# ДОДАТОК А

## Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії

25. Vlasenko, L. A., et al. "Stochastic Descriptor Pursuit Game." *Cybernetics and Systems Analysis* (2024): 1-9.

26. Kachko, O. G., et al. "Hash-based cryptography, its security and feasibility in modern cryptosystems" (2024): 1-11.

27. Kachko, O. G., et al. "Substantiation and proposals for the selection, improvement and standardization of the post-quantum electronic signature mechanism at the national and international levels" *Methods of promising cryptographic transformations* (2021): 5-7.

28. Kachko, O. G., et al. "Basic principles and results of comparison of electronic signatures properties of the postquantum period based on algebraic lattices" *Methods and algorithms of cryptographic information protection* (2021): 1-17.

# ДОДАТОК Б

## Коди схем підпису

Нижче наведено код для OTS Лампорта. Використовується Python в Jupyter Notebook.

```
import numpy as np
import hashlib

def signing_key(message):
    message = str(message)
    message =  ''.join(format(ord(i), '08b') for i in message)
    message = [int(i) for i in message]
    k = len(message) # Security Parameter
    return np.random.randint(0,2,size=(2*k,k)).tolist()

def verifying_key(key):

    verification_key = []

    for c in key:
        column = []
        for r in c:
            column.append(hashlib.sha256(str(r).encode()).hexdigest())

        verification_key.append(column)

    return verification_key

def signature(message,key):

    message = str(message)
    message =  ''.join(format(ord(i), '08b') for i in message)
    message = [int(i) for i in message]

    signature = []
    for idx, bit in enumerate(message):
        signature.append(key[2*idx+bit])

    return signature

def verification(message,verification_key,signature):

    message = str(message)
    message =  ''.join(format(ord(i), '08b') for i in message)
    message = [int(i) for i in message]

    hashed_signature = []

    if len(verification_key[0]) != len(message):
```

```
            return False

        else:
            for c in signature:
                column = []
                for r in c:

column.append(hashlib.sha256(str(r).encode()).hexdigest())

                hashed_signature.append(column)

            for idx, bit in enumerate(message):

                if verification_key[2*idx+bit] != hashed_signature[idx]:
                    return False

            return True


message = "Shine On You Crazy Diamond."
key=signing_key(message)
verification_key=verifying_key(key)
signature=signature(message,key)
verification(message,verification_key,signature)
```

Класи багаторазового підпису на дереві Меркла.

**LamportSignature**

```
from bitstring import BitArray
import hashlib
from os import urandom

class LamportSignature:
    def __init__(self):
            self.private_key = self.generate_private_key()
            self.public_key = self.generate_public_key()
            self.used = False

        def generate_private_key():

            return [(bytearray(urandom(32)), bytearray(urandom(32))) for i in
range(256)]

        def generate_public_key(self):
            return [(self.hash(a), self.hash(b)) for (a, b) in
self.private_key]

        def concatenate_key(key_list):


            ret = bytearray(0)
            if type(key_list[0]) is tuple:
```

```
            for a, b in key_list:
                ret += a + b
        else:
            for i in key_list:
                ret += i
        return ret

    def decatenate_key(key):

        if len(key) == 8192:  # signature key size 256×256 bits
            return [key[i:i + 32] for i in range(0, 8192, 32)]
        elif len(key) == 16384:  # public/private key size 2x256×256 bits
            ret = []
            for i in range(0, 16384, 64):
                ret.append((key[i:i + 32], key[i + 32:i + 64]))
            return ret
        else:
            raise ValueError("Wrong key size.")

    def get_key(self, key_type, concatenate):

        key = self.public_key if key_type == 'public' else
self.private_key
        if not concatenate:
            return key
        return self.concatenate_key(key)

    def sign(self, msg):
        if self.used:
            raise ValueError("Private and public keys already used!")
        self.used = True
        msg_hash = self.hash(msg)
        signature = []
        for (a, b), bit in zip(self.private_key,
BitArray(bytes=msg_hash).bin):
            if bit == "0":
                signature.append(a)
            elif bit == "1":
                signature.append(b)
        return signature

    def verify(cls, msg, signature, public_key):

        msg_hash = cls.hash(msg)
        signature_hash = [cls.hash(i) for i in signature]
        for sig_hash, (a, b), bit in zip(signature_hash, public_key,
BitArray(bytes=msg_hash).bin):
            if (bit == "0" and sig_hash != a) or (bit == "1" and sig_hash
!= b):
                return False
        return True

    def hash(data):

        if type(data) is not bytearray:
            data = data.encode('utf-8')
        return bytearray(hashlib.sha256(data).digest())
```

```
def main():
    for msg_sent, msg_to_check in (("abc", "abc"), ("abc", "aaa"),
("abc", "aabc")):
        lamport = LamportSignature()
        signature = lamport.sign(msg_sent)
        print(msg_sent, msg_to_check,
LamportSignature.verify(msg_to_check, signature, lamport.public_key))


if __name__ == "__main__":
    main()
```

## MerkleTree

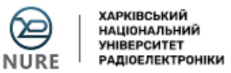
```
import hashlib
from math import log2, floor

class MerkleTree:

    def __init__(self, n_leaves):
        """MerkleTree object constructor.

        Args:
            n_leaves (int): Number of leaves in the tree.

        """
        if floor(log2(n_leaves)) != log2(n_leaves):
            raise ValueError("Wrong number of leaves.")
        self.tree = {}
        self.n_levels = None
        self.n_leaves = n_leaves

    def add_node(self, data, position, hashed=False):

        if data is None:
            self.tree[position] = None
        elif hashed and type(data) is str:
            self.tree[position] = bytearray.fromhex(data)
        elif hashed and type(data) is bytearray:
            self.tree[position] = data
        else:
            self.tree[position] = self.hash(data)

    def generate_tree(self):

        self.n_levels = int(log2(self.n_leaves)) + 1
        for level in range(self.n_levels):
            for pos in range(int(self.n_leaves / 2 ** level)):
                if (level, pos) not in self.tree:
                    self.tree[(level, pos)] = None

        for level in range(1, self.n_levels):
            for pos in range(int(self.n_leaves / 2 ** level)):
                left_child = self.tree[(level - 1, 2 * pos)]
                right_child = self.tree[(level - 1, 2 * pos + 1)]
```

```
                if left_child is not None and right_child is not None:
                    self.tree[(level, pos)] = self.hash(left_child +
right_child)

    def get_root(self):

        return self.tree[(self.n_levels - 1, 0)]

    def get_brother_node_hash(self, position):

        try:
            return self.tree[self.get_brother_node_position(position)]
        except:
            raise ValueError("No brother exists.")


    def get_brother_node_position(position):

        index = position[1] + 1 if position[1] % 2 == 0 else position[1]
- 1
        return position[0], index

    def get_authentification_path_hashes(self, index):

        return [self.tree[i] for i in
self.get_authentification_path(index)]

    def get_authentification_path(self, index):

        auth_path = []
        for level in range(self.n_levels - 1):
            auth_path.append(self.get_brother_node_position((level,
index)))
            index = int(floor(index / 2))  # Parent's node index.
        return auth_path

    def hash(data):

        if type(data) is not bytearray:
            data = data.encode('utf-8')
        return bytearray(hashlib.sha256(data).digest())

def main():
    mk = MerkleTree(n_leaves=8)
    mk.add_node("test", (0, 0))
    mk.add_node("retest", (0, 1))
    mk.add_node("test", (0, 2))
    mk.add_node("world", (0, 3))
    mk.add_node("test", (0, 4))
    mk.add_node("retest", (0, 5))
    mk.add_node("test", (0, 6))
    mk.add_node("world", (0, 7))
    mk.generate_tree()

    for key, value in mk.tree.items():
        print(key, value)
```

```
    print(mk.get_authentification_path(2))
    print(mk.get_authentification_path_hashes(2))


if __name__ == "__main__":
    main()
```

ДОДАТОК В

Слайди до презентації


**Дослідження постквантумних алгоритмів. Цифровий підпис.**

Неділько Д.С., ІПЗм-22-2
Науковий керівник: проф. Власенко Л.А.


25 червня 2024

# Дослідження

***Об'єктом дослідження*** *є* процеси в постквантових системах підпису .
**Предмет дослідження** – методи і засоби створення сучасних систем підпису

# Постановка задачі

У ході даної кваліфікаційної роботи потрібно провести дослідження предметної області застосування постквантових алгоритмів цифрового підпису, аналіз методів і засобів створення сучасних систем підпису, які стійкі до квантових алгоритмів. Буде проведено порівняльний аналіз різних постквантових алгоритмів цифрових підписів, порівняння їх з огляду на ефективність, продуктивність та стійкість до квантових атак. Ця робота спрямована на виявлення найкращих алгоритмів для створення сучасних систем цифрового підпису, стійких до квантових обчислень, та буде корисною для криптографів і розробників систем безпеки.


# Методологія

***Методика дослідження***:

- Аналіз схем підписів
- Порівння алгоритмів підписів
- Моделювання та порівняльний аналіз

# Квантові обчислення

$$a|0\rangle + b|1\rangle$$

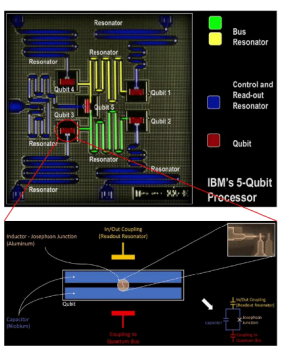
$|a|^2$ : ймовірність вимірювання 0

$|b|^2$ : ймовірність вимірювання 1

## Два стани замість одного

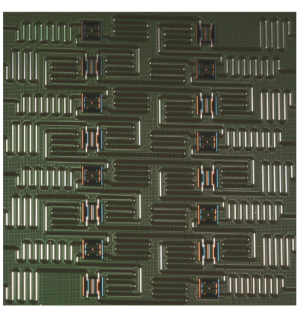
В процесі обчислень проходять маніпуляції з кубітами. Для класичних змінних група n біт зберігає єдине з  можливих значень на кожному етапі обчислень. Але група з n кубітів зберігає вже два в ступені n можливих значень із визначеною ймовірністю вимірювання кожного окремого стану.

# Реалізація квантового комп’ютера




**5 кубіт**

Нещодавно корпорація IBM представила перший модульний квантовий комп'ютер IBM Quantum System Two на базі новітнього 133-кубітного квантового процесора Heron. Вчені з Китаю повідомляють, що їх новий квантовий комп'ютер вирішив надзвичайно складне математичне завдання за мільйонну частку секунди. Це на 20 мільярдів років швидше, ніж міг би впоратися з таким самим завданням найшвидший у світі суперкомп'ютер.



**16 кубіт**

# Постквантова криптографія

### Криптографія на основі хешу

Ці схеми ґрунтуються на звичайних вимогах до безпеки хеш-функцій і вимагають менших передумов про безпеку, ніж теоретико-числові схеми. SPHINCS+

### Криптографія на основі коду

Ця криптосистема заснована на кодах з виправленням помилок для побудови односторонньої функції

### Багатоваріантна криптографія

Безпека багатоваріантної (багатовимірної) криптографії залежить від складності вирішення багатовимірних систем рівнянь. Rainbow

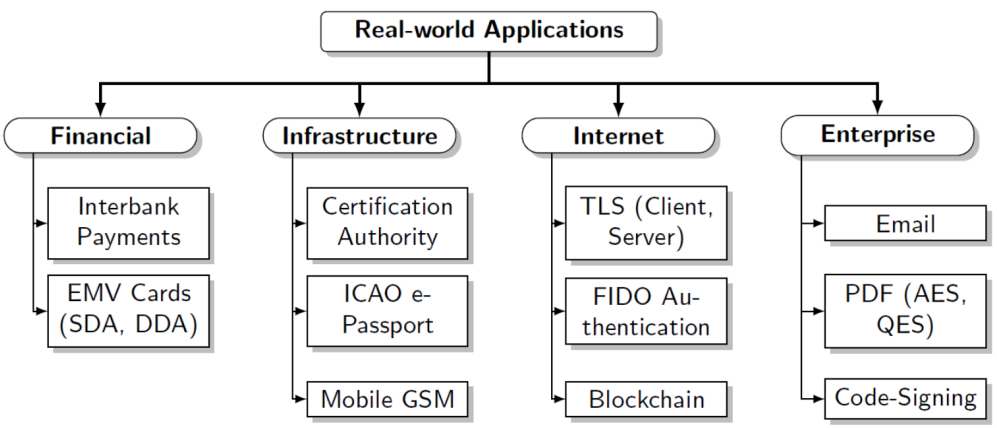
### Криптографія на ґратках

Криптографія на основі решітки є однією з успішних схем третього раунду процесу стандартизації NIST. В якості схем фіналістів обрано схеми шифрування на основі решітки Kyber, схеми підпису CRYSTAL Dilithium і Falcon

### Криптографія на основі ізогенії

Ці схеми забезпечується в рамках задачі побудови ізогенії між двома супер сингулярними кривими з однаковою кількістю точок. NIST немає.

# Цифровий підпис на практиці



# Схема цифрового підпису

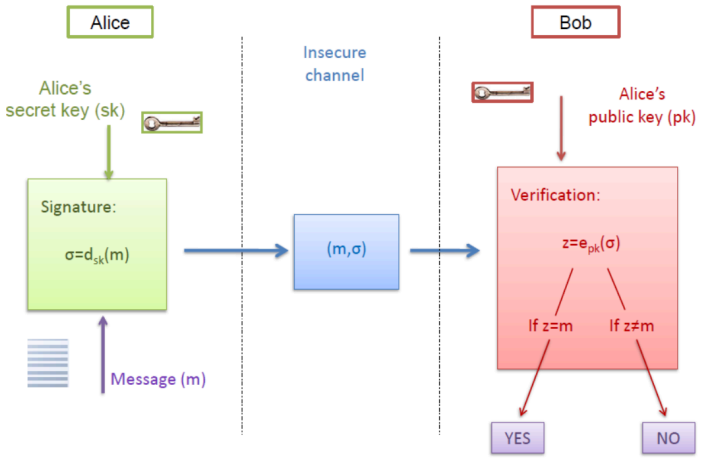


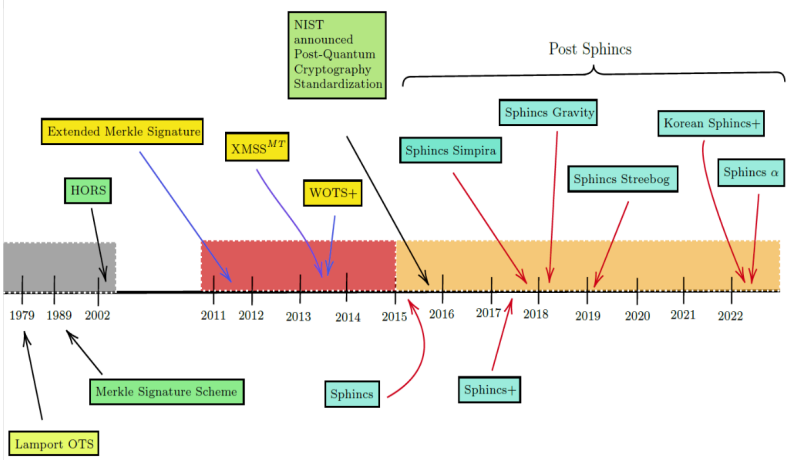

Схема цифрового підпису — це кортеж алгоритмів (KeyGen; Sign; Verify), визначених таким чином:

• Алгоритм генерації ключів KeyGen — це ймовірнісний алгоритм, який виводить відкритий ключ pk і секретний ключ sk, тобто пару ключів (pk; sk).

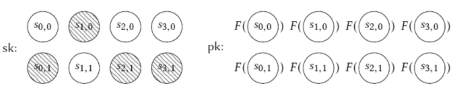
• Алгоритм підпису Sign — це можливо ймовірнісний алгоритм, який приймає як вхідні дані повідомлення і секретний ключ sk для створення підпису.

• Алгоритм перевірки Verify — це детермінований алгоритм, який приймає як вхідні дані повідомлення m, підпис і відкритий ключ pk. Він виводить логічне значення True, щоб вказати, що підпис прийнято, або False, щоб вказати відхилення.

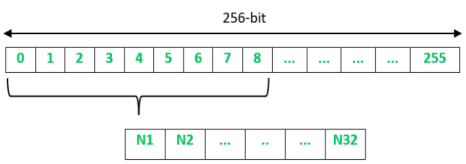
# АЛГОРИТМИ ПІДПИСІВ НА ОСНОВІ ХЕШУ



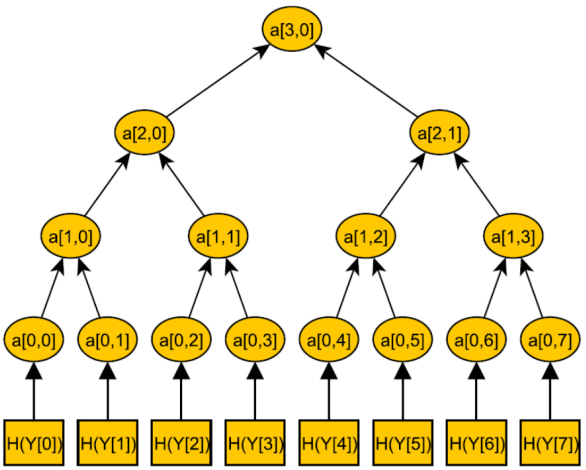

## Схема OTS Лампорта



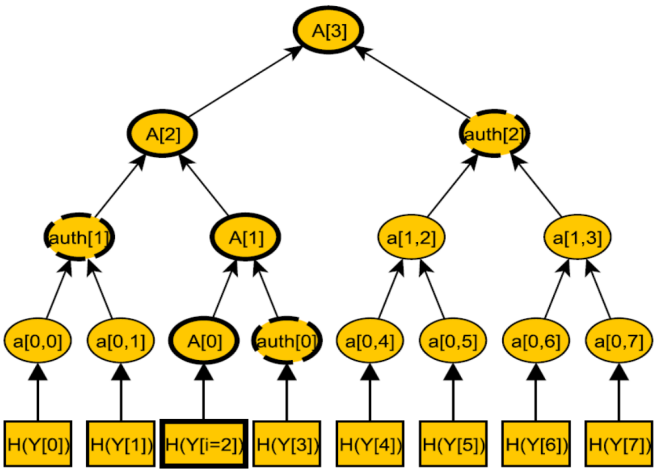

## Схема Вінтерніца



## Дерева Меркла



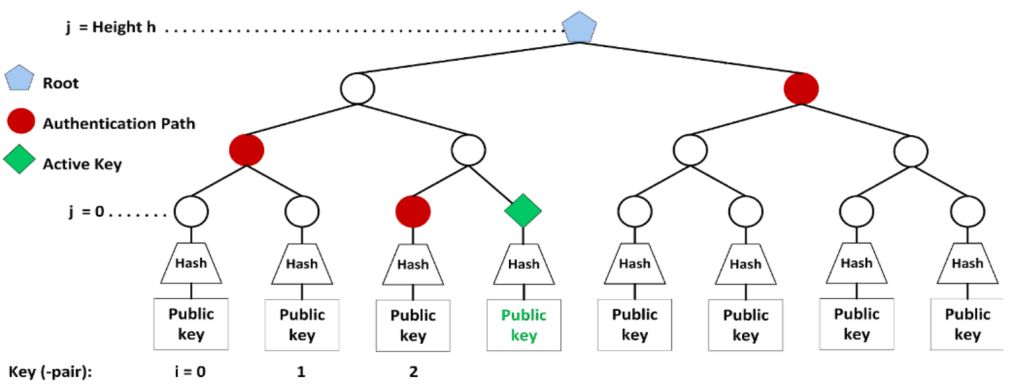

**Дерево Меркла з вісьмома листками**




**Дерево Меркла із шляхом A та шляхом автентифікації**

## Багаторазовий підпис на базі дерева Меркла. Імплементація Python

Щоб створити підпис повідомлення, Аліса вибирає пару відкритих/приватних ключів Лампорта, яка ще не використовувалася, і створює підпис Лампорта, пов'язаний із повідомленням. На рисунку 3.1 нижче обрана пара 3. Далі Аліса шукає шлях автентифікації, який складається з вузлів, необхідних для обчислення кореня Меркла, знаючи початковий вузол. Аліса передає Бобу номер пари ключів Лампорта, повідомлення та підпис Меркла, що складається з: 1. Підпис Лампорта (8 КіБ). 2. Відкритий ключ Лампорта, який використовувався для генерації підпису (16 КіБ)3. Шлях автентифікації (256 біт * 3 для дерева на рисунку). Розмір підпису Merkle становить приблизно 24 КіБ



# Класи

Щоб реалізувати алгоритм підпису на основі дерев Меркла, треба створити класи MerkleTree, LamportSignature, а також, по необхідності, класи Client і Server для частини моделювання мережі.

Клас MerkleTree має атрибути дерева, n_levels і n_leaves. Перший представляє дерево у формі словника кортежів (позиція, дані), причому сама позиція є кортежем (рівень, індекс). Ми рахуємо в дереві знизу вгору та зліва направо, тому корінь знаходиться в позиції (level_max, 0). Висоту дерева представляє n_levels (ми рахуємо від 0, тому простий корінь має висоту 0). Нарешті, n_leaves представляє кількість листя на дереві.

Клас LamportSignature має атрибути private_key, public_key і used. Перші два містять приватний і відкритий ключі, використані є логічним значенням, яке вказує, чи підпис уже використовувався. Два ключі містять 256 пар по 32 байти, тобто 256 біт: кожен ключ містить 2x256x256 біт або 16 КіБ. Під час створення об'єкта LamportSignature генерується випадковий закритий ключ, а з нього відповідний відкритий ключ.


## АНАЛІЗ АЛГОРИТМІВ ОБХОДУ ДЕРЕВА МЕРКЛА

Нехай Ti (i =1,2,3) — дерева Меркла висотою 1 H =5, 2 H =10 і 3 H = 20. Вимоги до часу для цих дерев (у кількості елементарних операцій)

Таблиця 1 – Вимоги до часу в кількості елементарних операцій

| Техніка обходу | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| Класичний Меркл | 10 | 20 | 40 |
| Вдосконалений | 5 | 10 | 20 |
| Фрактальний | 4.3068 | 6.0207 | 9.2552 |

Таблиця 2 – Вимоги до простору в кількості елементарних операцій

| Техніка обходу | Вимоги до простору для $T_1$ | Вимоги до простору для $T_2$ | Вимоги до простору для $T_3$ |
|---|---|---|---|
| Класичний Меркл | 12.50 | 50 | 200 |
| Вдосконалений | 15 | 30 | 60 |
| Фрактальний | 8.0753 | 22.5774 | 69.4138 |


# Вимоги до простору для різних хешів

Таблиця 3 – Вимоги до простору для 160-бітної хеш-функції

| Техніка обходу | Вимоги до простору для $T_1$ (кбіт) | Вимоги до простору для $T_2$ (кбіт) | Вимоги до простору для $T_3$ (кбіт) |
|---|---|---|---|
| Класичний Меркл | 0.250 | 1 | 4 |
| Вдосконалений | 0.3 | 0.6 | 1.2 |
| Фрактальний | 0.1615 | 0.4515 | 1.3883 |

Таблиця 4 – Вимоги до простору для 256-бітної хеш-функції

| Техніка обходу | Вимоги до простору для $T_1$ (кбіт) | Вимоги до простору для $T_2$ (кбіт) | Вимоги до простору для $T_3$ (кбіт) |
|---|---|---|---|
| Класичний Меркл | 0.250 | 1.6 | 6.4 |
| Вдосконалений | 0.48 | 0.96 | 1.92 |
| Фрактальний | 0.2584 | 0.7225 | 2.2212 |

# Висновки

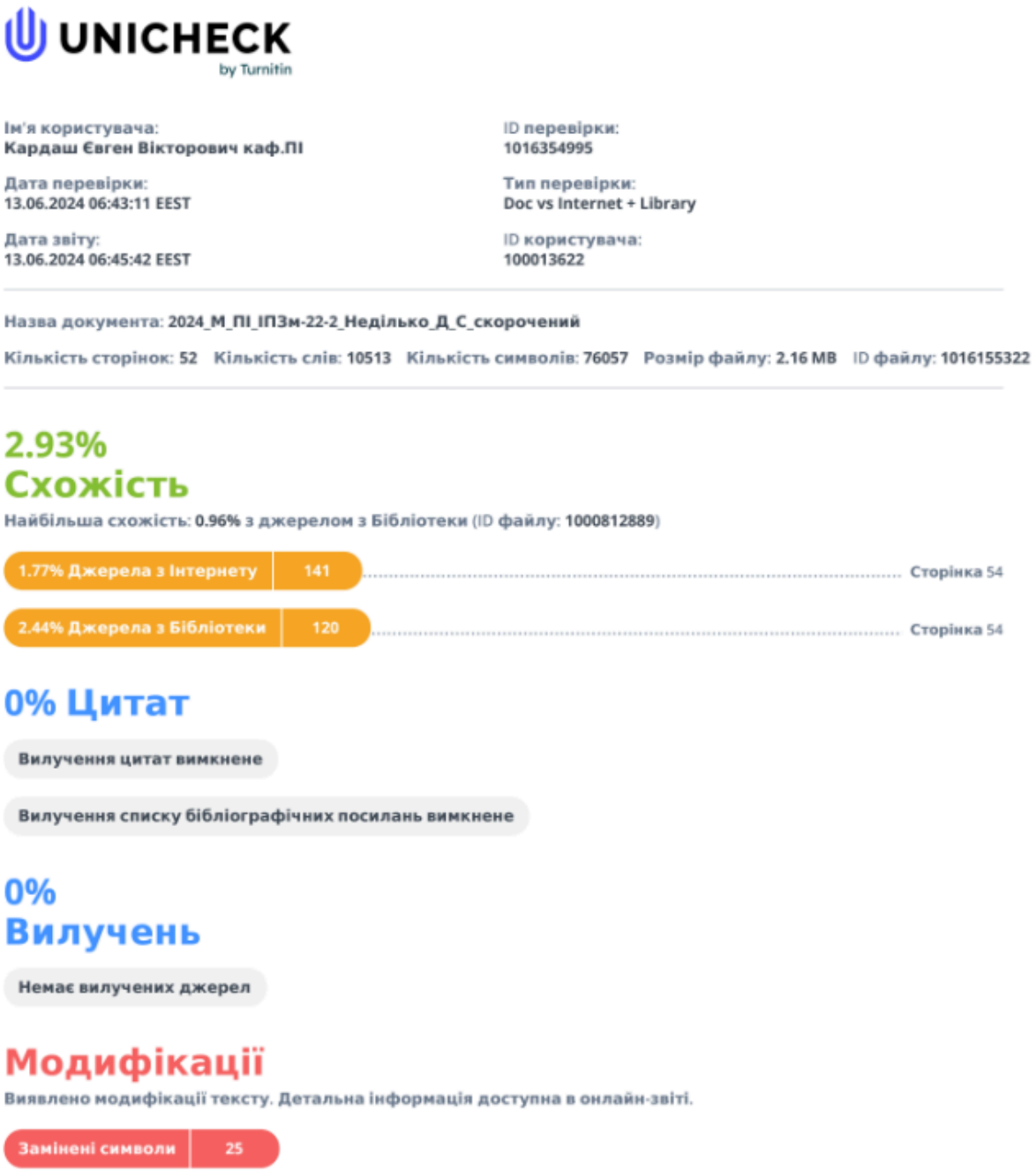
У роботі проведено аналіз шляхів підвищення ефективності побудови сучасних постквантових алгоритмів цифрових підписів. Було проведено огляд різних напрямів досліджень, які вивчаються в постквантовій криптографії. Розглянуто схеми цифрового підпису, проведено порівняльний аналіз постквантових алгоритмів. На основі цього аналізу виявлено недоліки та переваги схем підпису. Проведено аналіз різноманітних схем підписів на основі хеша. Детально розглянуто алгоритм OTS Лампорта, який лежить в основі більшості постквантових схем підписів. Для більш глибокого вивчення алгоритмів цифрових підписів була розроблена робоча модель OTC Лампорта, яка була реалізована в коді мовою Python. На основі цього моделювалося дерево Меркла для отримання багаторазових підписів. Аналізувався розмір відкритих і закритих ключів, розмір самого підпису при різних вхідних параметрах.

Далі вивчалась проблема ефективного обчислення наступного шляху автентифікації, необхідного для схеми підпису Меркла. Проблема обходу дерева Меркла відповідає на питання, як ефективно обчислювати шлях автентифікації для всіх листів один за одним, починаючи з першого. В цьому ракурсі проведен аналіз подібності та відмінності між алгоритмами підпису, було розглянуто вимоги до часу та простору для кожної техніки обходу з порівнянням їх ефективність у різних ситуаціях. Розглянуті аргументи про класичний обхід дерева та його вдосконалення. Наведено порівняльний аналіз роботи трьох алгоритмів для дерева Меркла фіксованих розмірів.

На основі моделювання та вивчення запропонованих схем підписів у цій роботі запропоновано конкретну реалізацію системи на основі дерева Меркла. Розроблено працюючі коди схем підписів. Результати дослідження мають значення для систем постквантових алгоритмів підписів. По-перше, дослідження показує, що структури наподобие дерева Меркла є найбільш перспективними кандидатами для створення ефективного шаблону постквантових підписів. По-друге, дослідження підкреслює важливість вирішення ключових проблем, пов'язаних з шляхом автентифікації та проблемою ефективного обходу дерева.

# ДОДАТОК Г

## Результати перевірки на академічний плагіат



UNICHECK
by Turnitin

Ім'я користувача:
Кардаш Євген Вікторович каф.ПІ

ID перевірки:
1016354995

Дата перевірки:
13.06.2024 06:43:11 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2024 06:45:42 EEST

ID користувача:
100013622

Назва документа: 2024_М_ПІ_ІПЗм-22-2_Неділько_Д_С_скорочений

Кількість сторінок: 52 Кількість слів: 10513 Кількість символів: 76057 Розмір файлу: 2.16 MB ID файлу: 1016155322

2.93%
Схожість

Найбільша схожість: 0.96% з джерелом з Бібліотеки (ID файлу: 1000812889)

1.77% Джерела з Інтернету 141 ..... Сторінка 54

2.44% Джерела з Бібліотеки 120 ..... Сторінка 54

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 25

# ДОДАТОК Ж

## Експертний висновок результатів перевірки роботи

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-22-2 (група)

Неділько Д.С
(прізвище, ім'я, по батькові)

Зауваження

| Пункт ДСТУ 3008-2015 | Зміст пункту | Сторінка кваліфікаційної роботи |
|---|---|---|
| 1 | 2 | 3 |
| | **7.6 Таблиці** | |
| 7.7.2 | Якщо подають переліки одного рівня підпорядкованості, на які у звіті немає посилань, то перед кожним із переліків ставлять знак «тире».<br>Якщо у звіті є посилання на переліки, підпорядкованість позначають малими літерами української абетки, далі — арабськими цифрами, далі — через знаки «тире». Після цифри або літери певної позиції переліку ставлять круглу дужку. | 20, далі за текстом. |
| 7.10.1 | Формули та рівняння подають посередині сторінки симетрично тексту окремим рядком безпосередньо після тексту, у якому їх згадано.<br>Найвище та найнижче розташування запису формул(и) та/чи рівняння(-нь) має бути на відстані не менше ніж один рядок від попереднього й наступного тексту. | 51 |
| Методичні вказівки до виконання кваліфікаційної роботи магістра… **ЗАТВЕРДЖЕНО** кафедрою ІП протокол № 5 від 13.11.2023р. 3.2 Оформлення пояснювальної записки згідно з ДСТУ 3008:2015 Звіти у сфері науки і техніки. Структура та правила оформлювання. **Шаблон** затверджений засіданням кафедри №3 від 16.10.2023. | Рисунок повинен розміщуватися одразу після його згадування у тексті, або на наступній сторінці. Під рисунком повинен бути підпис із словом Рисунок, порядковим номером цього рисунку, через тире з великої літери – назва рисунку та **в круглих дужках вказується джерело з якого взятий цей рисунок, або то, що його виконано самостійно.** | 12 |
| Методичні вказівки до виконання кваліфікаційної роботи магістра… **ЗАТВЕРДЖЕНО** кафедрою ІП протокол № 5 від 13.11.2023р. 3.2 Оформлення пояснювальної записки згідно з ДСТУ 3008:2015 Звіти у сфері науки і техніки. Структура та правила оформлювання. **Шаблон** затверджений засіданням кафедри №3 від 16.10.2023. | Назву таблиці друкують з великої літери і розміщують над таблицею з абзацного відступу та **в круглих дужках вказується джерело з якого взята ця таблиця, або то, що вона виконана самостійно. ПРИКЛАД: шаблон, стор.15** | 46 |

Експерт _____ Вадим НЕЧВОЛОД

(підпис) (прізвище, ініціали)

18.06.2024

ДОДАТОК Е

Апробація результатів кваліфікаційної роботи

## SECTION: COMPUTER ENGINEERING

# ДОСЛІДЖЕННЯ ПОСТКВАНТОВИХ АЛГОРИТМІВ. ЦИФРОВИЙ ПІДПИС

**Власенко Лариса Андріївна**
доктор технічних наук, професор
**Неділько Дмитро Сергійович**
здобувач вищої освіти магістерського рівня
dmytro.nedilko@nure.ua
Кафедра програмної інженерії
Харківський національний університет радіоелектроніки, Україна

Криптографія використовувалася з незапам'ятних часів для збереження конфіденційності інформації під час запису або передачі. Дослідження криптографії також еволюціонували від класичного шифру Цезаря до сучасних криптосистем [1], а також до криптосистем, заснованих на квантових обчисленнях. Поява квантових обчислень створює серйозну загрозу для сучасних криптосистем. Ця загроза спровокувала дослідження постквантової криптографії для розробки постквантових алгоритмів, здатних протистояти атакам квантових обчислень.

Квантовий комп'ютер — це пристрій, який використовує квантово-фізичні явища для виконання обчислень у спосіб, який принципово відрізняється від класичного комп'ютера [2]. Тоді як класичний комп'ютер у будь-який момент часу перебуває у фіксованому стані — наприклад, бітовий рядок — стан квантового комп'ютера може бути «сумішшю» кількох станів.

У розробці квантових алгоритмів є два новаторські алгоритми, які заклали міцну основу для зламу великої кількості просунутих криптосистем із відкритим ключем. У 1994 році Шор запропонував поліноміально-часовий алгоритм [2] для розв'язування задач цілочисельної факторизації та дискретного логарифмування. Квантовий алгоритм Шора та його варіанти можна використовувати для зламу більшості використовуваних на даний момент криптосистем з відкритим ключем. Квантовий алгоритм, відомий як алгоритм Гровера [2], - це алгоритм пошуку, здатний знаходити елемент у невпорядкованій базі даних із $N = 2^n$ елементів лише за $O\left(\sqrt{N}\right)$ кроків на квантовому комп'ютері. Це квадратичне прискорення порівняно з класичним комп'ютерним підходом, який вимагає в середньому $N/2$ кроків із використанням лінійного пошуку.

Постквантова криптографія (PQC) – це розробка криптографічних алгоритмів, які є безпечними в квантову еру з захистом як від класичних, так і від квантових комп'ютерів. Існує кілька підходів для побудови постквантових

криптографічних схем. Це криптографія на основі хешу [3], криптографія на основі коду [4], багатоваріантна криптографія [5], криптогра́фія на гра́тках [6] та схеми криптографії на основі ізогенії [7]. Практично всі алгоритми можна використовувати для цифрових підписів у тому чи іншому варіанті. Найчастіше вони й створювалися тільки з цією метою.

Для розгляду ми виділили лише ті алгоритми, які переважно використовуються для цифрового підпису. Для цього послідовно розглянемо переваги та недоліки алгоритмів підпису з наступних розділів: криптографія на гра́тках, багатоваріантна криптографія, на основі коду та криптографія на основі хешу.

Почнемо з криптографії на гра́тках. FALCON [8] і CRYSTALS-Dilithium [9] — це два різні криптографічні алгоритми, які запропоновано для постквантової криптографії (PQC). Цей набір містить схеми шифрування, протоколи обміну ключами, схеми цифрового підпису та хеш-функції. Мета розробки CRYSTAL полягає в створенні криптографічних примітивів, несприйнятливих до квантових атак, що робить його придатним для розгортання в постквантовому криптографічному ландшафті.

До недоліків можна віднести розмір відкритого і закритого ключа. І ці схеми погано адаптовані для створення цифрового підпису. До того ж, алгоритми мають високу складність, та існує ймовірність виникнення помилки дешифрування правильно створеного зашифрованого тексту. Але є значні переваги: ефективніше шифрування та дешифрування як у апаратній, так і в програмній реалізації; набагато швидша генерація ключів, що дозволяє використовувати одноразові ключі; низьке використання пам'яті дозволяє використовувати його в таких програмах, як мобільні пристрої та смарт-карти.

Як другий кандидат, ми обираємо представника з табору багатоінваріантної криптографії – це Rainbow [5]. Цей алгоритм використовується лише для підписів, у цьому його перевага. При цьому розмір відкритого ключа дорівнює 124 KB, закритого - 95 KB, і розмір підпису досить великий - 424 KB. Хоча в його основі лежить складність розв'язання систем багатовимірних рівнянь, цей алгоритм страждає від ризику багатьох нападів, таких як: пряма атака, атака мінімального рангу, атака високого рангу, атака UOV, атака узгодження UOV, атаки на хеш-функцію.

Перша криптосистема з відкритим ключем на основі коду була запропонована Робертом МакЕлісом ще у 1978 році [4]. Ця всеїдна криптографічна схема, вона успішно використовується як для шифрування, розшифровки та підпису. Припущення про надійність цієї криптосистеми полягає в тому, що декодування відомих лінійних кодів легко виконується ефективним алгоритмом декодування, але коли лінійний код маскується як загальний лінійний код за допомогою кількох секретних перетворень, декодування стає дуже складним. Шифрування та дешифрування МакЕліса не потребує обчислювально дорогих арифметичних засобів множинної точності. Отже, він призначений також для реалізації на вбудованих пристроях. Основним недоліком криптосистеми МакЕліса з відкритим ключем є дуже

великий відкритий ключ. Наприклад, для 80-бітного рівня безпеки відкритий ключ, який використовується в оригінальній схемі, має розмір 437,75 КВ. Тим не менш, одна з криптосистем цього сімейства успішно пройшла до третього раунду конкурсу NIST (Національного інституту стандартів та технологій).

Тепер розглянемо схеми підпису на основі хешу [3], яка покладається виключно на існування безпечної односторонньої функції. Підписи на основі хешу є одними з найстаріших криптосистем із відкритим ключем, і їхня безпека добре вивчена. Історично великий розмір підпису був важливою перешкодою для практичних схем. Але можливо, найбільша перешкода, однак, має зовсім іншу природу: виробники хеш-підписів повинні були підтримувати та оновлювати стан цієї системи. Замість того, щоб мати можливість створити пару ключів один раз, а потім використовувати їх довільно та нескінченно, секретний ключ ефективно змінюється з кожним підписом. Тем не менш, багато проблем цього сімейства алгоритмів були вирішені останнім часом. Так система SPHINCS+ володіє дуже вражаючими характеристиками: розмір відкритого ключа 32 В, закритого 64 В і розмір підпису всього 8 КВ. Постійно зростаюча загроза великомасштабного квантового комп'ютера відновила інтерес до цієї галузі, і за останнє десятиліття відбувся значний розвиток. Це призвело до XMSS, що демонструє життєздатність підписів на основі хешування, а згодом і до сімейства SPHINCS, в якому не потрібно підтримувати стан системи.

Таким чином, ми розглянули кілька постквантових алгоритмів для цифрового підпису. Область застосування кожного алгоритму визначається поставленим завданням. Тим не менш, із заяви аналітиків NIST випливає, що CRYSTAS-Dilithium, FALCON і SPHINCS+ використовуватимуться, коли потрібен цифровий підпис.

**Список використаних джерел**

1. Stinson D. R. Cryptography: theory and practice. Chapman and Hall/CRC, 2005.
2. Mermin N. Quantum computer science: an introduction. Cambridge University Press, 2007.
3. Butin Denis. "Hash-based signatures: State of play." IEEE security & privacy 15.4 (2017): 37-43.
4. McEliece R. J. "A public-key cryptosystem based on algebraic." Coding Thv 4244 (1978): 114-116.
5. Ding J., Petzoldt A. "Current state of multivariate cryptography." IEEE Security & Privacy 15.4 (2017): 28-36.
6. Ducas, Léo, et al. "Lattice signatures and bimodal Gaussians." Annual Cryptology Conference. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013
7. Jao David and Luca De Feo. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies." Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4. Springer Berlin Heidelberg, 2011

8. Fouque, P.A et al. Fast-Fourier Lattice-Based Compact Signatures over NTRU. URL: https://www.di.ens.fr/~prest/Publications/falcon.pdf.
9. Ducas, Léo, et al. "Crystals–dilithium: Digital signatures from module lattices. 2018." URL: https://eprint.iacr.org/2017/633.