ ПІДПИС

2.1 Сучасні схеми

Емануель Брессон, Жак Стерн та Майкл Шидло розширили поняття кільцевого підпису та створили пороговий кільцевий підпис, який стає все більш популярним у порівнянні зі звичайними кільцевими підписами [17]. Подібно до схеми кільцевого підпису, схема порогового кільцевого підпису дозволяє щонайменше *t* підписантам співпрацювати один з одним для підписання повідомлення, без витоку будь-якої інформації про підписантів. Пороговий кільцевий підпис визначений двома функціями:

- ***T-ring-sign****(m, P_1, P_2,...,P_n, s, S_s)* створення кільцевого підпису для повідомлення *m*, використовуючи публічні ключі P_1, P_2,...,P_n, разом зі секретними ключами $S_1, S_2,..., S_t$ t підписантів, що є справжніми володареми підпису. Значення *t*, а також *n* відкритих ключів включаються в результат.
- ***T-ring-verify****(m, σ)* функція підтвердження, перевіряє чи повідомлення *m* з підписом *σ* (що включає в себе всі публічні ключі) підписане *t* підписантами, повертає *true* або *false*.

Існуючі схеми порогового кільцевого підпису в основному базуються на теорії чисел, наприклад: “Мультиваріантна схема порогового кільцевого підпису” [18], “Схема порогового кільцевого підпису в сертифікованій криптографії” [19], “Пороговий кільцевий підпис без випадкових оракулів” [7], “Пороговий кільцевий підпис на основі ідентичності без пар” [20] та інші. Всі

вони, по суті, є модифікаціями першої схему, описаної Емануелем Брессоном та іншими.

Водночас існують схеми, що базуються на теорії кодування, такі як: “Provably secure code-based threshold ring signatures” [21], “A new efficient threshold ring signature scheme based on coding theory” [22], “An Efficient Code-Based Threshold Ring Signature Scheme with a Leader-Participant Model” [23], “An efficient code-based threshold ring signature scheme (2019)” [24]. Перевагою схем, що базуються на теорії кодування є стійкість до атак квантовими комп’ютерами.

Для дослідження було обрано схему, представлену Емануелем Брессоном та іншими. По-перше, вона не вимагає додаткових знань в теорії кодування, по-друге, в ній описані модифікації, що спрощують звичайну схему кільцевого підпису, а також, в основі схеми лежить цікавий алгоритм, що може бути застосованим і до інших задач.

2.2 Практичні застосування

Одним з практичних застосувань порогових кільцевих підписів є суд присяжних. За допомогою порогового кільцевого підпису t з n присяжних можуть підписати своє рішення, при цьому приховуючи особисте рішення кожного.

Також, схема порогового кільцевого підпису може бути корисна в системі відгуків та пропозицій. Наприклад, t з n працівників компанії вирішили поскаржитися на керівництво, щоб уникнути особистих конфліктів, працівники можуть підписати скаргу від імені t з n працівників компанії, при цьому приховуючи свої особистості.

2.3 Модифікація звичайної схеми

Для створення схеми порогового кільцевого підпису, потрібно дещо спростити та модифікувати існуючу схему кільцевого підпису, описану в Розділі 1. Надійність підпису та анонімність підписувачів в новій схемі залишаються такими ж.

2.3.1 Прості спостереження

Як описано в розділі 1.3.4 комбінуюча функція використовує функцію E_k – функція симетричного шифрування або хеш-функція. Для спрощення схеми будемо представляти E_k, лише як хеш-функцію. В алгоритмі з розділу 1.5 E_k можна представити, як хеш-функцію.

Отже, v_i(див. 1.5) рахується наступним чином:

$v_s = H_k(\sigma = u)$, $v_{i+1} = H_k(v_i \oplus y_{i+1})$, де H_k: хеш-функція.

Також, алгоритм верифікації можна починати з будь-якої позиції i_0, не обов’язково з 1. Це надає змогу підписувачу вказати позицію, з якої потрібно перевіряти підпис та ніяк не погіршує надійність підпису.

2.3.2 Розрив

В схемі кільцевого підпису, значення z комбінуючої функції, дорівнює її початковому значенню v (див. 1.4): $z = v$. Початкового значення v та рівності $z = v$ можна позбутися додавши число “розриву” (“gap” value): у вибраній позиції i_γ додамо число “розриву” γ: $v_{i_\gamma+1} = H_k(v_{i_\gamma} \oplus y_{i_\gamma+1} \oplus \gamma)$. Таке спрощення дає змогу підписувачу вільно вибирати значення γ, та позбутися v,

оскільки значення v не можна підібрати випадковим чином. Така модифікація також не впливає на надійність підпису, але при умові, що γ не може бути обрано підробником. Таким чином, ця модифікація не може використовуватися у звичайному підписі, без доказів, що γ не є підробкою.

2.3.3 Модифікована схема

В результаті модифікацій схеми, описаної в Розділі 1, отримаємо таку комбінуючу функцію:

$$C_{k,v,i_0}\left(i_\gamma, \gamma, y_1, y_2, \ldots, y_n\right) =$$

$$= H_k(y_{i_0-1} \oplus H_k(y_{i_0-2} \oplus \ldots H_k(y_{i_\gamma} \oplus \gamma \oplus E_k(\ldots \oplus E_k\left(y_{i0} \oplus v\right)\ldots))))$$

Це можна подати схематично так:

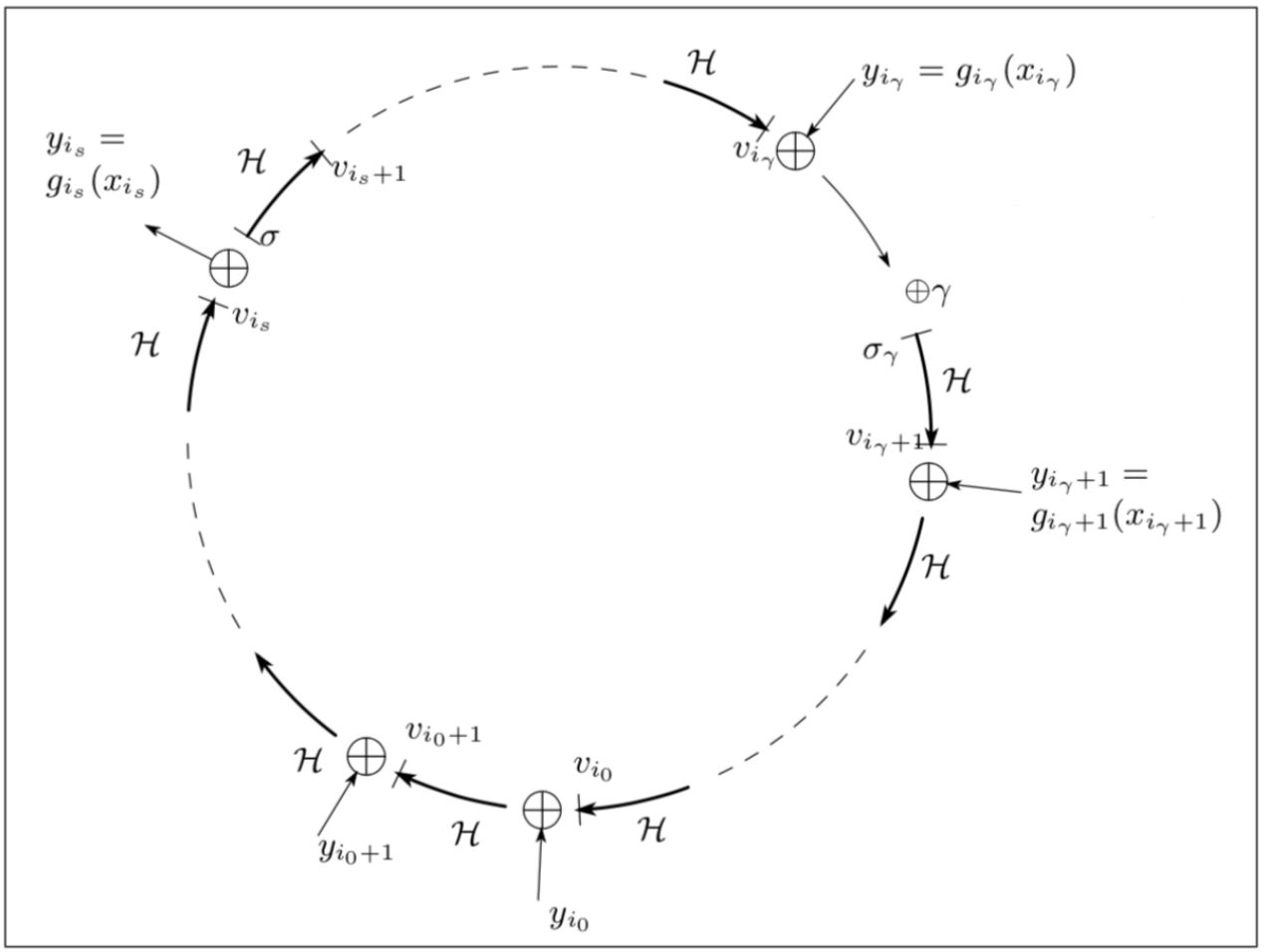

Рисунок 2 – Схема модифікованого кільцевого підпису

2.4 Схема порогового кільцевого підпису

Для початку розглянемо тривіальне рішення. Оскільки, звичайна схема кільцевого підпису дає змогу переконатися, що як мінімум один учасник групи створив підпис, то найпростішою ідеєю створення порогового кільцевого підпису буде наступна: розбити всіх учасників на t різних груп, що не перетинаються, та помістити кожного справжнього підписанта в окрему групу. Таким чином ми створимо t незалежних кільцевих підписів, перевіривши які, можна легко переконатися, що участь взяли мінімум t учасників.

Проте таке схема не забезпечує повну анонімність підписантів, оскільки різко зменшується кількість різних потенційних груп справжніх підписантів (учасники, що були розподілені в одну групу не могли бути справжніми підписантами).

Основна ідея полягає в створенні П таких “розбиттів”, щоб для будь-яких t учасників з групи можливих підписантів, знайшлося таке розбиття, де кожен з цих t належить окремій групі. Зрозуміло, що тепер потрібен спосіб об’єднання всіх розбиттів в одну схему, та забезпечення не підробності такої схеми.

Спочатку об’єднаємо всі групи одного розбиття: для цього можна скористатися числами “розриву” (див. 2.3.2) модифікованих кільцевих підписів, створених для кожної з груп. Нехай $\gamma_1, \gamma_2, \ldots, \gamma_t$ – значення “розриву” кільцевих підписів. Тоді числом розбиття u буде $u = \gamma_1 || \gamma_2 || \ldots || \gamma_t$, де $||$ – операція конкатенації бітових представлень чисел, тобто довжина числа uбуде $t * b$ біт (за умови, що всі γ довжиною b біт).

Тепер потрібно об’єднати всі розбиття в одну схему, при цьому забезпечити не підробність такої схеми та анонімність підписантів. Для цього скористаємося звичайним кільцевим підписом, з наступною комбінуючою функцією:

$$C_{k,\,v}\left(u_1,\,u_2,\ldots,u_n\right)=v=\;G_k(u_n\oplus G_k(u_{n-1}\oplus G_k(u_{n-2}\oplus G_k(\ldots\oplus G_k\left(u_1\oplus v\right)\ldots$$

де G_k – хеш-функція, що повертає рядок довжиною $t\ *\ b$біт.

Таким чином, отримаємо схему порогового кільцевого підпису. Результат можна зобразити схематично:

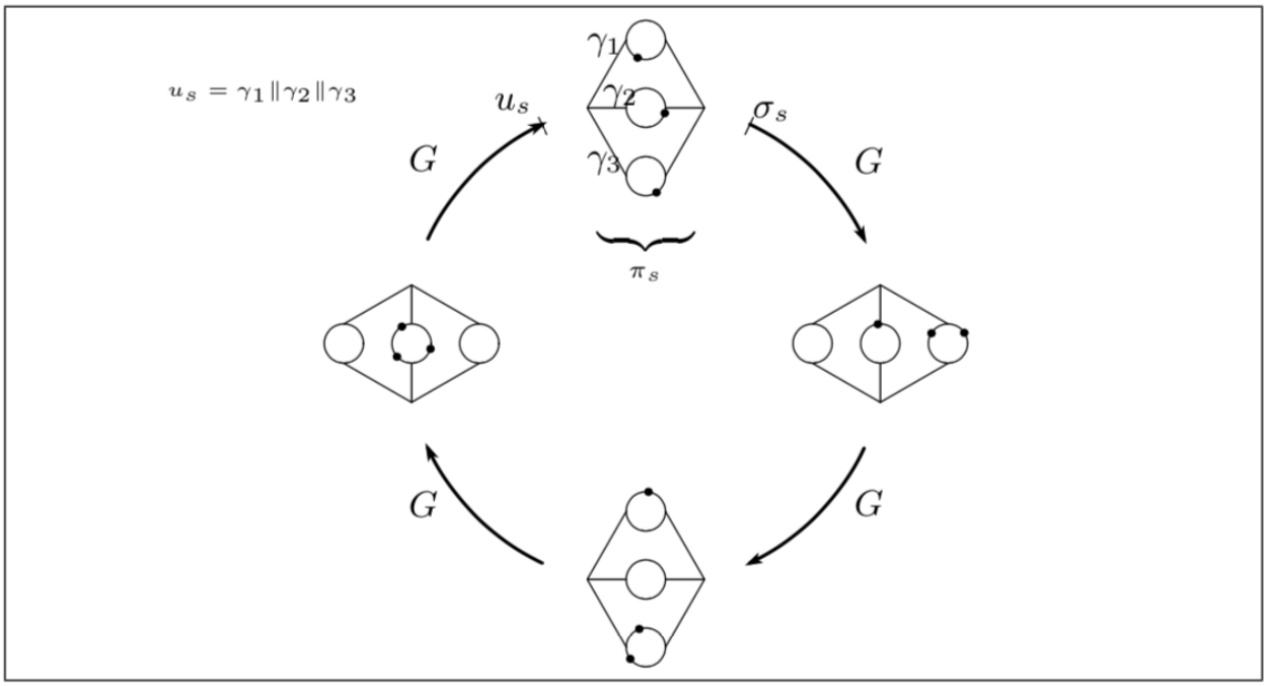

Рисунок 3 – Схема порогового кільцевого підпису

Доведення надійності та реалізація такої схеми описані в наступних розділах.

2.5 Надійність

Зрозуміло, що з не заданими наперед γ, кожен модифікований підпис в розбитті можна підробити. Проте, всі розбиття об'єднує звичайний кільцевий підпис. А якщо звичайний кільцевий підпис дійсний, тобто значення комбінуючої функцію z дорівнює v, то одне зі значень u_s не може містити в собі модифіковані кільцеві підписи з випадковими γ. А оскільки, $u = \gamma_1 || \gamma_2 || \ldots || \gamma_t$, то це означає, що в цьому розбитті всі кільцеві підписи справжні, тому що лише в справжньому підпису, можна задати γ потрібного значення, а не згенерувати випадковим чином. Звідси, як мінімум *t* учасників створили підпис.

2.6 Алгоритм створення підпису

1. Вибирається ключ: підписувач рахує симетричний ключ k, як хеш-функцію від повідомлення m: $k = h(m)$.
2. Генеруються П розбиттів $\pi_1, \pi_2, \ldots, \pi_П$ таким чином, щоб для будь-яких *t* учасників групи, знайшлося таке розбиття π_i, в якому кожен з цих учасників належить різній групі. Розбиття в якому знаходяться справжні підписанти позначається π_s.
3. Для кожного розбиття, окрім π_s створюються модифіковані кільцеві підписи для кожної з груп:
    a. Вибираються x_i: підписувач випадковим чином генерує x_i $(1 \leq i \leq n)$
    b. Рахуються y_i: $y_i = g_i(x_i)$
    c. Для кожної групи $p_1, p_2, \ldots, p_t$ генеруються випадкові $v_1, v_2, \ldots, v_t$
    d. Для кожної групи $p_1, p_2, \ldots, p_t$ рахуються $\gamma_1, \gamma_2, \ldots, \gamma_t$:

$$\gamma_i = C_{k,\ v_i}(y_{p_i1},\ y_{p_i2}, \ldots) \oplus v_i$$

e. Повертаються кільцеві підписи, що задають параметри комбінуючих функцій:

$$C_{k,\ v,\ 1}\left(1, \gamma_i, y_{\ 1}, y_{p_i1},\ y_{p_i2} \ldots, y_n\right)$$

f. Рахується значення u: $u\ =\ \gamma_1 || \gamma_2 || \ldots || \gamma_t$

4. Рахується число r: підписувач випадковим чином генерує $t*b$-бітне число r та рахує $v_s = E_k(r)$, де s – номер розбиття, в якому знаходяться справжні підписанти.

5. Послідовно від $s + 1$ до П рахується: $v_{i+1} = E_k(v_i \oplus u_{i+1})$, та зберігається$v = v_n$.

6. Послідовно від 1 до $s - 1$ рахується: $v_1 = E_k(v_n \oplus u_1)$, $v_{i+1} = E_k(v_i \oplus u_{i+1})$

7. Рахується u_s: з наступного рівностей маємо:

$$u_s\ = r \oplus v_{s-1} = \gamma_1 || \gamma_2 || \ldots || \gamma_t$$

8. За отриманим u_s та $\gamma_1 || \gamma_2 || \ldots || \gamma_t$ рахуємо кільцеві підписи для розбиття π_s.

9. Повертається пороговий кільцевий підпис, що містить значення v, розбиття П та всі модифіковані кільцеві підписи.

Складність алгоритму залежить кількості розбиттів П. Як доведено в публікації “Color coding” [25] можна створити $2^{O(t)} log_2 n$ таких розбиттів. Таким чином, розмір порогового кільцевого підпису

$$2^{O(t)} log_2 n\ *\ (t\ *\ b\ +\ n\ *\ b)\ =\ O(b\ 2^t n\ log_2 n)$$

А асимптотичний час створення підпису – $O(2^t n\ log_2 n)$.

Також, якщо це задовольняє умову задачі, можна створити розбиття, що не покриває всі можливі групи з t учасників, що значно зменшить розмір підпису та час на його створення.

2.7 Алгоритм верифікації

1. За заданим розбиттям П будуються модифіковані кільцеві підписи та перевіряються на коректність, як звичайні кільцеві підписи (див 1.6).
2. Рахуються значення $u_i = \gamma_1 || \gamma_2 || \dots || \gamma_t$.
3. Будується звичайний кільцевий підпис на значеннях u_i та перевіряється, як в розділі 1.6.

Асимптотичний час верифікації підпису, залишається таким самим, як і при створенні $O(2^t n\ log_2 n)$. Проте константний час, набагато менше, оскільки не потрібно генерувати розбиття.

РОЗДІЛ 3. ПРОСТЕЖУВАНИЙ КІЛЬЦЕВИЙ ПІДПИС

3.1 Сучасні схеми

В 2007 році Ейчіро Фудзісакі та Кутару Сузукі представили поняття простежуваного(одноразового) кільцевого підпису, що дозволяє визначити чи були створені два кільцевих підписи однією людиною, та при повторному використанні розкриває особу підписанта [6]. Більш детально, простежуваний кільцевий підпис гарантує наступні умови:

- **Публічна простежуваність** - будь-кого, хто створює два підписи для різних повідомлень використовуючи один і той же публічний ключ, можна відстежити, де відстеження можна здійснити лише з парами повідомлень, підписів та публічних ключів.
- **Пов'язаність ключів** - кожні два підписи, згенеровані за допомогою одного і того ж ключа - пов'язані, тобто загальна кількість підписів щодо одного і того ж набору ключів не може перевищувати загальної кількості підписантів, якщо кожні два підписи не є пов'язаними.
- **Анонімність** - поки підписант не підписує два різних повідомлення використовуючи один ключ, особа підписувача залишається невідомою. Крім того, будь-які, два підписи створені відносно двох різних наборів ключів, завжди неможливо зв'язати, тобто неможливо визначити чи генерувалися вони одним підписувачем.
- **Виправданість** - чесного підписанта не можна звинуватити в тому, що він підписав повідомлення. А саме, зловмисник не може створити простежуваний кільцевий підпис таким чином, що разом з

оригінальним підписом цілі, чесного підписанта буде звинувачено у повторному використанні. Це повинно бути нездійсненним навіть після того як зловмисник має дані всіх інших підписантів, окрім цілі.

Існуючих схем простежуваного кільцевого підпису набагато менше, ніж звичайного чи порогового підписів, всі вони були представлені лише декілька років тому, хоча практичне використання не менш широке.

Ксав'є Бультель і Паскаль Лафуркад представили “k-Times Full Traceable Ring Signature”, що дає змогу одному підписанту створювати підпис не більше k разів, перш ніж його буде особистість буде викрито [28]. Ке Гу та На Ву представили модифіковану схему, що дозволяє створювати простежуваний кільевий підпис без випадкових оракулів та має константний розмір [29], і вже в 2020 році Ке Гу та інші автори покращили представлену схему, що дозволяє більш ефективно створювати підпис без генерації пар ключів [30]. Майже одночасно з публікацією попередньою схеми, автори Ханвен Фен та інші на конференції RSAConference2020, опублікували схему, що дозволяє створювати простежувані кільцеві підписи стійкими до атак квантових комп’ютерів [31]. В 2021 році Фен Тан та інші автори описали схему, що дозволяє ефективно використовувати простежуваний кільевий підпис в блокчейнах [32].

В даній роботі було досліджено схему, представлену Ейчіро Фудзісакі та Кутару Сузукі, та модифіковано за допомогою еліптичних кривих, для можливого застосування в криптовалютах.

3.2 Практичні застосування

Одним з практичних застосувань простежуваних кільцевих підписів є система таємного електронного голосування, що дозволить забезпечити анонімність та заборонити повторне голосування. Проте, найбільш широко простежувані кільцеві підписи застосовуються в криптовалютах.

Оригінальний протокол CryptoNote [33] реалізує простежуваний кільцевий підпис, щоб запобігти подвійним витратам, таким чином власник криптовалюти не може створити більше однієї транзакції одним і тим же ключем, не помічаючи його на блокчейні. Розробники криптовалюти Monero модифікували схему простежуваного кільцевого підпису, що дає змогу приховути суми надіслані в транзакції [34].

3.3 ECDSA

Розглянута схема простежуваного кільцевого підпису базується на еліптичних кривих, тому нижче представлено загальний опис побудови цифрових підписів на еліптичних кривих.

ECDSA – Elliptic Curve Digital Signature Algorithm, криптографічний алгоритм для створення цифрового підпису, що визначений в групі точок еліптичної кривої [35].

Для генерації ключів вибирається еліптична крива E визначена над скінченним простим полем F_p(p - просте число), і точка P простого порядку q кривої $E(F_p)$. Кожен користувач генерує ключ за допомогою наступних дій:

1. Вибирається випадкове ціле число k з інтервалу [1, q - 1].
2. Обчислюється кратне $Q = kP$, $Q = (x, y)$.

Відкритими ключем користувача є точка Q, а закритим k.

Для створення цифрового підпису над повідомленням m потрібно виконати наступні дії:

1. Порахувати $h = H(m)$, де H - хеш-функція.
2. Порахувати $r = x \bmod q$.
3. Порахувати $k^{-1} \bmod q$та визначити $s = k^{-1}(h + x * r) \bmod q$.

Підписом повідомлення буде пара чисел (r, s).

Для перевірки цифрового підпису потрібно виконати наступні дії:

1. Порахувати $u_1 = s^{-1} h \, mod \, q$.
2. Порахувати $u_2 = s^{-1} r \, mod \, q$.
3. Порахувати $u_1 * P + u_2 * Q = (x_0, y_0)$ і відносно x_0 порахувати $v = x_0 \, mod \; q$.
4. Підпис вважається вірним тоді і лише тоді, коли $v = r$.

3.4 Алгоритм створення підпису

В розглянутому алгоритмі кожен підписант має два ключі схеми ECDSA. Таким чином, для всіх підписантів вибираються дві еліптичні криві E_1, E_2 визначені над скінченними простими полями F_{p_1}, F_{p_2}, відповідно, а також точки на цих кривих P_1, P_2.

Ключі підписантів генеруються наступним чином: $Q_{1_i} = k_{1_i} P_1$, $Q_{2_i} = k_{2_i} P_2$, де (Q_{1_i}, Q_{2_i})- публічні ключі, а (k_{1_i}, k_{2_i}) - закриті ключі i-ого підписанта.

Маючи інформацію про всі публічні ключі, закриті ключі (k_1, k_2) та повідомлення m підписант генерує підпис за наступним алгоритмом:

1. Вибирається ключ: підписувач рахує симетричний ключ k, як хеш-функцію від повідомлення m: $k = hash(m)$
2. Вибираються x_i та y_i: підписувач випадковим чином генерує x_i, y_i $(1 \leq i \leq n, \; i \neq s)$ для всіх інших підписантів, крім себе.
3. Підписувач рахує $n - 1$ підробку чужих підписів $ECDSA(m_i, \alpha_i, \beta_i)$ наступним чином:

$$\alpha_i = x_i P_1 + y_i Q_{1_i};\ \beta_i = -\ hash(\alpha_i) * y_i^{-1} mod\ N;\ m_i = x_i * \beta_i\ mod\ N$$

4. Аналогічно, до звичайної схеми кільцевого підпису рахується значення m_s так, щоб замкнулася комбінуюча функція:

$$C_{k,\ v}(m_1,\ m_2, ..., m_n) = v = E_k(m_n \oplus E_k(m_{n-1} \oplus E_k(m_{n-2} \oplus E_k(...\oplus E_k(m_1 \oplus v)...)))))$$

   a. Генерується випадкове u.

   b. Рахується $v = E_k(m_n \oplus E_k(m_{n-1} \oplus E_k(...\ m_{s+1} \oplus E_k(u))))$

   c. Рахується $u' = E_k(m_{s-1} \oplus E_k(m_{s-2} \oplus E_k(...\oplus E_k(m_1 \oplus v)...)))$

   d. $m_s = u \oplus u'$

5. За допомогою секретних ключів (k_1, k_2) рахуються значення (α_s, β_s):

   a. $\alpha_s = k_2 P_1$

   b. $\beta_s = (m_s + hash(\alpha_s) * k_1) * k_2^{-1} mod\ N$

Також, крім комбінуючої функції, підписант має створити доведення еквівалентності двох множників k_2 : $\alpha_s = k_2 P_1$, $Q_{2_s} = k_2 P_2$. Доведення має зберігати властивість анонімності, тобто індекс підписанта s залишається в секреті. Для цього потрібно виконати наступні дії:

1. Генеруються випадкові пари $(q_i,\ w_i)$ з простору $\{1,...,\ N - 1\}^2$.

2. Рахується L_i:

   a. $L_i = q_i P_1$, якщо $i = s$.

   b. $L_i = q_i P_1 + w_i * \alpha_i$, якщо $i \neq s$.

3. Рахується R_i:

   a. $R_i = q_i P_2$, якщо $i = s$.

b. $R_i = q_i P_2 + w_i^* Q_{2_i}$, якщо $i \neq s$.

4. Рахується хеш $c = hash(L_1, ..., L_n, R_1, ..., R_n, m)$.
5. Рахується c_i:
   a. $c_i = w_i$, для всіх $i \neq s$.
   b. $c_s = c - \sum_{i=1}^{n} c_i \bmod N$.
6. Рахується r_i:
   a. $r_i = q_i$, для всіх $i \neq s$.
   b. $r_s = q_s - c_s * k_2 \bmod N$

Доказом, еквівалентності двох множників є набір пар (c_i, r_i).

Таким чином, простежуваний підпис визначений наступними значеннями:

- набір публічних ключів (Q_{1_i}, Q_{2_i})
- набір трійок $ECDSA(m_i, \alpha_i, \beta_i)$
- значення v комбінуючої функції $C_{k,v}$
- пара чисел для доказу підпису (c_i, r_i)

3.5 Алгоритм верифікації

Для верифікації простежуваного кільцевого підпису потрібно виконати перевірки наступних рівностей:

1. $\alpha_i = (m_i \beta_i^{-1}) P_1 + (hash(\alpha_i) * \beta_i^{-1}) Q_{1_i}$
2. $v = E_k(m_n \oplus E_k(m_{n-1} \oplus E_k(m_{n-2} \oplus E_k(...\oplus E_k(m_1 \oplus v)...))))$

3. $\sum_{i=1}^{n} c_i = hash(L'_1,..., L'_n, R'_1,..., R'_n, m) \bmod N$, де

   a. $L'_i = r_i P_1 + c_i \alpha_i$

   b. $R'_i = r_i P_2 + c_i Q_{2_i}$

Щоб перевірити чи не були два підписи створені однією людиною, достатньо перевірити наявність однакових значень $\alpha_i = k_{2_i} P_1$ в двох різних підписах, якщо такі значення існують, то можна визначити індекс публічного ключа, та сам публічний ключ підписанта.

3.6 Надійність

В основі простежуваного кільцевого підпису лежить комбінуюча функція, отже значення m_s не можна підібрати випадковим чином. Теоретично, значення (α_s, β_s) можна підібрати, щоб виконувалася рівність 1 з розділу 3.5, проте рівність 3 гарантує, що хоча б одне зі значень $\alpha_i = k_{2_i} P_1$, звідси

$$k_{2_s} P_1 = k_{2_s}(m_s \beta_s^{-1}) P_1 + (hash(\alpha_s) * \beta_s^{-1}) Q_{1_s} => \beta_s = (m_s + hash(\alpha_s) * k_1) * k_2^{-1}$$

.

Рівність 3 дещо схожа на комбінуючу функцію, можливо підібрати всі (c_i, r_i) випадковим чином, крім пари (c_s, r_s), оскільки мають виконуватися рівності 3a та 3b, що можливо лише за допомогою секретного ключа k_2.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ

4.1 Вибір мови програмування

Для реалізації алгоритмів створення та верифікації схем кільцевого підпису слід обрати зручну мову програмування. Тому, вибрана мова програмування повинна відповідати наступним критеріям:

1. Кросплатформність
2. Наявність добре документованої, кросплатформної та швидкої бібліотеки для генерації ключів RSA та ECDSA
3. Наявність бібліотеки для генерації великих псевдовипадкових чисел
4. Наявність вбудованої реалізації алгоритмів довгої арифметики
5. Надійність компілятора/інтерпретатора
6. Можливість написання простого, зрозумілого та легко-змінного коду
7. Наявність засобів тестування та перевірки коректності програми

Python – високорівнева мова програмування загального спрямування – від простих скриптів на веб-сервері до реалізації алгоритмів штучного інтелекту [26].

Python робить процес розробки програм на ній елегантним, потужним і добре продуманим. Синтаксис ядра Python – мінімалістичний, але в той же час стандартна бібліотека включає широкий спектр готових функцій.

Серед основних її переваг можна назвати такі:

- чистий синтаксис
- переносимість програм
- можливість використання в діалоговому режимі

- зручність у використанні для складних математичних обчислень: робота з комплексними числами, оперування з цілими числами довільної величини

4.2 Вибір допоміжних бібліотек

Для Python існує величезна кількість бібліотек, що полегшують розробку. Серед них бібліотека – PyCrypto.

PyCrypto – це збірка захищених хеш-функцій (таких як SHA256 та RIPEMD160) та різних алгоритмів шифрування (AES, DES, RSA, ElGamal та інші) [27]. Пакет структурований таким чином, що додавати нові модулі досить легко, та складається з п'яти підпакетів:

1. Crypto.Cipher – реалізація симетричних (AES, DES, ARC4) та асиметричних (RSA, ElGamal) алгоритмів шифрування.
2. Crypto.Hash – алгоритми хешування (MD5, SHA, HMAC).
3. Crypto.Protocol – криптографічні протоколи (Chaffing, AONT, KDF).
4. Crypto.PublicKey – алгоритми асиметричного шифрування для створення електронних підписів (RSA, DSA).
5. Crypto.Util – велика кількість корисних модулів та функцій (генерація випадкових чисел, функції теорії чисел та інші).

Використання бібліотеки PyCrypto суттєво полегшить реалізацію алгоритму кільцевого підпису та зменшить кількість коду та можливих помилок.

Також, корисною під час реалізації буде бібліотека fastedcsa – легка та потужна бібліотека, що дозволяє генерувати різні типи еліптичних кривих для схеми ECDSA.

4.3 Генерація ключів RSA

Щоб перевірити роботу алгоритмів потрібно згенерувати публічні ключі для всіх можливих підписантів та зберегти секретний ключ особи, що створює підпис. Для цього скористаємося бібліотекою PyCrypto, а саме: модулем Crypto.PublicKey.RSA. Для генерації ключів маємо наступний код:

```
#generator.py
import sys, os, hashlib, random, Crypto.PublicKey.RSA
def _rn():
  return Crypto.PublicKey.RSA.generate(1024, os.urandom);

own_key = Crypto.PublicKey.RSA.generate(1024, os.urandom);
size = int(sys.argv[1])
keys = map(_rn, range(size))

file = open("public_keys.txt", "w")
file.write(str(own_key.e) + ' ' + str(own_key.n) + '\n')
for key in keys:
  file.write(str(key.e) + ' ' + str(key.n) + '\n')
file.close()

file = open("secret_key.txt", "w")
file.write(str(own_key.d))
file.close()
```

Програма generator.py згенерує задану кількість публічних ключів та запише їх разом з публічним ключем підписанта у файл public_keys.txt. Модуль згенерованих ключів буде складатися з 1024 бітів у двійковому записі.

Секретний ключ підписувача буде зберігатися у файлі secret_key.txt.

4.4 Реалізація алгоритму створення кільцевого підпису

Для створення кільцевого підпису потрібно реалізувати три допоміжні методи: хеш-функція, E_k функція та розширення односторонньої функції RSA:

```
def hash(m):
    return int(hashlib.sha256(m).hexdigest(), 16)

def E(k, x):
    m = (str(k) + str(x)).encode('utf-8')
    return int(hashlib.sha1(m).hexdigest(), 16)

def g(x, e, n, l):
    q, r = divmod(x, n)
    if ((q + 1) * n) <= ((1 << l) - 1):
        res = q * n + pow(r, e, n)
    else:
        res = x
    return res
```

Метод hash(m) шифрує повідомлення m надійним алгоритмом SHA256. Функція E_k реалізована, як хешування та використовує алгоритм SHA1. Щоб включити повідомлення m в результат функції, E_k повертає значення хеш-функції від конкатенації рядкового представлення числа x та повідомлення m. Розширення односторонньої функції RSA реалізовано так, як описано в розділі 1.3.2.

Програма ring.py має в основі дві методи: sign(m, keys, secretKey) та verify(m, X, keys).

Метод sign(m, keys, secretKey) приймає на вхід повідомлення m, що потрібно підписати, значення публічних ключів всіх можливих підписантів та

секретний ключ підписанта. Метод sign(m, keys, secretKey) є прямою реалізацію алгоритму описаного в розділі 1.5:

```
def sign(m, keys, secretKey):
    z = random.randint(1, len(keys) - 1)
    keys[0], keys[z] = keys[z], keys[0]

    h = Signature.hash(m.encode('utf-8'))
    n = len(keys)
    l = 1024
    q = 1 << (l - 1)
    s = [None] * n
    u = random.randint(0, q)
    c = v = Signature.E(u, h)

    for i in (list(range(z + 1, n)) + list(range(0, z))):
      s[i] = random.randint(0, q)
      e = Signature.g(s[i], keys[i][0], keys[i][1], l)
      v = Signature.E(v ^ e, h)
      if (i + 1) % n == 0:
        c = v
    s[z] = Signature.g(v ^ u, int(secretKey), keys[z][1], l)
    return [c] + s
```

Результатом роботи методу sign(m, keys, secretKey) є масив довжиною *n+1* (*n* – кількість підписантів), де перше число *v* – є початковим значення комбінуючої функції та верифікаційним для кільцевого підпису (див. 1.3.4), а наступні *n* чисел – випадкові аргументи $x_1, x_2, ..., x_n$ для розширених односторонніх функцій RSA (див. 1.3.2).

4.5 Реалізація алгоритму верифікації кільцевого підпису

Використовуючи допоміжні методи, описані в розділі 4.4, повідомлення m, результат кільцевого підпису X, та публічні ключі keys, маємо наступний метод для верифікації підпису:

```
def verify(m, X, keys):
    h = Signature.hash(m.encode('utf-8'))
    n = len(keys)
    l = 1024

    def _f(i):
      return Signature.g(X[i + 1], keys[i][0], keys[i][1], l)
    y = list(map(_f, range(len(X) - 1)))

    def _g(x, i):
      return Signature.E(x ^ y[i] ,h)
    r = functools.reduce(_g, range(n), X[0])

    return r == X[0]
```

В даному методі рахується результат комбінуючої функції. Якщо значення комбінуючої функції співпадає з верифікаційним числом X[0] (X[0] = v), то підпис вважається дійсним, інакше – ні.

4.6 Тестування реалізації кільцевого підпису

Для тестування ефективності алгоритму, було створено повідомлення “Hello World!” та згенеровано 10000 публічних ключів. На операційній системі macOS з процесором 2.2 GHz Intel Core i7 маємо наступний результат:

- час створення підпису: 0.98434 с.
- час верифікації підпису: 0.95668 с.

Як бачимо для 10000 ключів кожна програма працює менше 1 секунди, хоча насправді, 10000 ключів дуже велика кількість для створення кільцевого підпису, оскільки навіть для декількох ключів неможливо визначити за реальний час хто насправді створив підпис. Це означає, що дана реалізація є ефективною і може використовуватися на серверах з високим навантаженням та створювати велику кількість підписів за короткий час, або один з дуже великою кількістю публічних ключів.

Для тестування алгоритму верифікації було створено велику кількість кільцевих підписів, для яких програма verify.py завжди видавала відповідь “True”. При спробі створити недійсний кільцевий підпис або модифікувати справжній, програма verify.py видає відповідь “False”.

4.7 Створення розбиттів для порогового кільцевого підпису

Як описано в схемі порогового кільцевого підпису (див. 2.4), для її реалізації потрібно створити П таких “розбиттів”, щоб для будь-яких *t* учасників з групи можливих підписантів, знайшлося таке розбиття, де кожен з цих *t* належить окремій групі. Тривіальним рішенням буде перебір всіх можливих підгруп з *t* людей, та створенням таких розбитів, де кожен лежить в

окремій групі. Зрозуміло, що таких груп буде C_n^t, що є занадто багато, тому було розроблено власний алгоритм для створення меншої кількості розбиттів.

Розглянемо частковий випадок, де $t = 2$:

1. Для зручності пронумеруємо всіх людей від 0 до $n - 1$.
2. Розглянемо всі такі i, що $1 \leq i \leq log_2(n)$.
3. Для кожного i розіб'ємо всіх людей на дві групи, де в першій групі будуть всі чий порядковий номер містить 0 на i-ій позиції в двійковому представленні, а в другій групі будуть всі інші.
4. Тоді, для будь-яких двох людей з номерами (x, y) знайдеться така позиція i, в якій значення бітів їх номерів відрізняється.

Таким чином, для $t = 2$достатньо зробити всього $log_2 n$ розбиттів.

Розглянемо випадок коли $t > 2$. Як зрозуміло з попереднього прикладу, що для будь-яких t людей, знайдеться $t - 1$ позиція в двійковому представлені їх номерів така, що набір бітів в цих позиціях буде унікальний для кожного з цих t людей. Звідси можна сформулювати наступний алгоритм:

1. Розглянемо всі можливі позиції $i_1, i_2, \ldots, i_{t-1}$, де $1 \leq i_1 < i_2 < \ldots < i_{t-1} \leq log_2(n)$. Таких позицій буде $C_{log_2(n)}^{t-1}$.
2. Розглянемо всі можливі значення з $t - 1$ біту, таких значень 2^{t-1}.
3. Оскільки значення в позиціях бітів для кожного номеру мають бути унікальні, то потрібно розглянути всі можливі комбінації, що можна утворити з $t - 1$ бітних чисел для t людей, таких комбінацій буде $C_{2^{t-1}}^t$. Позначимо таку комбінацію як $b_1, b_2, \ldots, b_t$, де b_i- $(t - 1)$-бітне число.

4. Створимо розбиття за наступним правилом: в першій групі будуть всі чий порядковий номер в двійковому представленні містить число b_1 в возиціях i_1, i_2, ..., i_{t-1}, в другій групі чий номер містить b_2 і так далі.

5. Тоді, для будь-яких t людей з номерами (x_1, x_2, ..., x_t) знайдуться такі i_1, i_2, ..., i_{t-1}та b_1, b_2, ..., b_t, що для цих людей існує розбиття.

Отже, складність такого алгоритму складає $O(C^{t-1}_{log_2(n)} * C^{t}_{2^{t-1}})$.

Представлений алгоритм є ефективним лише при невеликих t, наприклад для $n = 1024$, $t = 4$, представлений алгоритм матиме всього 8400 розбиттів, в порівнянні з ~10^{12} розбиттів для тривіального алгоритму.

4.8 Реалізація алгоритму модифікованого кільцевого підпису

Метод signMock(m, keys, secretKey) приймає на вхід повідомлення *m*, що потрібно підписати, значення публічних ключів всіх можливих підписантів та секретний ключ підписанта. Метод signMock(m, keys, secretKey) є реалізацію схеми описаної в розділі 2.3.3:

```
def signMock(m, keys, secretKey):
    z = random.randint(1, len(keys) - 1)
    keys[0], keys[z] = keys[z], keys[0]

    h = Signature.hash(m.encode('utf-8'))
    n = len(keys)
    l = 1024
    q = 1 << (l - 1)
    s = [None] * n
```

```
v0 = v = 0

for i in (list(range(0 n)):
   s[i] = random.randint(0, q)
   e = Signature.g(s[i], keys[i][0], keys[i][1], l)
   v = Signature.E(v ^ e, h)
   if i == 0:
      v0 = v

c = v0 ^ v
return [c] + s
```

Результатом роботи методу signMock(m, keys, secretKey) буде масив довжиною $n+1$ (n – кількість підписантів), де перше число γ – є числом “розриву” комбінуючої функції (див. 2.3.2), а наступні n чисел – випадкові аргументи $x_1, x_2, ..., x_n$ для розширених односторонніх функцій RSA (див. 1.3.2).

4.9 Тестування порогового кільцевого підпису

Для тестування ефективності алгоритму, було використано повідомлення “Hello World!”, 30 публічних та 15 секретних ключів. На операційній системі macOS з процесором 2.2 GHz Intel Core i7 маємо наступний результат:

- час створення підпису: 3.164207 с.
- час верифікації підпису: 1.57241 с.

Алгоритм створення порогового кільцевого підпису працює набагато повільніше, в порівнянні зі звичайним кільцевим підписом. Це пояснюється тим, що для створення порогового кільцевого підпису потрібно створити велику кількість “розбиттів”. Якщо використовувати наперед задані розбиття,

час на створення підпису можна значно зменшити. Наприклад, для 30 публічних та 15 секретних ключів, та для вже згенерованих “розбиттів”, час на створення підпису займе 1.38446 с. Таким чином, час на створення “розбиттів” можна не враховувати. Для централізованих систем, “розбиття” для будь-якої кількості підписантів можна порахувати наперед та зберегти їх на сервері, і під час створення нового підпису використати потрібні.

Також, можна зауважити, що час на створення порогового кільцевого підпису можна зменшити за допомогою паралельного програмування. Оскільки, участь у створення підпису беруть щонайменше *t* осіб, можна розподілити створення модифікованих кільцевих підписів порівно, між всіма підписантами.

Для тестування алгоритму верифікації було створено велику кількість пороговоих кільцевих підписів, для яких програма t-verify.py завжди видавала відповідь “True”. При спробі створити недійсний підпис або модифікувати справжній, програма t-verify.py видає відповідь “False”.

4.10 Генерація ключів ECDSA

Для створення простежуваного кільцевого підпису, потрібно щоб у всіх підписантів були визначені публічні ключі на двох еліптичних кривих. Для цього скористаємося бібліотекою fastecdsa:

```
#ecdsa.py
from fastecdsa import keys, curve
curve1 = curve.secp256k1 # Bitcoin curve
curve2 = curve.P256

file = open("ecdsa_info.txt", "w")
```

```
        file.write("p1: " + str(curve1.p) + "\n")
        file.write("q1: " + str(curve1.q) + "\n")
        file.write("P1: " + str(curve1.G.x) + " " + str(curve1.G.y) + "\n")
        file.write("p2: " + str(curve2.p) + "\n")
        file.write("q2: " + str(curve2.q) + "\n")
        file.write("P2: " + str(curve2.G.x) + " " + str(curve2.G.y) + "\n")
        file.close()

        own_key_1 = keys.gen_private_key(curve1)
        own_key_2 = keys.gen_private_key(curve2)
        public_key_1 = keys.get_public_key(own_key_1, curve1)
        public_key_2 = keys.get_public_key(own_key_2, curve2)

        file = open("ecdsa_secret_keys.txt", "w")
        file.write(str(own_key_1) + "\n" + str(own_key_2) + "\n")
        file.close()

        size = int(sys.argv[1])
        file = open("ecdsa_public_keys.txt", "w")
        file.write(str(public_key_1.x) + " " + str(public_key_1.y) + " " +
                str(public_key_2.x) + " " + str(public_key_2.y) + "\n")
        for i in range(size - 1):
          key1 = keys.get_public_key(keys.gen_private_key(curve1), curve1)
          key2 = keys.get_public_key(keys.gen_private_key(curve2), curve1)
            file.write(str(key1.x) + ' ' + str(key1.y) + ' ' + str(key2.x) + ' ' +
str(key2.y) + '\n')
        file.close()
```

Програма ecdsa.py згенерує задану кількість публічних ключів на еліптичних кривих secp256k1(еліптична крива системи Bitcoin) і P256, та запише їх разом з публічними ключами підписанта у файл ecdsa_public_keys.txt. Інформація про еліптичні криві буде зберігатися у файлі ecdsa_info.txt, а про секретні ключі у ecdsa_public_keys.txt.

4.11 Реалізація алгоритму створення простежуваного підпису

Під час реалізація простежуваного кільцевого підпису використаємо допоміжні функції, реалізовані для звичайного кільцевого підпису. Метод traceable_sign(message, q, P1, P2, k1, k2, Q1, Q2) є прямою реалізацією алгоритму описаного в розділі 3.4:

```
#traceable_ring.py
def traceable_sign(message, q, P1, P2, k1, k2, Q1, Q2):
    s = random.randint(1, len(keys) - 1)
    Q1[0], Q1[s] = Q1[s], Q1[0]
    Q2[0], Q2[s] = Q2[s], Q2[0]

    k = Signature.hash(message.encode('utf-8'))
    n = len(Q1)
    x, y, alpha, beta, m = ([None] * n for i in range(5))
    for i in range(n):
      x[i] = random.randint(1, q - 1)
      y[i] = random.randint(1, q - 1)
      alpha[i] = x[i] * P1 + y[i] * Q1[i]
      beta[i] = -Signature.hash(alpha[i]) * Signature.inv(y[i]) % q
```

```
u = random.randint(1, q - 1)
v = c = Signature.E(u, k)
for i in (list(range(s + 1, n)) + list(range(0, s))):
   c = Signature.E(v ^ m[i], h)
   if (i + 1) % n == 0:
      v = c
m[s] = u ^ c
alpha[s] = k2 * P1
beta[s] = (m[s] + Signature.hash(alpha[s]) * k1) *
            Signature.inv(k2, q) % q1

r, c, L, R, = ([None] * n for i in range(4))
sum_c = 0
C = 0
for i in range(n):
   r[i] = random.randint(1, n - 1)
   c[i] = random.randint(1, n - 1)
   if (i == s):
      L[i] = r[i] * P1
      R[i] = r[i] * P2
   else:
      L[i] = r[i] * P1 + c[i] * alpha[i]
      R[i] = r[i] * P2 + c[i] * Q2[i]
   sum_c = (sum_c + c[i]) % n
   C = Signature.E(C ^ L[i] ^ R[i], k)
c[s] = (C - sum_c + n) % n
r[s] = r[s] - (c[s] * k2) % q
return [v] + m + alpha + beta + c + r
```

Результатом роботи методу traceable_sign(m, keys, secretKey) буде масив довжиною $5 * n + 1$ (n – кількість підписантів), де перше число v – є верифікаційним значення комбінуючої функції, наступні $3 * n$ чисел це набір набір трійок $ECDSA(m_i, \alpha_i, \beta_i)$, і останні $2 * n$ чисел це набі пар чисел (c_i, r_i) для доказу підпису.

4.12 Реалізація алгоритму верифікації простежуваного підпису

Використовуючи результат простежуваного кільцевого підпису $\{v, ECDSA(m_i, \alpha_i, \beta_i), (c_i, r_i)\}$, повідомлення message, а також інформацію про еліптичні криві, маємо наступний метод для верифікації підпису:

```
def traceable_verify(message, q, P1, P2, Q1, Q2, v, m, alpha, beta, c, r):
    k = Signature.hash(m.encode('utf-8'))
    n = len(Q1)

    for i in range(n):
        inv_beta = Signature.inv(beta[i])
        check_alpha = (m[i] * inv_beta * P1 +
                       Signature.hash(alpha[i]) * inv_beta * Q1[i]) % q
        if (alpha[i] != check_alpha):
            return False

    check_v = v
    for i in range(n):
        check_v = Signature.E(check_v ^ m[i], h)
    if (check_v != v):
        return False
```

```
sum_c = 0
L = R = ([None] * n for i in range(2))
C = 0
for i in range(n):
   L[i] = r[i] * P1 + c[i] * alpha[i]
   R[i] = r[i] * P2 + c[i] * Q2[i]
   sum_c = (sum_c + c[i]) % n
   C = Signature.E(C ^ L[i] ^ R[i], k)
return sum_c == C
```

В даному методі спочатку перевіряється правильність даних $ECDSA(m_i, \alpha_i, \beta_i)$, потім рахується результат комбінуючої функції та звіряється з верифікаційним числом v. В кінці перевіряється доказ, що існує таке s, що $\alpha_s = k_2 P_1$.

ВИСНОВКИ

В даній роботі було досліджено різні схеми кільцевого підпису. В основі базової схеми кільцевого підпису лежить цікава ідея комбінуючої функції, завдяки якій підписант може визначати невідомий аргумент лише за допомогою власного секретного ключа.

Схема кільцевого підпису, що базується на RSA підписах, є дуже ефективною та надійною. Підписання та верифікація мають асимптотичний час $O(n)$, тобто алгоритм прямо пропорційно пов'язаний з кількістю використаних відкритих ключів. Під час виконання роботи було реалізовано схему кільцевого підпису та створено зручний інтерфейс для її використання на мові Python.

Кільцевий підпис має різні модифікації, що дає можливість створювати більш практичні та спеціалізовані алгоритми. Було досліджено схеми порогового кільцевого підпису (підпис вважається дійсним, якщо його підписали не менше *t* з *n* учасників) і простежуваного кільцевого підпису (при повторному підписанні розкривається відкритий ключ підписанта), а також запропоновані власні модифікації та створено програмний засіб на мові Python, що дозволяє автоматично створювати ці підписи.

Як було досліджено в роботі, пороговий кільцевий підпис має більш складну схему та більший розмір (в кращому випадку $O(2^t n \log_2 n)$). Асимптотично найскладнішим етапом в схемі порогового підпису є створення “розбиттів”, тому було запропоновано алгоритм, що ефективно створює розбиття для невеликих *t*.

Досліджені схеми простежуваного кільцевого підпису мають менший розмір в порівнянні з пороговим і асимптотично однаковий зі звичайним кільцевим підписом, проте мають більше складне математичне обґрунтування. Хоч і простежуваний підпис можна зробити простішим на схемі RSA, в даній роботі було досліджено модифікації, що базуються на еліптичних кривих, оскільки такі підписи мають застосування в криптовалютах. Для прикладу в реалізації було використано еліптичну криву з криптовалюти Bitcoin.

В порівнянні з іншими схемами, реалізовані схеми є більш зручними для використання та не менш надійними. Водночас варто зауважити, що з настанням ери квантових комп’ютерів знадобляться нові схеми, що базуються на теорії кодування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. https://uk.wikipedia.org/wiki/Електронний_цифровий_підпис#Механізм_ЕЦП [Веб-сайт].
2. https://en.wikipedia.org/wiki/Ring_signature [Веб-сайт].
3. Ronald L.Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret [Стаття] : 2001.
4. Emmanuel Bresson, Jacques Stern, Michael Szydlo.Threshold Ring Signatures and Applications to Ad-hoc Groups [Стаття].
5. Liu, Joseph K.; Wong, Duncan S. Linkable ring signatures: Security models and new schemes [Стаття] : 2005.
6. Eiichiro Fujisaki, Koutarou Suzuki. Traceable ring signature [Стаття].
7. H. Shacham and B. Waters, Efficient ring signatures without random oracles [Стаття] : 2007.
8. J. Kar. Online/off-line ring signature scheme with provable security [Стаття] : 2015.
9. K. Wang, Y. Mu, and W. Susilo. Identity-based quotable ring signature [Стаття] : 2015.
10. S. Zeng, Q. Li, Z. Qin, and Q. Lu. Non-interactive deniable ring signature without random oracles [Стаття] : 2016.
11. https://cryptonote.org/inside#untraceable-payments [Веб-сайт].
12. http://shadow.cash/downloads/shadowcash-anon.pdf [Стаття].
13. N. Shirsath Priyanka and K. Becomp. Data Sharing in Cloud Using Identity Based Ring Signature [Стаття].
14. B. Wang, B. Li, and H. Li. Oruta: Privacy-preserving public auditing for shared data in the cloud [Стаття].

15. https://en.wikipedia.org/wiki/RSA_(cryptosystem) [Веб-сайт].
16. https://en.wikipedia.org/wiki/Hash_function [Веб-сайт].
17. E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures and applications to ad-hoc groups [Стаття] : Berlin, Germany : 2002.
18. A. Petzoldt, S. Bulygin, and J. Buchmann. A multivariate based threshold ring signature scheme [Стаття] : 2013.
19. H. Wang and S. Han. A provably secure threshold ring signature scheme in certificateless cryptography [Стаття] : 2010.
20. H. Xiong, Z. Qin, F. Li, and J. Jin. Identity-based threshold ring signature without pairings [Стаття] : 2008.
21. L. Dallot and D. Vergnaud. Provably secure code-based threshold ring signatures [Стаття] : 2009.
22. C. Aguilar Melchor, P.-L. Cayrel, P. Gaborit, and F. Laguillaumie. A new efficient threshold ring signature scheme based on coding theory [Стаття] : 2011.
23. https://www.hindawi.com/journals/scn/2017/1915239/ [Веб-сайт].
24. https://www.sciencedirect.com/science/article/abs/pii/S221421261830393 4 [Веб-сайт].
25. N. Alon, R. Yuster and U. Zwick. Color Coding [Стаття].
26. https://en.wikipedia.org/wiki/Python_(programming_language) [Веб-сайт].
27. https://www.dlitz.net/software/pycrypto/api/2.6/ [Веб-сайт].
28. Xavier Bultel, Pascal Lafourcade. k-Times Full Traceable Ring Signature [Стаття] : 2016.
29. Ke Gu, Na Wu. Constant Size Traceable Ring Signature Scheme without Random Oracles [Стаття] : 2018.
30. Ke Gu, Xinying Dong and Linyu Wang. Efficient traceable ring signature scheme without pairings [Стаття] : 2020.

31. Hanwen Feng, Jianwei Liu, Qianhong Wu, Ya-Nan Li. Traceable Ring Signatures with Postquantum Security : RSAConference2020.
32. Fei Tang, Junjie Pang, Kefei Cheng, and Qianhong Gong. Multiauthority Traceable Ring Signature Scheme for Smart Grid Based on Blockchain [Стаття] : 2021.
33. https://cryptonote.org/whitepaper.pdf [Веб-сайт].
34. Shen Noether. Ring Confidential Transactions [Стаття]
35. https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm [Веб-сайт].