МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ЕЛЕКТРОНІКИ, РОБОТОТЕХНІКИ І ТЕХНОЛОГІЙ МОНІТОРИНГУ ТА ІНТЕРНЕТУ РЕЧЕЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Шутко В.М.
«____»_____ 2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
ЗІ СПЕЦІАЛЬНОСТІ 171 «ЕЛЕКТРОНІКА»
ОПП «ЕЛЕКТРОННІ СИСТЕМИ»

Тема: «Потокове шифрування інформації на основі стохастичних матриць Уолша»

Виконавець
студент групи ЕС-413Б _____ Новіков Костянтин Андрійович

Керівник
д.т.н., професор _____ Білецький Анатолій Якович

Нормоконтролер _____ Сініцин Р.Б.

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки і телекомунікацій

Кафедра електроніки, робототехніки і технологій моніторингу та інтернету речей

Напрям (спеціальність) 171 «Електроніка»
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Шутко В.М.

«____»_____ 2021 р.

ЗАВДАННЯ

на виконання дипломної роботи

Новікову Костянтину Андрійовичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: «Потокове шифрування інформації на основі стохастичних матриць Уолша».

затверджена наказом ректора від «01» квітня2021р. №526/ст.

2. Термін виконання роботи : з «01» квітня 2021р. по «14» червня 2021р.

3. Вихідні дані до роботи: створення сучасного алгоритму та програмною реалізації потокового шифрування інформації на основі стохастичних матриць Уолша.

4. Зміст пояснювальної записки: теоретичний опис способу, розробка програмного рішення, результати аналізу та тестування.

5. Перелік обов’язкового ілюстративного матеріалу: Двійкова інверсія. Формування ЗЛКГ, ЗПКГ. Перестановка вхідних даних для різних систем Уолша. Стохастична перестановка ключем шифрування. Структурна схема ЗЛКГ, ЗПКГ. Взаємозв'язок номерів функцій в системах Уолша. Базова операція алгоритму ШПФ(метелик). Дерево швидкого перетворення Фур’є з проріджуванням по часу. Алгоритм шифрування 8-біт деревом ШПФ за різними системами Уолша(Адамара, Пелі, Качмажа, Кулі). Етапи стохастичного шифрування. Дерево ШПФ для 32 вхідних бітів.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Підпис керівника
1	Написати заяву кваліфікаційної роботи	12.03.2021	
2	Ознайомитися та обґрунтувати актуальність обраної теми	13.03.2021-01.0.02021	
3	Ознайомитися з загальними положеннями про дипломне проектування та оформлення	06.05.2021-07.05.2021	
4	Сформулювати мету і завдання кваліфікаційної роботи	08.05.2021-09.05.2021	
5	Здійснити бібліографічний пошук за темою кваліфікаційної роботи	10.05.2021	
6	Написати вступ та загальні відомості	11.05.2021-19.05.2021	
7	Ознайомитися з системи функцій Уолша. Написати 2-й розділ	20.05.2021-28.05.2021	
8	Описати принцип роботи алгоритму стохастичного шифрування	29.05.2021-01.06.2021	
9	Написати програмного забезпечення стохастичного шифрування	02.06.2021-11.06.2021	
10	Усунути недоліки. Дооформити роботу. Подати кваліфікаційну роботу на кафедру	12.06.2021	

7. Консультація з окремого(мих) розділу(ів):

Назва розділу	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Загальні відомості	д.т.н., професор Білецький А.Я.	11.05.2021	

8. Дата видачі завдання: «01» квітня 2021р.

Керівник дипломної роботи (проекту)_____Білецький А. Я.
(підпис керівника) П.І.Б.)

Завдання прийняв до виконання_____Новіков К. А.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Потокове шифрування інформації на основі стохастичних матриць Уолша»: 44 с., 31 рис., 9 табл., 5 літературних джерела.

Об’єкт дослідження: системи функцій Уолша і стохастичне шифрування інформації на основі матриць Уолша.

Мета роботи: дослідити різновиди систем функцій Уолша та створити стохастичний алгоритм шифрування інформації на їх основі.

Методи дослідження: абстрактно-логічний, порівняльний аналіз, проведення обрахунків, обробка літературних джерел.

Результати бакалаврської роботи рекомендується використовувати під час проведення лекцій, наукових досліджень та в практичній діяльності для шифрування інформації.

В першому розділі описано принцип формування різних систем функцій Уолша, основу шифрування по Грею, принцип формування зворотного ліво- і право-стороннього кодування по Грею.

У другому розділі описано принцип шифрування даних за допомогою швидкого перетворення Фур’є, перевірено алгоритм шифрування за допомогою швидкого перетворення Фур’є для різних систем функцій Уолша(Адамара, Пелі, Качмажа, Кулі), зашифровано 8-біт за допомогою алгоритму швидкого перетворення Фур’є для різних систем функцій Уолша(Адамара, Пелі, Качмажа, Кулі).

У третьому розділі розроблено стохастичний алгоритм шифрування даних за допомогою швидкого перетворення Фур’є – Уолша, на основі якого показано шифрування чотирьохбайтного блоку даних.

КРИПТОГРАФІЯ, ЗАХИСТ ІНФОРМАЦІЇ, ПОТОКОВЕ ШИФРУВАННЯ, СИСТЕМИ ФУНКЦІЙ УОЛША, СТОХАЧТИЧНІ МАТРИЦІ УОЛША,

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БПС - Безпілотне літальне судно;

ШПФ - Швидке перетворення Фур'є;

ДІП - Двійково-інверсна перестановок;

ДПФ - Дискретного перетворення Фур’є;

ДІП - Двійково-інверсна перестановка;

ЗЛКГ - Зворотне лівостороннє кодування по Грею;

ПЛКГ - Пряме лівостороннє кодування по Грею;

ЗПКГ - Зворотне правостороннє кодування по Грею;

ППКГ - Пряме правостороннє кодування по Грею.

ВСТУП

З поширенням технологій інтернет-речей і безпілотних повітряних суден постала проблема захищеності даних. Основне завдання БПС є розвідка наземної місцевості з підвищеним рівнем небезпеки, в яких отримання інформації, навіть авіарозвідкою, є ускладнене або неможливе. БПС надає можливість швидко, а саме головне, безпечно для людини дослідити важкодоступну місцевість, для подальшої обробки і аналізу інформації у випадках надзвичайних ситуацій. Оскільки будь-які розрахунки потребують часу і споживають енергію, для БПС, які живляться від акумуляторів, це вкрай критично.

Інформація, отримана БПС, повинна передаватися в режимі реального часу, для ефективної взаємодії з нею в пункті зв'язку, а саме обробкою й аналізу інформації. Повідомлення від БПС до пункту керування передається по каналах зв'язку, які в свою чергу повинні бути захищені він зовнішнього вторгнення. Ці атака спрямовані на перехоплення керування, виведення апарату з ладу, крадіжку розвідувальної інформації та завдання і навіть знаходження пункту зв'язку. Тому канали зв'язку повинні мати високий рівень захищеності від злому. Цим займається така наука як криптографія яка грає ключову роль як в теорії завадостійкого кодування, так і в інших спеціальних розділах криптографічного захисту інформації

Не дивлячись на новизну криптографії як науки, у неї ще є невирішені проблеми. На сьогодні фахівці виділяють кілька проблем в криптографії:

- час для шифрування та знаходження ключа;
- збільшення розміру зашифрованих блоків даних;
- ненадійність фундаменту шифрування;

Отже, існує потреба розробити шифри під потреби сучасних завдань захисту даних.

Мета даної роботи є збільшення ефективності криптографічного захисту інформація на основі потокового шифрування в основі якого використовуються стохастичні матриці Уолша. Застосування таких матриць дає можливість збільшити час знаходження ключа до нескінченності і збільшує захищеність шифру закодованої інформації.

Основне завдання роботи полягає в тому, щоб провести аналіз сучасних криптографічних методів захисту інформації за критеріями стійкості, швидкодії та необхідних ресурсів для розрахунків, а також розробити новий алгоритм шифрування який матиме складний ключ шифрування, покращену надійність зашифрованих даних і зможе швидко шифрувати і розшифровувати дані.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ

1.1 Формування Матриць

В формуванні систем функцій Уолша слугує функції Адамара. Системи функцій Адамара вперше були застосовані в математиці в кінці XIX-го століття. Після цього системи Адамара починають відігравати надзвичайно важливу роль в теорії та практиці цифрової обробки дискретних сигналів. Пізніше було помічено, що ці системи функцій можуть бути використані для побудови завадостійких кодів. І лише порівняно недавно системи функцій Адамара почали застосовувати в криптографії, стисненні інформації і в багатьох інших областях науки і технологій.
Дана система функцій виконує ортогональне, симетричне, лінійне перетворення над матрицею розміром 2^m з дійсних чисел.

Алгоритму побудови систем функцій Адамара є досить простим, в основі якого є відомий метод Пелі. Даний метод синтезу можна застосовувати лише для таких матриць, які кратні чотирьом порядків систем функцій Адамара, для яких величина $p = N - 1$ є простим числом. Прикладами позначених порядків є числа 12, 20, 24 і т.д., яким відповідають прості числа 11, 19, і 23 відповідно.
Метод Пелі заснований на обчисленні квадратичних лишків і квадратичних невирахувань і супроводжується рішенням найпростішого рівняння виду:

$$x^2 \equiv a \ (mod\ p), \tag{1.1}$$

де a і p - взаємно прості числа, а p - непарним простим числом.
Тобто, якщо порівняння $x^2 \equiv a \ (mod\ p)$, має рішення, то число a називається квадратичним лишком за модулем p. В іншому випадку, число a називається квадратичним невирахуванням. Таким чином ми можемо побудувати базову нормалізовану матрицю Адамара восьмого порядку, яка має вид [1]:

$$H_8 = \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & - & + & + & - & + & - & - \\ + & - & - & + & + & - & + & - \\ + & - & - & - & + & + & - & + \\ + & + & - & - & - & + & + & - \\ + & - & + & - & - & - & + & + \\ + & + & - & + & - & - & - & + \\ + & + & + & - & + & - & - & - \end{bmatrix}$$

1.2 Класична система функцій Уолша

Системи функцій Уолша є підмножиною систем Адамара. Характерна особливість функцій Уолша полягає в тому:

- функції Уолша визначені на двійково-раціональному інтервалі, з цього випливає, що порядок квадратних матриць систем Уолша є ступенем двійки і має вигляд: $N = 2^m$, де m - невід'ємне ціле число;
- матриці Уолша, які позначаються як W_N) є симетричними матрицям. ;
- матриця систем функції Уолша приймають лише два значення (+1 або -1), при чому їх кількість однакова в обох частинах функцій, і тому дані системи функцій зручні для цифрової обробки інформації;
- вони є дійсними функціями;
- дискретне перетворення в базисі функцій Уолша економне за обсягом обчислень і забезпечує на порядок більшу швидкість визначення.

Матриці Уолша різних порядків представлені нижче [1]:

$$W_1 = [1]$$

$$W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Можна зауважити, що наступні порядки матриць будуються з елементарних матриць меншого порядку. Тому матриця восьмого порядку W_8 буде складатися з матриць 4 порядку W_4 і матиме вигляд:

$$W_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Оскільки нумерація функцій Уолша може бути проведена різними способами, то можливі і різні системи функцій Уолша. Зручним методом запису таких систем є відображення їх у вигляді квадратних матриць, в яких кожен рядок - це функція Уолша, причому для простоти запису і більшої зрозумілості матриць, замість значень елементів цих функцій (+1 і -1) будемо записувати тільки їх знаки «+» або «-».

1.3 Система функції Уолша-Адамара

Історично першою, в розглянутому класі систем, з'явилась симетрична система функцій Адамара, яку прийнято називати системою Уолша, впорядкованої по Адамара, або просто - системою Уолша-Адамара. Позначимо цю матрицю як H_N, де N - розмір матриці. У цій системі, функції розташовуються одна під одній в такому порядку, що з них утворюється матриця Адамара. Систему функцій Адамара легко побудувати, спираючись на властивостях матриць Адамара. Матриця Адамара мінімального (N = 2) порядку має вигляд:

$$H_2 = \begin{bmatrix} + & + \\ + & - \end{bmatrix}$$

Для $N = 4$ матриця H_4 подібна матриці H_2, але її елементами будуть вже елементарні матриці H_2:

$$H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} \rightarrow \begin{bmatrix} + & + & + & + \\ + & - & + & - \\ + & + & - & - \\ + & - & - & + \end{bmatrix}$$

Аналогічно, для $N = 8$ матриця H_8, буде складатися з елементів матриці H_4.

$$H_8 = \begin{bmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{bmatrix} \rightarrow \begin{bmatrix} H_2 & H_2 & H_2 & H_2 \\ H_2 & -H_2 & H_2 & -H_2 \\ H_2 & H_2 & -H_2 & -H_2 \\ H_2 & -H_2 & -H_2 & H_2 \end{bmatrix} \rightarrow \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & - & + & - & + & - & + & - \\ + & + & - & - & + & + & - & - \\ + & - & - & + & + & - & - & + \\ + & + & + & + & - & - & - & - \\ + & - & + & - & - & + & - & + \\ + & + & - & - & - & - & + & + \\ + & - & - & + & - & + & + & - \end{bmatrix}$$

Зауважимо, що в матриця Адамара є симетричною, тобто не змінюється, якщо рядки і стовпці поміняти місцями.

1.4 Система функції Уолша-Пелі

Тепер розглянемо систему функцій Уолша, впорядковану за іншим принципом. Цю систему позначимо P_N і будемо називати системою функцій Уолша-Пелі. Функція системи Уолша-Пелі може бути отримана з системи Уолша-Адамара шляхом запису двійкового номера функції матриці H в зворотному порядку, тобто методом двійкової інверсії номерів функцій(табл. 1).

Таблиця 1

Двійкова інверсія

DEC	HEX	Двійкова інверсія	Двійково-інверсний DEC
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Матриця системи функцій Уолша-Пелі восьмого порядку така[1]:

$$P_N = \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & + & + & + & - & - & - & - \\ + & + & - & - & + & + & - & - \\ + & + & - & - & - & - & + & + \\ + & - & + & - & + & - & + & - \\ + & - & + & - & - & + & - & + \\ + & - & - & + & + & - & - & + \\ + & - & - & + & - & + & + & - \end{bmatrix}$$

1.5 Кодування по Грею

Коди Грея на зорі своєї появи, запропоновані в 1953 році у відповідь на запити інженерів, щодо побудови оптимальних за критерієм мінімальних помилок, привернули до себе увагу не тільки дослідників математиків, а й широкого кола розробників різноманітної електронної апаратури. Відмітна особливість кодів Грея полягає в тому, що в двійковій системі числення при переході від зображення одного числа до зображення сусіднього старшого або сусіднього молодшого числа відбувається зміна цифр (1 на 0 або навпаки) тільки в одному розряді числа . Такі коди відносять до групи двійкових кодів з одиничною відстанню Хеммінга. Код Грея не єдиний в цій групі, але його застосування в системах зв'язку, аналого-цифрових перетвореннях і в інших областях науки і техніки в силу ряду причин було найкращим.

Одна з особливостей даного дослідження виявилась поза увагою, як математиків, так і розробників електронної апаратури. Це можливість побудови кодів, протилежних по напрямку формування класичним КГ. В класичній схемі процес формування прямих і зворотних кодів розвивається у напрямку зліва направо. При цьому старший (лівий) розряд перетворюваного числа не змінюється як при прямому, так і зворотному перетвореннях. Разом з тим, можна побудувати схему перетворення зворотного по напрямку формування класичного (лівостороннього) перетворення Грея. У такому класі правосторонніх перетворень Грея при прямому і зворотному перетвореннях зберігається незмінне значення молодшого (правого) розряду перетворюваного числа.

Комбінація лівосторонніх і правосторонніх перетворень Грея (як прямих, так і зворотних) спільно з операцією інверсної перестановки кодів призвела до можливості побудови комбінованих або складових кодів Грея (СКГ) $\boldsymbol{G}$, утворених добутком сукупності простих кодів Грея $\boldsymbol{g}_i$. Вона має вигляд [3]:

$$G = \prod_{i=1}^{k} \boldsymbol{g}_i \qquad (1.2)$$

де k – довжина СКГ.

Застосування СКГ виявилося дуже успішним в задачах визначення структури та взаємозв'язку симетричних систем функцій Уолша, в криптографії, кодуванні та в інших напрямках.

1.6 Зворотне лівостороннє кодування по Грею

Алгоритм зворотного лівостороннього перетворенню по Грея вирішується звичайними алгебраїчними перетвореннями:

$$y_i = x_{i+1} \oplus_2 x_i,\ i = \overline{n-1,0},\ x_n = 0, \tag{1.3}$$

де $\oplus_2$ операція порозрядного додавання по модулю 2 (операція XOR).

Для двійкової системи числення правило перетворення вектора *x* в вектор *y* для ЗЛКГ досить просте і має вигляд:

$$\begin{aligned} x_3 &= y_3; \\ x_2 &= (y_2 - x_3)_2; \\ x_1 &= (y_1 - x_2)_2; \\ x_0 &= (y_0 - x_1)_2; \end{aligned} \tag{1.4}$$

В модульній арифметиці, для двійкової системи числення

$$(-1)_2 \equiv 1. \tag{1.5}$$

З урахуванням (1.5) отримуємо таку систему рівнянь:

$$\begin{aligned} x_3 &= y_3; \\ x_2 &= (y_2 + x_3)_2; \\ x_1 &= (y_1 + x_2)_2; \\ x_0 &= (y_0 + x_1)_2; \end{aligned} \tag{1.6}$$

Структурна схема формування зворотного лівостороннього кодування по Грею наведена на рис.1 [2].

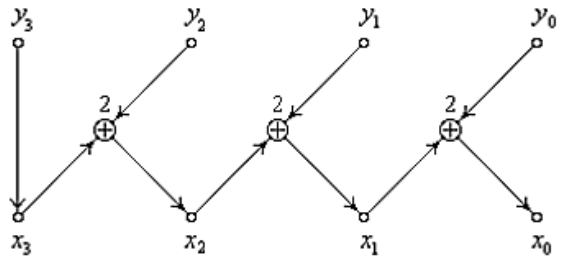

Рисунок 1 «Структурна схема зворотного лівостороннього кодування по Грею»

Можливий також алгоритм паралельного обчислення зворотного лівостороннього перетворення по Грея, який ми і будемо використовувати, він має вид:

$$x_3 = y_3;$$
$$x_2 = (-y_3 + y_2)_2;$$
$$x_1 = (y_3 - y_2 + y_1)_2; \quad (1.7)$$
$$x_0 = (-y_3 + y_2 - y_1 + y_0)_2;$$

Враховуючи (1.5) отримуємо:

$$x_3 = y_3;$$
$$x_2 = (y_3 + y_2)_2;$$
$$x_1 = (y_3 + y_2 + y_1)_2; \quad (1.8)$$
$$x_0 = (y_3 + y_2 + y_1 + y_0)_2;$$

З цього випливає, що для виконання зворотного лівостороннього перетворення по Грею в двійковій системі числення досить скласти двійковий код з самим собою, послідовно зсуваючи код на один розряд вправо. При цьому молодші розряди кодових комбінацій губляться, а додавання ведеться по модулю |2|(1+1 = 0), без урахування перенесення одиниці в наступний розряд. Як приклад розглянемо зворотне лівостороннє перетворення по Грею числа - 10110011.

Таблиця 2

Формування ЗЛКГ

	1	0	1	1	0	0	1	1	⋮								
		1	0	1	1	0	0	1	⋮	1							
			1	0	1	1	0	0	⋮	1	1						
				1	0	1	1	0	⋮	0	1	1					
$\oplus_2$					1	0	1	1	⋮	0	0	1	1				
						1	0	1	⋮	1	0	0	1	1			
							1	0	⋮	1	1	0	0	1	1		
								1	⋮	0	1	1	0	0	1	1	
	1	**1**	**0**	**1**	**1**	**1**	**0**	**1**	⋮	загублені розряди							

Виконавши обрахунки ЗЛКГ для всіх… отримуємо табл. 3 з перестановками …

Таблиця 3

Перестановка ЗЛКГ

DEC	BIN	ЗЛКГ
0	000	000
1	001	001
2	010	011
3	011	010
4	100	111
5	101	110
6	110	100
7	111	101

1.7 Система функції Уолша-Качмажа

У 1923 р Уолшем була запропонована повна система знакових функцій, впорядкованих за зростанням знакозмінних чисел на інтервалі N базисних функцій системи. Надалі така система отримала назву системи Уолша, впорядкованої за Качмажем або системи Уолша-Качмажа. Відмітна особливість систем Уолша-Качмажа полягає в тому, що вона виявилась дуже зручною для застосування в техніці зв'язку, так як функції зазначеної системи мають велику схожість зі звичними для інженерів гармонійними функціями.

Таким чином, як і матриця Пелі, матриця Качмажа утворюється в результаті певних організованих перестановок рядків матриці Адамара. Матриця цієї системи, яку позначимо через K_N, являється як і раніше симетричної і для має вигляд[1]:

$$K_N = \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & + & + & + & - & - & - & - \\ + & + & - & - & - & - & + & + \\ + & + & - & - & + & + & - & - \\ + & - & - & + & + & - & - & + \\ + & - & - & + & - & + & + & - \\ + & - & + & - & - & + & - & + \\ + & - & + & - & + & - & + & - \end{bmatrix}$$

Для того щоб матрицю Пелі P_N, N-го порядку трансформувати в матрицю Качмажа, досить переставити номера базисних функцій системи Пелі в двійковому поданні за законом зворотного лівостороннього кодуванню по Грею.

З цього випливає, для того щоб матрицю Уолша-Адамара H_N перетворити в матрицю Уолша-Качмажа K_N, спочатку слід від H_N перейти до P_N за допомогою операції двійкової інверсії номерів базисних функцій системи Адамара, а потім номера базисних функцій системи Пелі P_N трансформувати зворотнім лівостороннім кодуванням по Грею. В результаті різних перестановок і перетворень номерів базисних функцій ми можемо без проблем перейти з матриці Уолша-Адамара H_N в матрицю Уолша-Качмажа K_N і навпаки, по даній схемі рис.2 [3]:

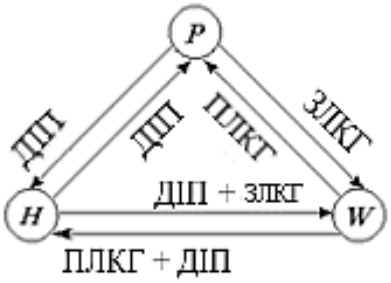

Рисунок 2 «Взаємозв'язок номерів функцій в системах Уолша»

1.8 Зворотне правостороннє кодування по Грею

Алгоритм зворотного правостороннього кодування по Грея вирішується звичайною систему рівнянь:

$$y_i = x_i \oplus_2 x_{i-1},\ i = \overline{0, n-1},\ x_{-1} = 0 \qquad (1.9)$$

Зокрема ЗПКГ має вигляд:

$$\begin{aligned} x_0 &= y_0; \\ x_1 &= (y_1 - x_0)_2; \\ x_2 &= (y_2 - x_1)_2; \\ x_3 &= (y_3 - x_2)_2; \end{aligned} \qquad (1.10)$$

На підставі тотожності (1.5) дану систему рівнянь можна записати так:

$$\begin{aligned} x_0 &= y_0; \\ x_1 &= (y_1 + x_0)_2; \\ x_2 &= (y_2 + x_1)_2; \\ x_3 &= (y_3 + x_2)_2; \end{aligned} \qquad (1.11)$$

Після чого легко скласти схему ЗПКГ яка наведена на рис.3[2].

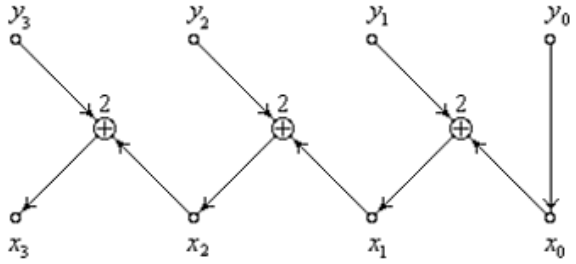

Рисунок 3 «Структурна схема зворотного правостороннього кодування по Грею»

Зі співвідношень (1.10) або безпосередньо зі схеми перетворення (рис. 11) випливає, що для виконання ЗПКГ досить скласти код з копіями цього ж коду, послідовно зсуваючи на один розряд вліво. При цьому старші розряди кодових комбінацій губляться, а додавання ведеться по модулю |2|(1+1 = 0), без урахування перенесення одиниці в наступний розряд. Як приклад розглянемо зворотне правостороннє кодування по Грею числа - 10110011.

Таблиця 4

Формування ЗПКГ

							⋮	1	0	1	1	0	0	1	1	
						1	⋮	0	1	1	0	0	1	1		
					1	0	⋮	1	1	0	0	1	1			
				1	0	1	⋮	1	0	0	1	1				$\oplus_2$
			1	0	1	1	⋮	0	0	1	1					
		1	0	1	1	0	⋮	0	1	1						
	1	0	1	1	0	0	⋮	1	1							
1	0	1	1	0	0	1	⋮	1								
загублені розряди								1	0	0	1	0	0	0	1	

Схема формування ЗПКГ (рис. 11) може бути отримана зі структурної схеми ЗЛКГ(рис. 10). З цією метою досить повернути рис. 10 відносно центральної вертикальної осі на 180°. Операнди перетворення *x* і *y* при цьому повинні залишатися на тих же самих місцях, які вони займали до розвороту.

Виконавши обрахунки ЗЛКГ для всіх… отримуємо табл. 3 з перестановками …

Таблиця 5

Перестановка ЗПКГ

DEC	BIN	ЗПКГ
0	000	000
1	001	100
2	010	110
3	011	010
4	100	111
5	101	011
6	110	001
7	111	101

1.9 Система функції Уолша-Кулі

Звернемо увагу на те, що системи функцій Н, Р, К пов'язані між собою зворотнім і прямим лівостороннім перетворенням по Грею, а також двійковою інверсією номерів функцій. В основі системи функцій Уолша-Кулі є впорядкована перестановка номерів базисних функцій матриць Уолша-Пелі за допомогою зворотного правостороннього перетворення по Грея. В результаті перестановки рядків матриць Пелі восьмого порядку маємо матрицю Уолша-Кулі, яку позначимо C_N [1].

$$C_N = \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & + & - & - & - & - & + & + \\ + & - & - & + & + & - & - & + \\ + & - & + & - & - & + & - & + \\ + & - & + & - & + & - & + & - \\ + & - & - & + & - & + & + & - \\ + & + & - & - & + & + & - & - \\ + & + & + & + & - & - & - & - \end{bmatrix}$$

Одна з проблем синтезу симетричних систем функцій Уолша полягає в тому, що за весь час дослідження даних систем не було запропоновано жодної нової системи, крім трьох вище сказаних класичних, симетричних систем Уолша, упорядкованих по Адамара Н, Качмажу К, Пелі Р. Всі системи Уолша N-го порядку складаються з одного і того ж повного набору базисних функцій і відрізняються

один від одного лише способами їх впорядкування, які, власне, і призводять до симетричних форм систем Уолша. Проблеми, пов'язані з пошуком методів впорядкування симетричних системи Уолша, виявилися дозволеними тільки після того, як було введено правостороннє кодування по Грею. Але це не дає великої надійності в шифруванні даних.

РОЗДІЛ 2
ПРИНЦИП ШИФРУВАННЯ

2.1 Фундамент шифрування

Основа шифрування даних лежить на дереві швидкого перетворення Фур'є з проріджуванням по часу, в тій чи іншій системі Уолша. Основною особливість ШПФ є алгоритм швидкого шифрування і розшифровування інформації. Суть цього алгоритму полягає в багаторазовому членуванні заданої послідовності відліків вхідних сигналів(даних) на більш короткі послідовності.

Нехай задано послідовність відліків:

$$x(n), n = 0, 1, \dots, N - 1, \tag{2.1}$$

де $N = 2^m$, m – ціле число.

Введемо дві (*N/2*) - точкові послідовності $x_1(n)$ *і* $x_2(n)$, відповідно з парними і не парними членами *x(n):*

$$x_1(n) = x(2n),\ n = 0, 1, \dots, \left(\frac{N}{2}\right) - 1; \tag{2.2}$$

$$x_2(n) = x(2n + 1),\ n = 0, 1, \dots, \left(\frac{N}{2}\right) - 1; \tag{2.3}$$

На рис. 2 наведено послідовність для $N = 8$.

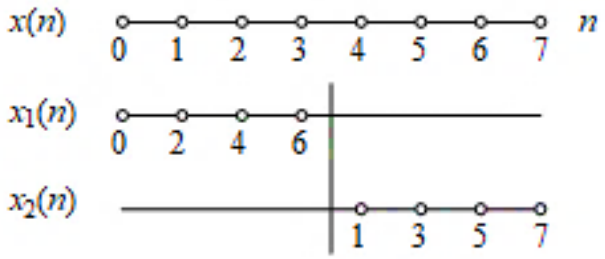

Рисунок 4 «Схематичне зображення послідовностей відліків сигналу»

Виходячи з ДПФ, ці послідовності *x(n)* можна записати так:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n)\, W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\, W_N^{(2n+1)k} \tag{2.4}$$

З урахуванням того, що

$$W_N = e^{-j\frac{2\pi}{N}} \to W_N^2 = \left(e^{-j\frac{2\pi}{N}}\right) = e^{-j\frac{2\pi}{N/2}} = W_{N/2} \tag{2.5}$$

перепишемо вираз (2.5) у вигляді:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x_1(n)\, W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x_2(n)\, W_{N/2}^{nk} \tag{2.6}$$

або більш компактно:

$$X(k) = X_1(k) + W_N^k X_2(k), \tag{2.7}$$

де $X_1(k)$ і $X_2(k)$, рівні (*N/2*) - точкові послідовності $x_1(n)$ і $x_2(n)$.

Таким чином N – точкове ДПФ *X(k)* можна розкласти на дві (*N/2*) - точкові послідовності, результати яких об'єднуються відповідно до співвідношення (2.6). Оскільки *X(k)* визначено на інтервалі $0 \le k \le N-1$, а $X_1(k)$ і $X_2(k)$, задано для значень $0 \le k \le (N/2)-1$, то необхідно довизначити формулу (2.6) для $k \ge N/2$

Вона має вигляд:

$$X_1(k) = X_1\left(k - \frac{N}{2}\right); \tag{2.8}$$

$$X_2(k) = X_2\left(k - \frac{N}{2}\right), k = \frac{N}{2}, \frac{N}{2} + 1, \dots n - 1 \tag{2.9}$$

Зауважимо, що для $k \ge N/2$

$$W_N^k = -W_N^{k-\frac{N}{2}}, \frac{N}{2} \le k \le N-1. \tag{2.10}$$

На підставі співвідношень (2.7), (2.10) вираз (2.8), (2.9) можна подати у вигляді системи:

$$X(k) = \begin{cases} X_1(k) + W_N^k X_2(k), & 0 \le k \le \left(\frac{N}{2} - 1\right) \\ X_1\left(k - \frac{N}{2}\right) - W_N^{k-\frac{N}{2}} X_2\left(k - \frac{N}{2}\right), & \frac{N}{2} \le k \le (N-1) \end{cases} \tag{2.11}$$

Дану систему рівнянь зручно зображувати у вигляді, так званої операції «метелика», яка широко використовується в алгоритмах ШПФ рис. 3 [4].

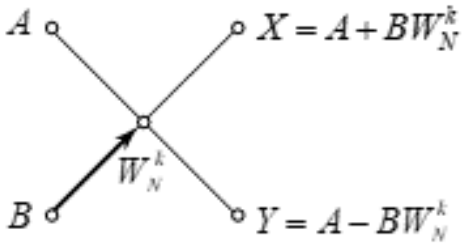

Рисунок 5 «Базова операція алгоритму ШПФ(метелик)»

Верхній вихід відповідає сумі, а нижній - різниці змінних A і B, при чому змінна B береться з вагою W_N^k. З цього випливає, що все шифрування даних зводиться до простих операцій додавання і віднімання комплексних чисел.

На рис. 3 за допомогою базової операції ШПФ «метелик», наведено послідовність операцій при виконанні восьмиточкового ДПФ . Вхідну послідовність спочатку розбиваємо на дві послідовності $x_1(n)$ і $x_2(n)$. і (див. рис. 2) з парних і непарних членів *x(n)* після чого розраховуємо їх перетворення $X_1(k)$ і $X_2(k)$. Потім відповідно до формули (2.10) і базової операцій ШПФ, поданої на рис. 2.2, дістаємо *X(k)* для *k = 0, 1, ... N - 2* .

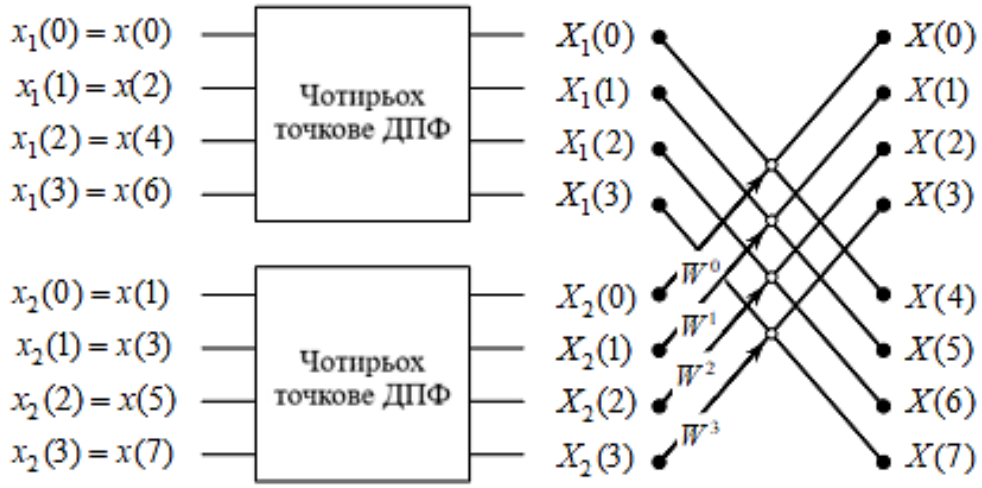

Рисунок 6 «Обчислення восьмиточкового ДПФ через два чотирьохточкові ДПФ»

Розглянуту схему обчислень можна використати для розрахунку *N/4*-точкового ДПФ, які розраховуються по аналогії *N/2* – точкового ДПФ і має вигляд для *X(k)*,

де $0 \le k \le (N/4) - 1$; *A(k)* і *B(k)* - *N/4* точкові ДПФ відповідно парних і непарних членів *x(n)*. Оскільки:

$$W_{N/2}^{k} = e^{-j\frac{2\pi}{N/2}k} = e^{-j\frac{2\pi}{N}2k} = W_N^{2k}, \qquad (2.12)$$

де *k* змінюється від 0 до *(N/4) - 1*.

$$X(k) = \begin{cases} X_1(k) + W_N^{2k} X_2(k), & 0 \le k \le \left(\frac{N}{4} - 1\right) \\ X_1\left(2k - \frac{N}{4}\right) - W_N^{2k-\frac{N}{4}} X_2\left(k2 - \frac{N}{4}\right), & \frac{N}{4} \le k \le \left(\frac{N}{2} - 1\right) \end{cases} \qquad (2.13)$$

На рис. 7 і рис. 8 зображено граф обчислення чотирьохточкового ДПФ для послідовності відліків $x_1(n)$ і $x_2(n)$.

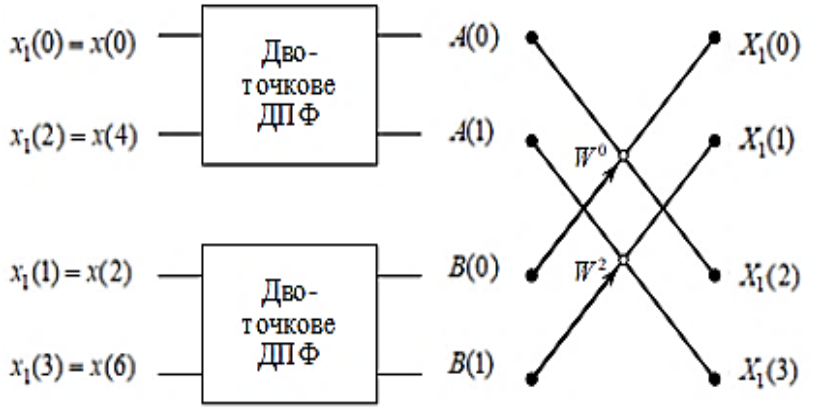

Рисунок 7 «Обчислення чотириточкового ДПФ для послідовності $x_1(n)$ через два двоточкових ДПФ»

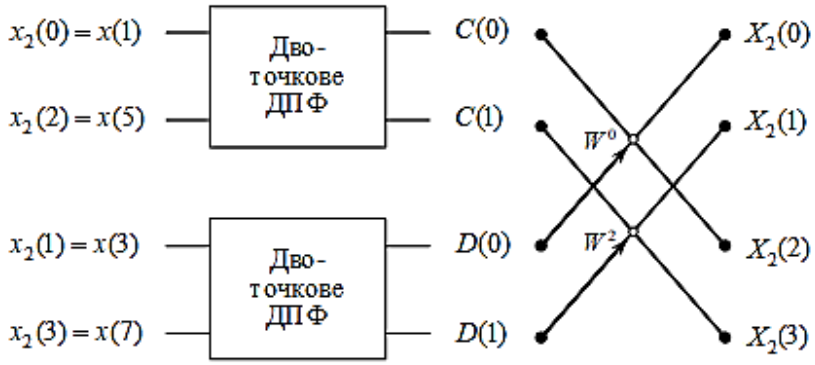

Рисунок 8 «Обчислення чотириточкового ДПФ для послідовності $x_2(n)$ через два двоточкових ДПФ»

Процес зменшення розміру ДПФ від *N* до *N*/2, де *N* дорівнює степені 2, можна продовжити доти, доки не залишаться тільки двоточкове ДПФ (як це зображено на рис. 9 і 10 для *N* = 8). Оскільки в двоточковому ДПФ граф перетворення базується на операції «метелик» (див. рис 3) із вагою ребра, що дорівнює *W* у нульовому степені ($W^0 = 1$), то обчислення зводяться до простих арифметичних операцій додавання і віднімання без використання множення.

Таким чином, восьмиточкове ДПФ у підсумку зводиться до алгоритму який має 3 ступені(етапи) обрахунків, показане на рис. 9 [5].

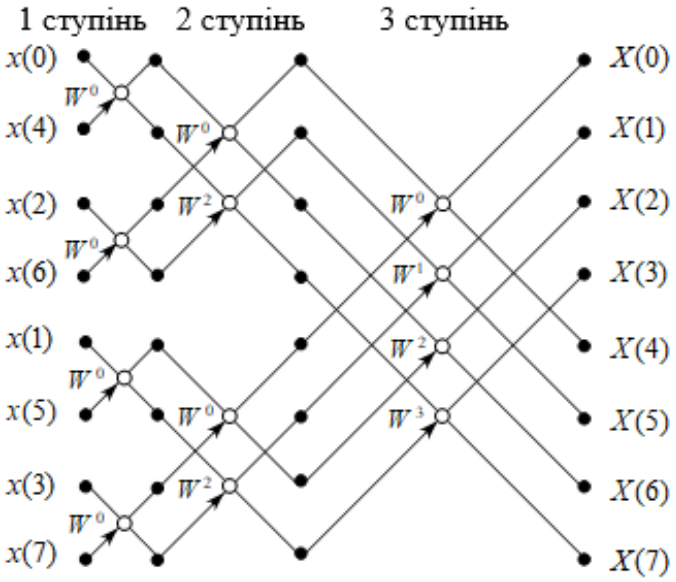

Рисунок 9 «Восьмиточкове ШПФ з проріджуванням у два рази»

Так, наприклад, на першому етапі ШПФ виконуються тільки операції додавання і віднімання комплексних чисел(вхідних даних). На другому етапі також використовуються тільки операції додавання і віднімання значень, які були отримані на попередньому етапі, так само і на третьому етапі. Це значно зменшує час шифрування, завдяки простим арифметичним обрахункам, які апаратно обраховуються швидше, ніж інші арифметичні операції, а за допомогою ШПФ ускладняється шифрування інформації. На рис. 10 за допомогою базових операцій «метелика» показане більш зрозуміле дерево ШПФ 8 порядку.

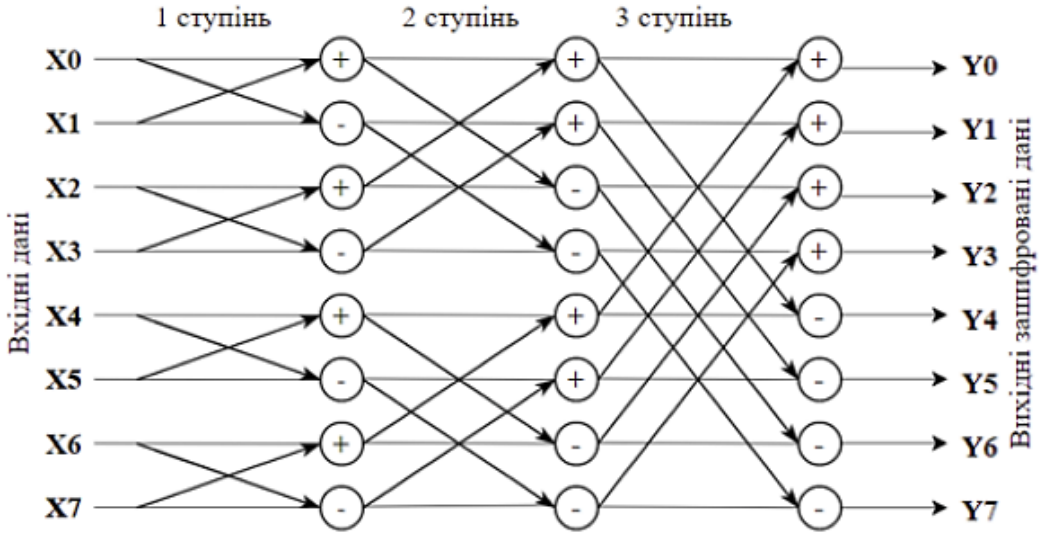

Рисунок 10 «Дерево швидкого перетворення Фур'є з проріджуванням по часу»

Аналіз рис. 8 і процедури послідовного скорочення розмірів перетворень показує, що найбільший виграш виходить для послідовності відліків вхідних сигналів для $N = 2^к$, так як в цьому випадку процес розбиття на дві послідовності вдається довести до 2-х точкового перетворення Фур'є. При цьому на кожному етапі об'єднання двох алгоритмів ШПФ меншого порядку потрібно *N/2* операцій множення. Загальна кількість операцій комплексного множення для обчислення ШПФ вираховується за формулою:

$$N_{ОП} = \frac{N}{2} \cdot log_2 N \qquad (2.14)$$

Як приклад розглянемо ШПФ послідовності з $N = 1024$ відліків вхідних сигналів. Для ДПФ нам би знадобилося N^2 операцій множення. Це приблизно 1 мільйон операцій. При ШПФ нам буде потрібно $512 \cdot 10 = 5120$ операцій комплексного множення. Це в 200 раз менше чим при ДПФ і в декілька десятків разів швидше.

При шифруванні за допомогою ШПФ дерево перетворення (рис. 7) залишається незмінним при будь-яких перетворень систем Уолша, але послідовність вхідних сигналів розподіляються по входах процесора за законом, відповідно обраному перетворенню: Адамара, Пелі, Качмажа, Кулі.

2.2 Шифрування за Уолшом-Адамара

Перевіримо алгоритм шифрування інформації деревом ШПФ і зашифруємо дані за Уолшом-Адамаром. Так як послідовність відліків вхідних даних(бітів) записується на вхід процесора одна під одній в такому порядку яка вона є. На рис.11 ми можемо побачити 4-х входове дерево ШПФ для Уолша-Адамара.

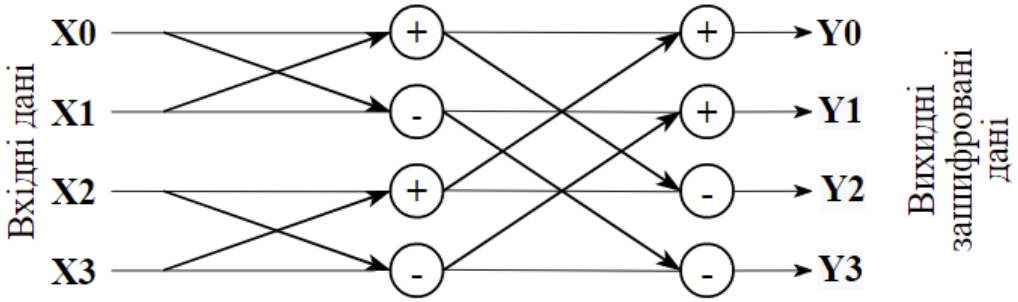

Рисунок 11 «Чорьохвхдове дерево ШПФ для Уолша-Адамара»

Для прикладу візьмемо вхідні дані у вигляді: a, b, c, d, e ,f, g, h.

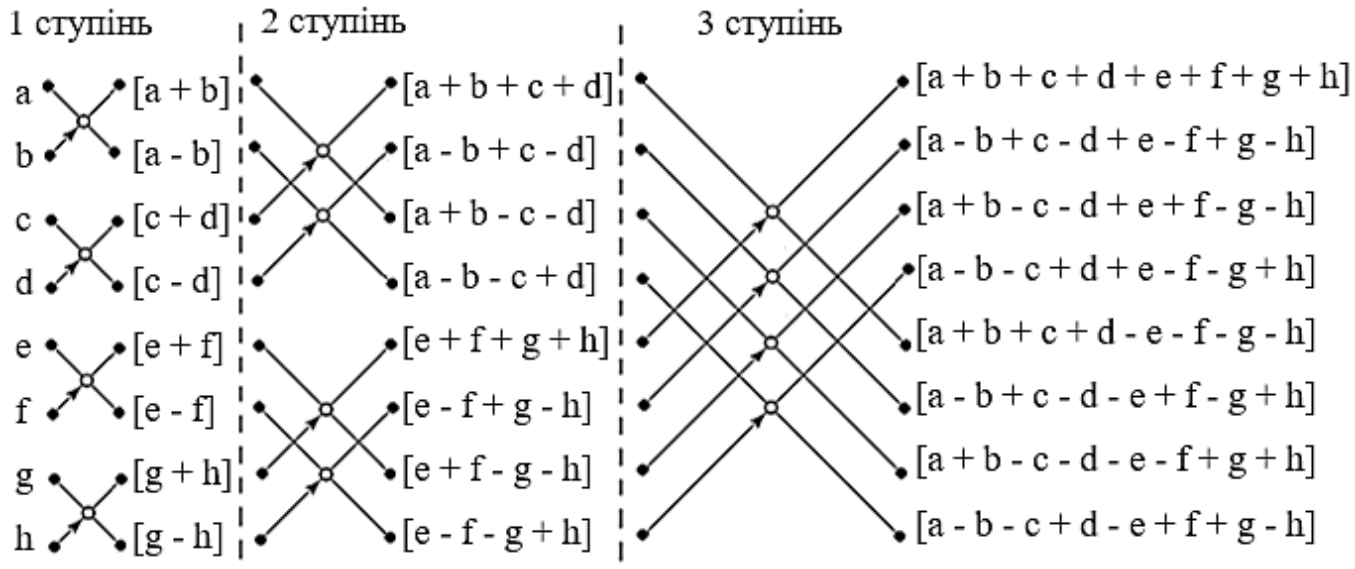

Рисунок 12 «Перевірка алгоритму шифрування деревом ШПФ за Уолшом-Адамаром»

Прибравши всі зашифровані дані (a, b, c, d, e ,f, g, h), отримуємо матрицю з «+» і «-» зашифровану деревом ШПФ з проріджуванням по часу, яка співпадає з матрицею Уолша-Адамара (H_8). Це означає коректну роботу алгоритму і ми можемо приступати до шифрування і розшифрування даних. Для прикладу візьмемо вхідні 8 біт – 1 0 0 1 1 0 1 0.

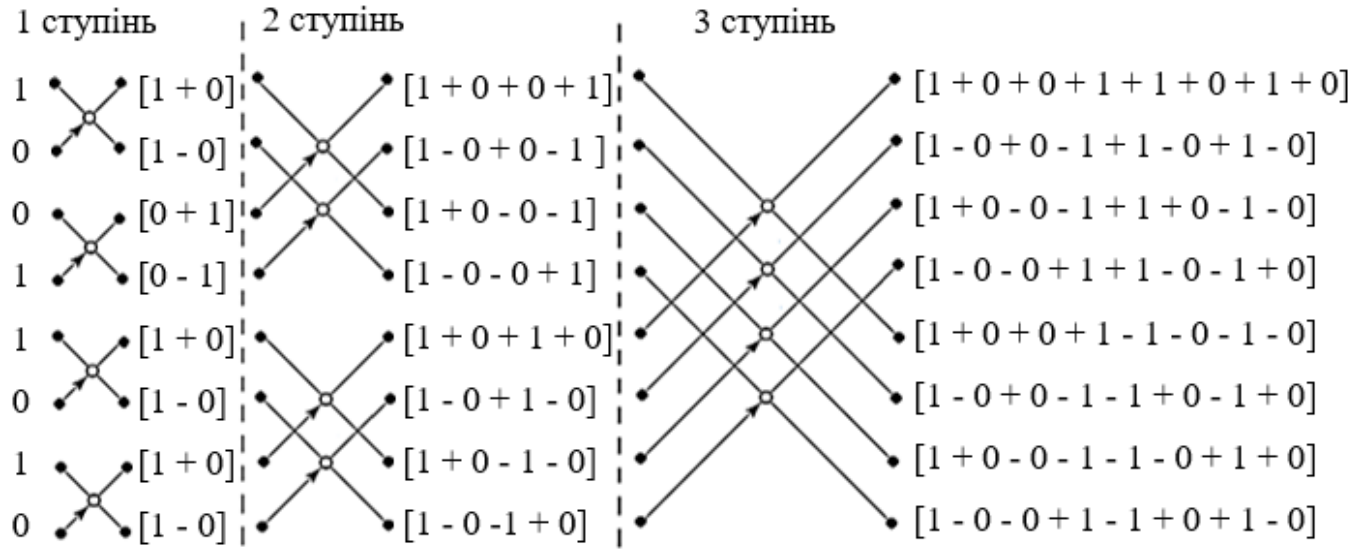

Рисунок 13 «Алгоритм шифрування 8-біт деревом ШПФ за Уолшом-Адамаром»

Виконавши обрахунки після 3-го ступеня шифрування на виході отримуємо зашифровані 8 біт – 4 2 0 2 0 -2 0 2. Тепер розшифруємо дані тим самим алгоритмом, а саме деревом ШПФ(див. рис. 7).

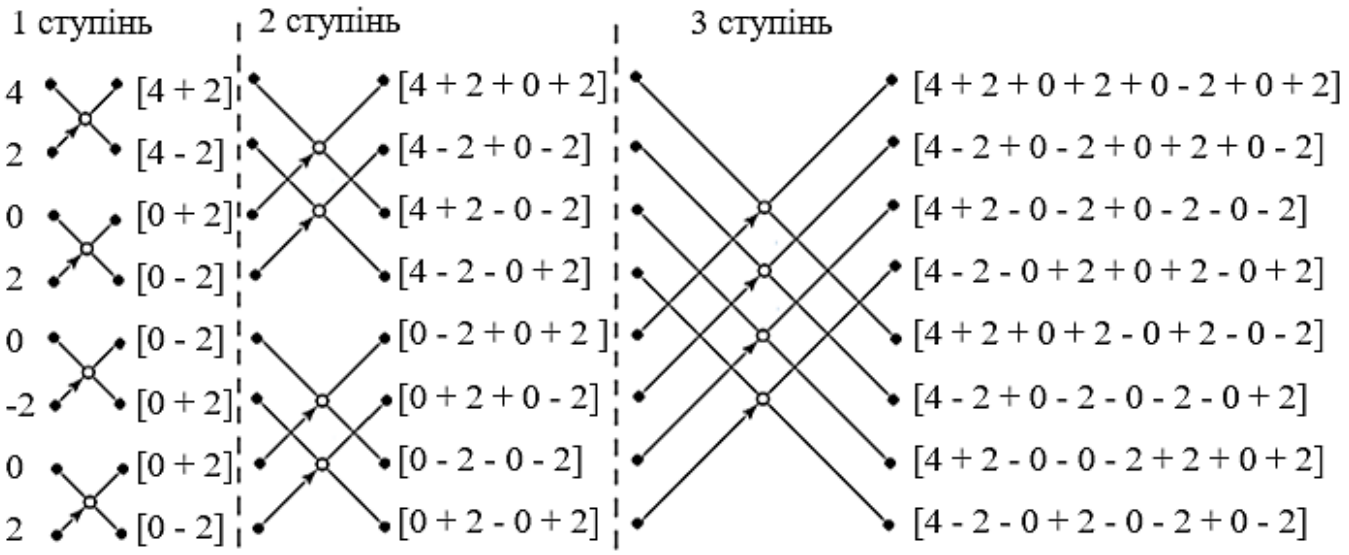

Рисунок 14 «Алгоритм розшифрування 8-біт деревом ШПФ за Уолшом-Адамаром»

Виконавши обрахунки після 3-го ступені розшифрування на виході отримуємо – 8 0 0 8 8 0 8 0. Після ділення розшифрованих даних - 8 0 0 8 8 0 8 0 на $N = 8$, отримуємо вхідні 8 біт – 1 0 0 1 1 0 1 0.

2.3 Шифрування за Уолшом-Пелі

При шифруванні за Уолшем-Пелі, послідовність віділків вхідних даних(бітів) які шифруються, піддаються організованій перестановці, а саме двійковій інверсії, що в свою чергу, вже зашифровує дані на вході процесора шифрування ШПФ. Для прикладу візьмемо вхідні біти у вигляді - a, b, c, d, e, f, g, h. Переставивши ці вхідні біти двійковою інверсією(див. табл. 1) отримуємо на вході процесора шифрування ШПФ таку послідовність вхідних даних(табл. 6).

Таблиця 6

Перестановка вхідних даних для Уолша-Пелі

a	b	c	d	e	f	g	h
↓							
двійкова інверсія							
↓							
a	e	c	g	b	f	d	h

Виконавши шифрування 8-ми входовим деревом ШПФ отримуємо:

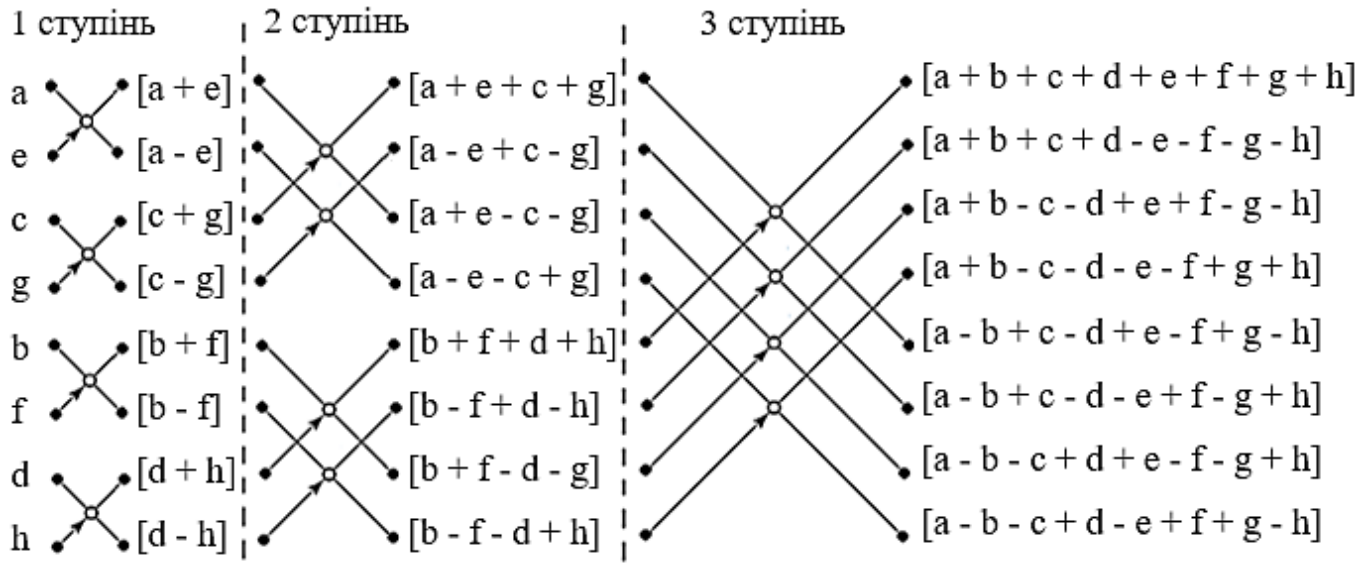

Рисунок 15 «Перевірка алгоритму шифрування деревом ШПФ за Уолшом-Пелі»

Знову перевіримо алгоритм на коректну роботу, прибравши зашифровані дані (a, b, c, d, e ,f, g, h) отримуємо матрицю з «+» і «-» зашифровану деревом ШПФ з проріджуванням по часу, яка співпадає з матрицею Уолша-Пелі (P_8). Це означає коректну роботу алгоритму і ми можемо приступати до шифрування і розшифрування даних. Візьмемо вхідні 8 біт – 10011010, не забуваючи здійснити двійкову інверсію перед входом в процесор шифрування ШПФ. На вході процесора отримуємо на 8 біт – 1 1 0 1 0 0 1 0.

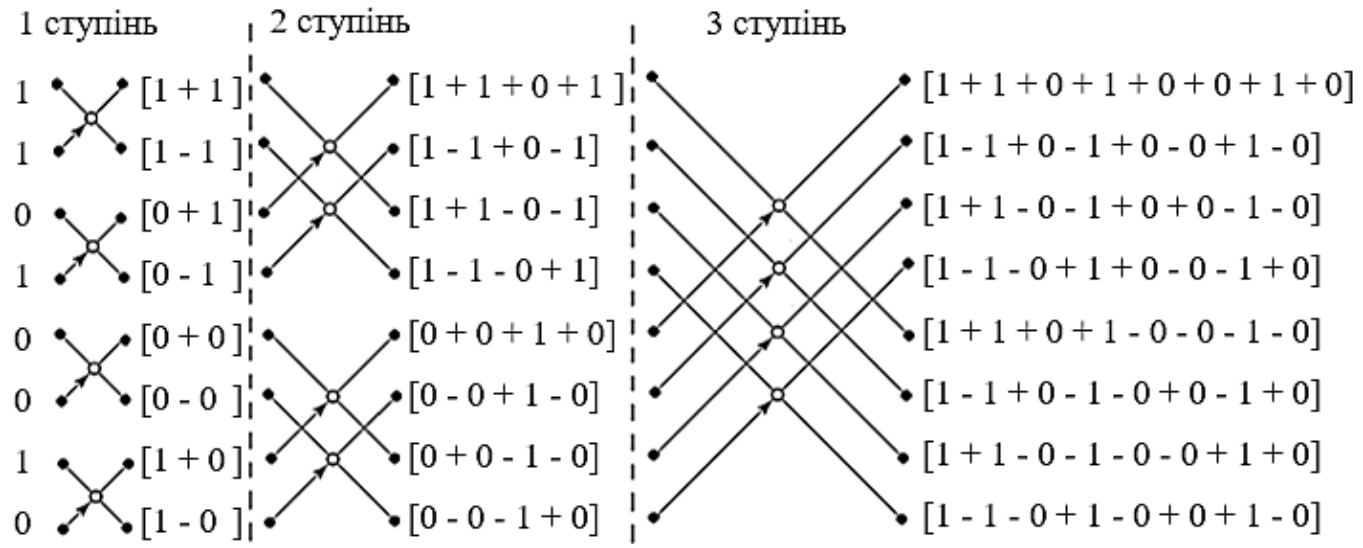

Рисунок 16 «Алгоритм шифрування 8-біт деревом ШПФ за Уолшом-Пелі»

Виконавши обрахунки після 3-го ступені шифрування на виході отримуємо зашифровані 8 біт – 4 0 0 0 2 -2 2 2. Тепер розшифруємо дані тим самим алгоритмом, а саме деревом ШПФ(див. рис. 7).

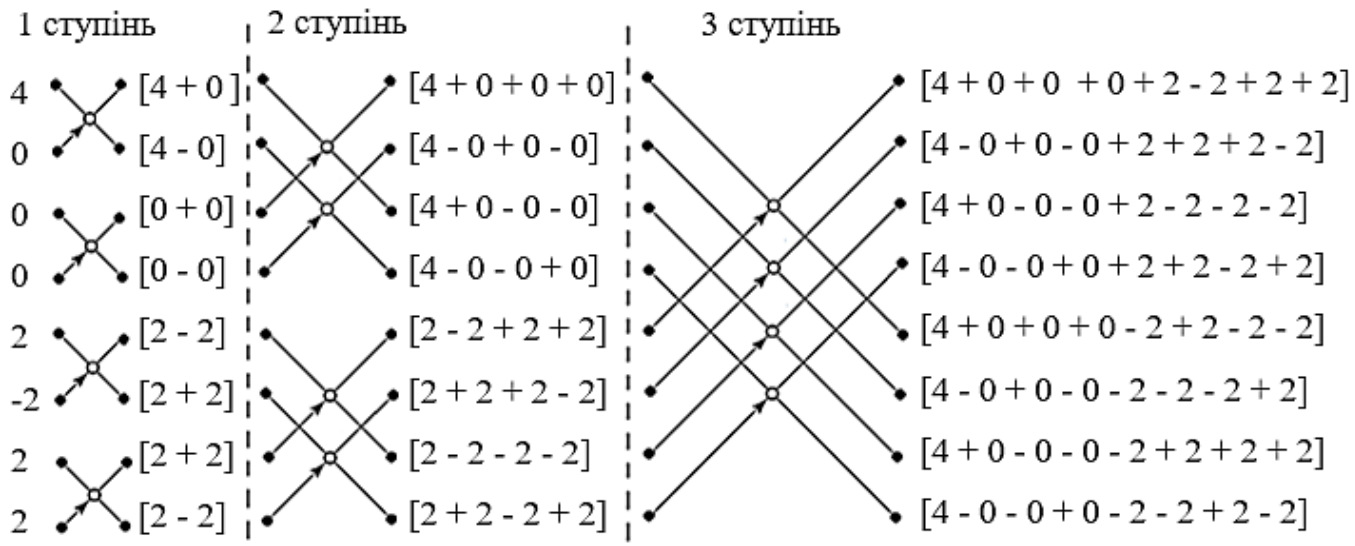

Рисунок 17 «Алгоритм розшифрування 8-біт деревом ШПФ за Уолшом-Пелі»

Виконавши обрахунки після 3-го ступені розшифрування на виході отримуємо – 8 8 0 8 0 0 8 0. Після ділення розшифрованих даних - 8 8 0 8 0 0 8 0 на $N = 8$, отримуємо вхідні 8 біт – 1 1 0 1 0 0 1 0. Якщо переставити ці дані за допомогою організованої перестановок - зворотної двійкової інверсії отримаємо початкові вхідні дані - 1 0 0 1 1 0 1 0 Уолша-Адамара.

2.4 Шифрування за Уолшом-Качмажа

Для шифрування даних за Уолшом-Качмажем, потрібно трансформувати вхідні біти - a, b, c, d, e, f, g, h організованою перестановкою, а саме двійковою інверсією і законом зворотного лівостороннього кодуванню по Грею. Враховуючи інформування в попередніх підрозділах, переставляємо ці вхідні біти двійковою інверсією(див. табл. 1) і ЗЛКГ отримуємо на вході процесора шифрування ШПФ таку послідовність вхідних даних зображених в табл. 7.

Таблиця 7

Перестановка вхідних даних для Уолша Качмажа

a	b	c	d	e	f	g	h
↓							
Двійкова інверсія							
↓							
a	e	c	g	b	f	d	h
↓							
ЗЛКГ							
↓							
a	h	d	e	b	g	c	f

Виконавши шифрування 8-ми входовим деревом ШПФ отримуємо:

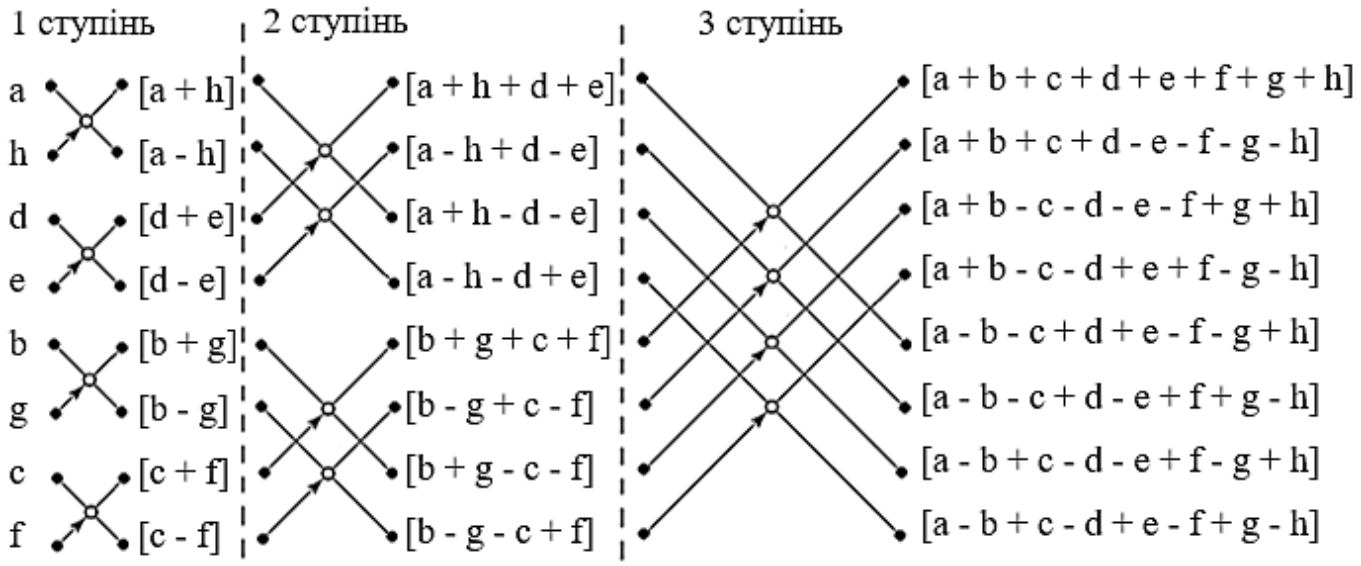

Рисунок 18 «Перевірка алгоритму шифрування деревом ШПФ за Уолшом-Качмажем»

Перевіримо алгоритм на коректну роботу, прибравши зашифровані дані (a, b, c, d, e ,f, g, h) отримуємо матрицю з «+» і «-» зашифровану деревом ШПФ з проріджуванням по часу, яка співпадає з матрицею Уолша-Качмажа (K_8). Це означає коректну роботу алгоритму і ми можемо приступати до шифрування і розшифрування даних. Для прикладу візьмемо все ті ж вхідні 8 біт – 1 0 0 1 1 0 1 0, не забуваючи зробити двійкову інверсією і ЗЛКГ. Так ми отримаємо на вході процесора шифрування ШПФ 8 біт – 1 0 1 1 0 1 0 0.

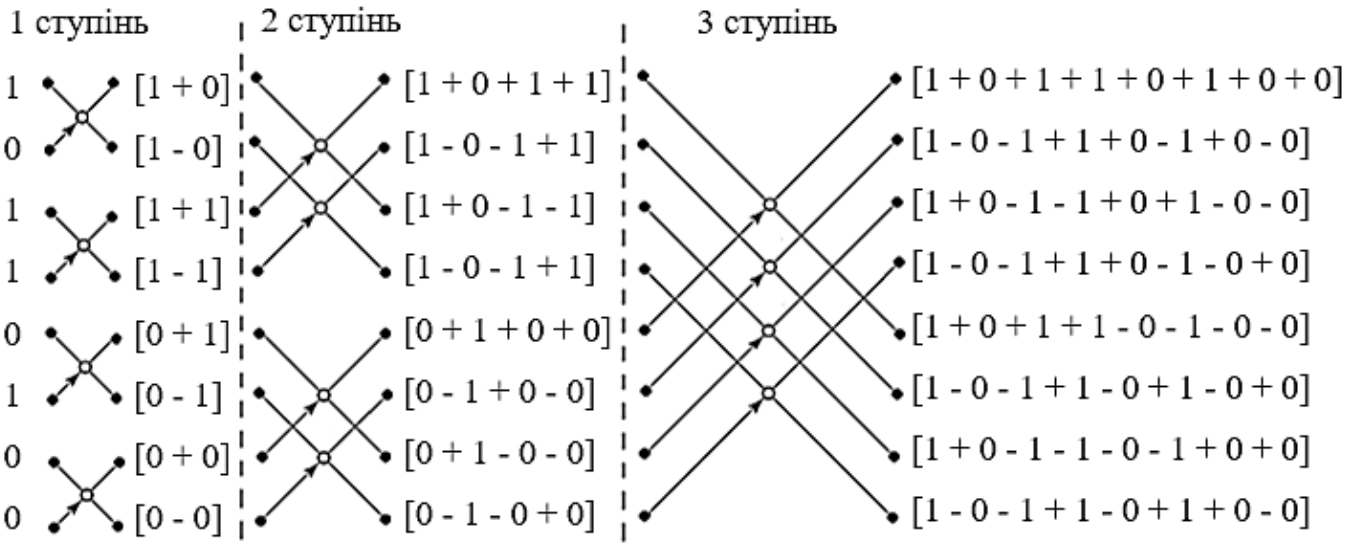

Рисунок 19 «Алгоритм шифрування 8-біт деревом ШПФ за Уолшом-Качмажем»

Виконавши обрахунки після 3-го ступеня шифрування, отримуємо зашифровані 8 біт – 4 0 0 0 2 2 -2 2. Тепер розшифруємо дані тим самим алгоритмом, а саме деревом ШПФ(див. рис. 7).

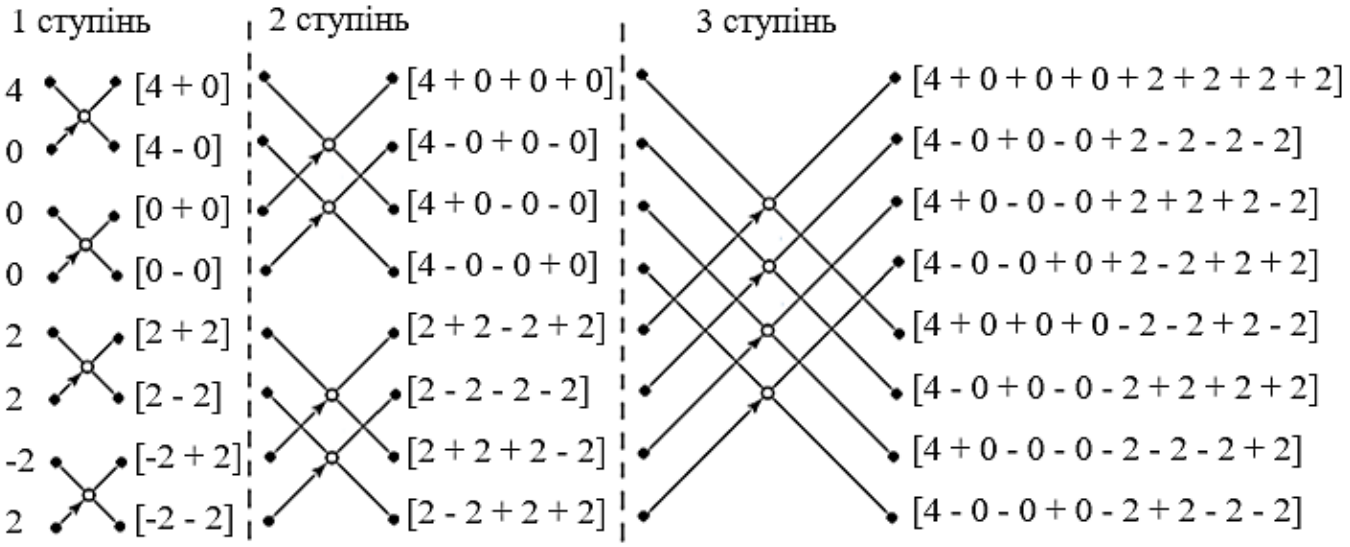

Рисунок 20 «Алгоритм розшифрування 8-біт деревом ШПФ за Уолшом-Качмажем»

Виконавши обрахунки після 3-го ступені розшифрування, отримуємо – 8 0 8 8 0 8 0 0. Після ділення розшифрованих даних - 8 0 8 8 0 8 0 0 на N = 8, отримуємо наші вхідні 8 біт – 1 0 1 1 0 1 0 0. Якщо ці дані переставити за допомогою ПЛКГ отримаємо вхідні дані Уолша-Пелі - 1 1 0 1 0 0 1 0 і якщо ці біти переставити двійковою інверсією, то отримаємо початкові вхідні дані - 1 0 0 1 1 0 1 0 - Уолша-Адамара.

2.5 Шифрування за Уолшом-Кулі

Для шифрування даних за Уолшом-Кулі, потрібно трансформувати вхідні біти - a, b, c, d, e, f, g, h організованою перестановкою, а саме двійковою інверсією і законом зворотного правостороннього кодуванню по Грею. Враховуючи інформування в попередніх підрозділах, переставляємо ці вхідні біти двійковою інверсією(див. табл. 1) і ЗПКГ отримуємо на вході процесора шифрування ШПФ таку послідовність вхідних даних зображених в табл. 8.

Таблиця 8

Перетворення вхідних даних для Уолша-Кулі

a	b	c	d	e	f	g	h
↓ Двійкова інверсія ↓							
a	e	c	g	b	f	d	h
↓ ЗПКГ ↓							
a	e	g	c	h	d	b	f

Виконавши шифрування 8-ми входовим деревом ШПФ отримуємо:

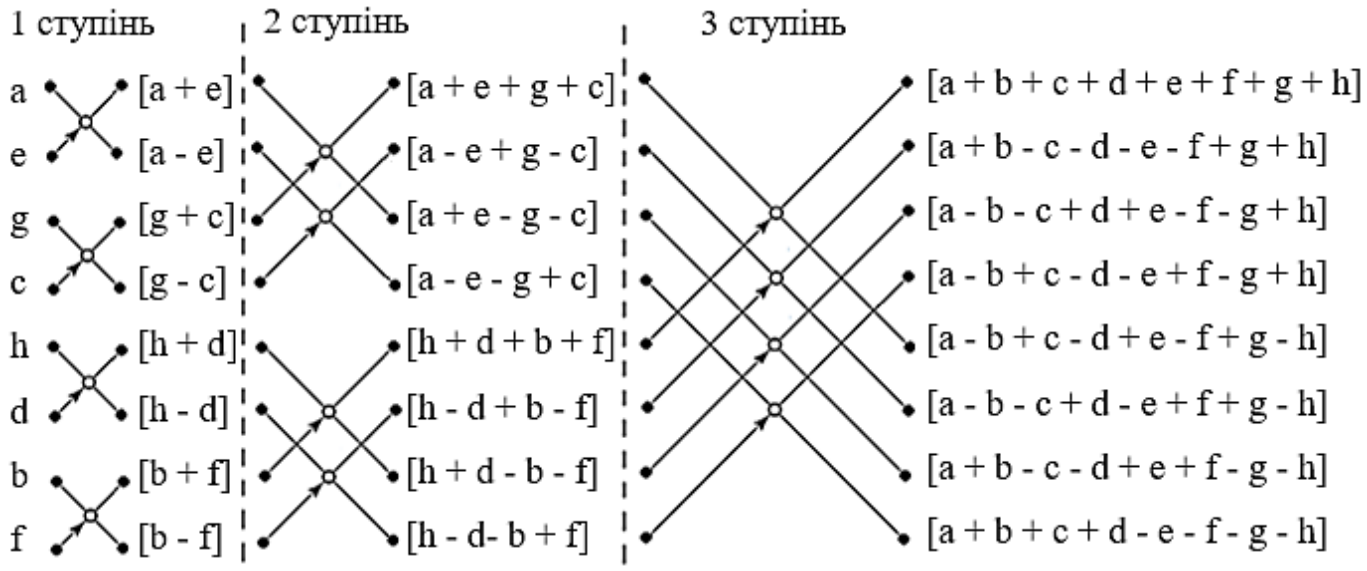

Рисунок 21 «Перевірка алгоритму шифрування деревом ШПФ за Уолшом-Кулі»

Перевіримо алгоритм на коректну роботу, прибравши зашифровані дані (a, b, c, d, e ,f, g, h) отримуємо матрицю з «+» і «-» зашифровану деревом ШПФ з проріджуванням по часу, яка співпадає з матрицею Уолша-Кулі (C_8). Це означає коректну роботу алгоритму і ми можемо приступати до шифрування і розшифрування даних. Для прикладу візьмемо все ті ж вхідні 8 біт – 1 0 0 1 1 0 1 0, не забуваючи виконати двійкову інверсією і ЗПКГ. Так на вході процесора ми отримаємо вхідні 8 біт для шифрування – 1 1 1 0 0 1 0 0.

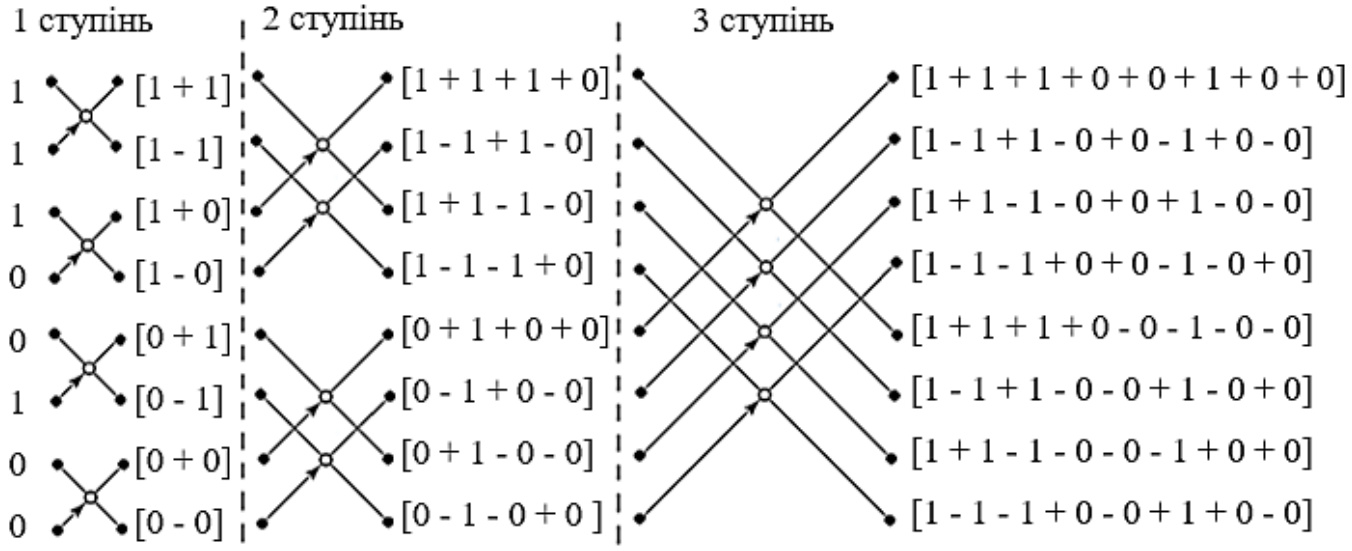

Рисунок 22 «Алгоритм шифрування 8-біт деревом ШПФ за Уолшом-Кулі»

Виконавши обрахунки після 3-го ступеня шифрування, отримуємо зашифровані 8 біт – 4 0 2 -2 2 2 0 0. Тепер розшифруємо дані тим самим алгоритмом, а саме деревом ШПФ(див. рис. 7).

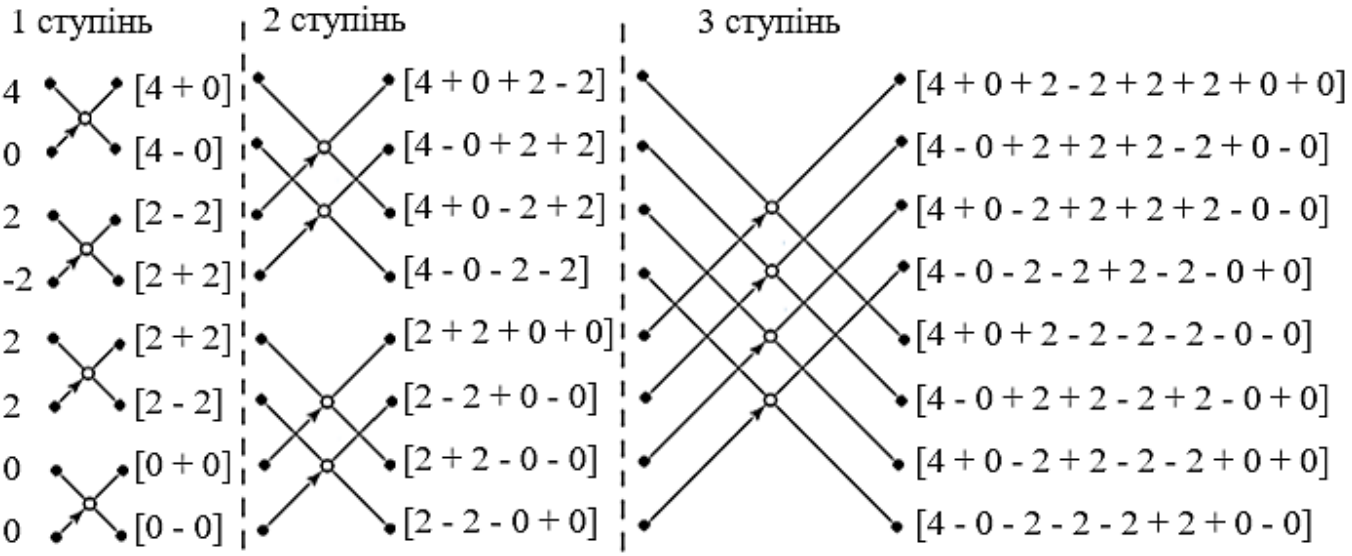

Рисунок 23 «Алгоритм розшифрування 8-біт деревом ШПФ за Уолшом-Кулі»

Виконавши обрахунки після 3-го ступеня розшифрування на виході отримуємо – 8 8 8 0 0 8 0 0. Після ділення розшифрованих даних – 8 8 8 0 0 8 0 0 на $N = 8$, отримуємо наші вхідні 8 біт – 1 1 1 0 0 1 0 0. Якщо ці дані переставити за допомогою ППКГ отримаємо вхідні дані Уолша-Пелі – 1 1 0 1 0 0 1 0 і якщо ці біти переставити двійковою інверсією, отримаємо початкові вхідні дані - 1 0 0 1 1 0 1 0 Уолша-Адамара.

РОЗДІЛ 3
СТОХАСТИЧНЕ ШИФРУВАННЯ

3.1 Алгоритм стохастичного шифрування даних

Суть стохастичного алгоритму шифрування полягає в наступному. Вхідний текст розбивається на чотирьохбайтні блоки ***X*** (1byte = 8bit), тобто в одному блоці зашифрується 32 біти. Процес шифрування чотирьохбайтних блоків ШПФ - Уолша містить п'ять етапів (ступенів), які показані на рис. 12. Саме дерево ШПФ – Уолша дещо відрізніться(див. рис 10), оскільки на вході буде подано не 8 біт, а 32, тому до етапів шифрування додасться ще 2 етапи і всього їх буде 5. Дерево ШПФ – Уолша для 32 бітів можна побачити на рис. 13.

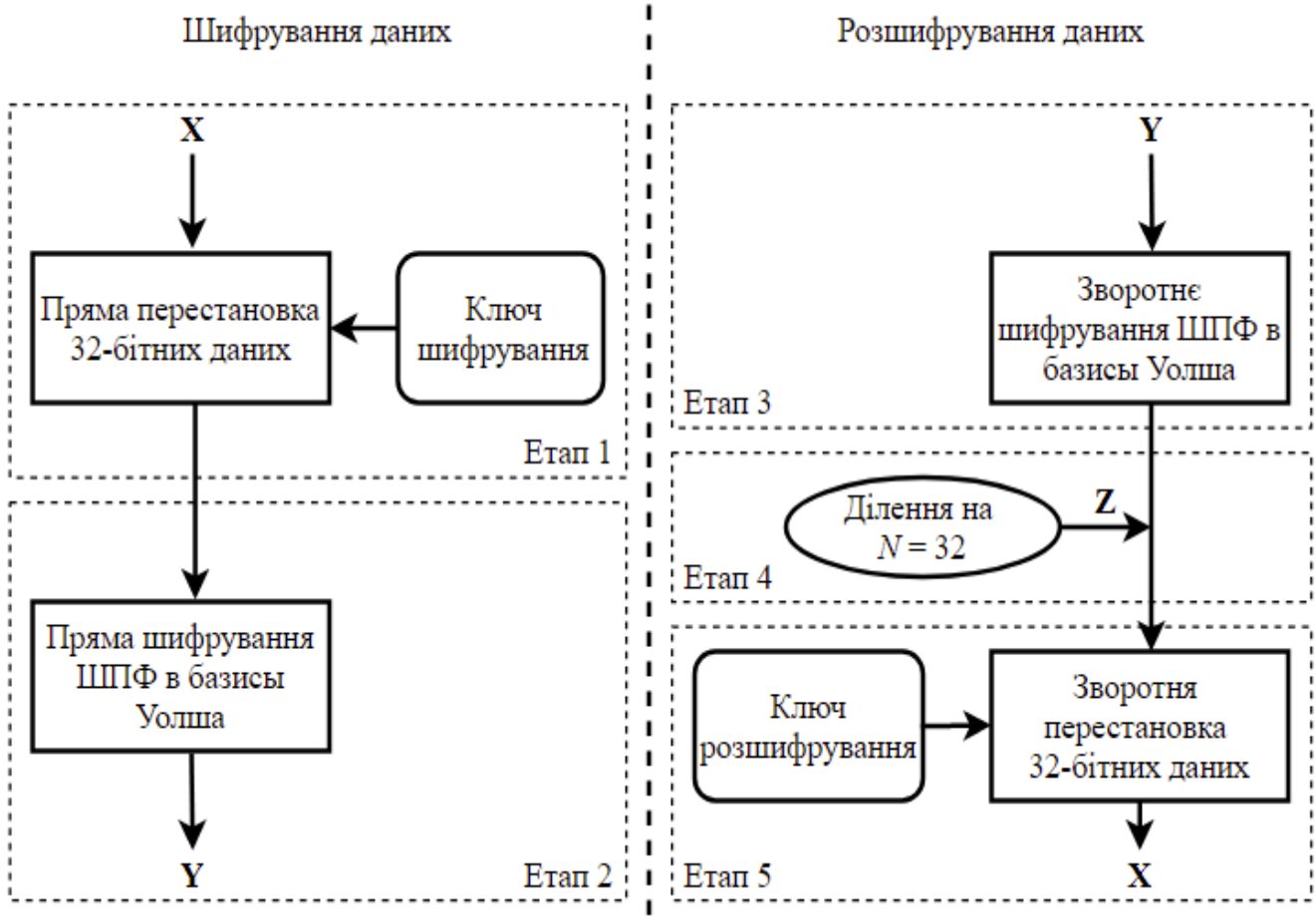

Рисунок 24 «Етапи стохастичного шифрування»

При шифруванні вхідні біти, чотирьохбайтного блоку ***X***, піддаються стохастичній перестановці, керованої секретним «Ключем зашифровування», який збігається з «Ключем розшифровування». Всього існує $32!(\approx 2{,}6 \cdot 10^{35})$ таких ключів. Кожен розряд ***Y*** після шифрування ШПФ - Уолша передається байтами, в

яких молодші сім бітів містять абсолютне значення зашифрованих даних розряду ***Y***, а старший восьмий біт призначений для ідентифікації знаку розряду ***Y***, так як після шифрування ми можемо отримати від'ємне значення, «+» кодується 0, а «-» 1.

При розшифруванні, зашифровані біти **Y** знову поступають на дерево ШПФ-Уолша, після чого на виході зворотного ШПФ байти розрядів ***Z*** діляться на 32, а результати ділення (1 або 0) стають розрядами 32-бітних вихідних даних. Потім вихідні біти піддаються зворотній перестановці, керованої секретним «Ключем розшифровування».

Головна перевага стохастичного шифрування над звичайним шифрування різними системами функцій Уолша(Адамара, Пелі, Качмажа, Кулі) полягає в тому, що при розшифруванні даних знайти правильний алгоритм перестановки вихідних даних, для отримання вхідної інформації неможливо без ключа шифрування, або це займе дуже багато часу. На сьогоднішній день не існує комп'ютера який здатний перевірити всі можливі варіанти перестановок, за кінцевий проміжок часу, для отримання правильного розміщення вхідних даних. В свою чергу вхідні дані зашифровані різними системами функцій Уолша, деревом ШПФ – Уолша, можливо визначати простим перебором впорядкованих перестановок вихідних даних: двійковою інверсією, ЗЛКГ, ЗПКГ, ПЛКГ, ППКГ.

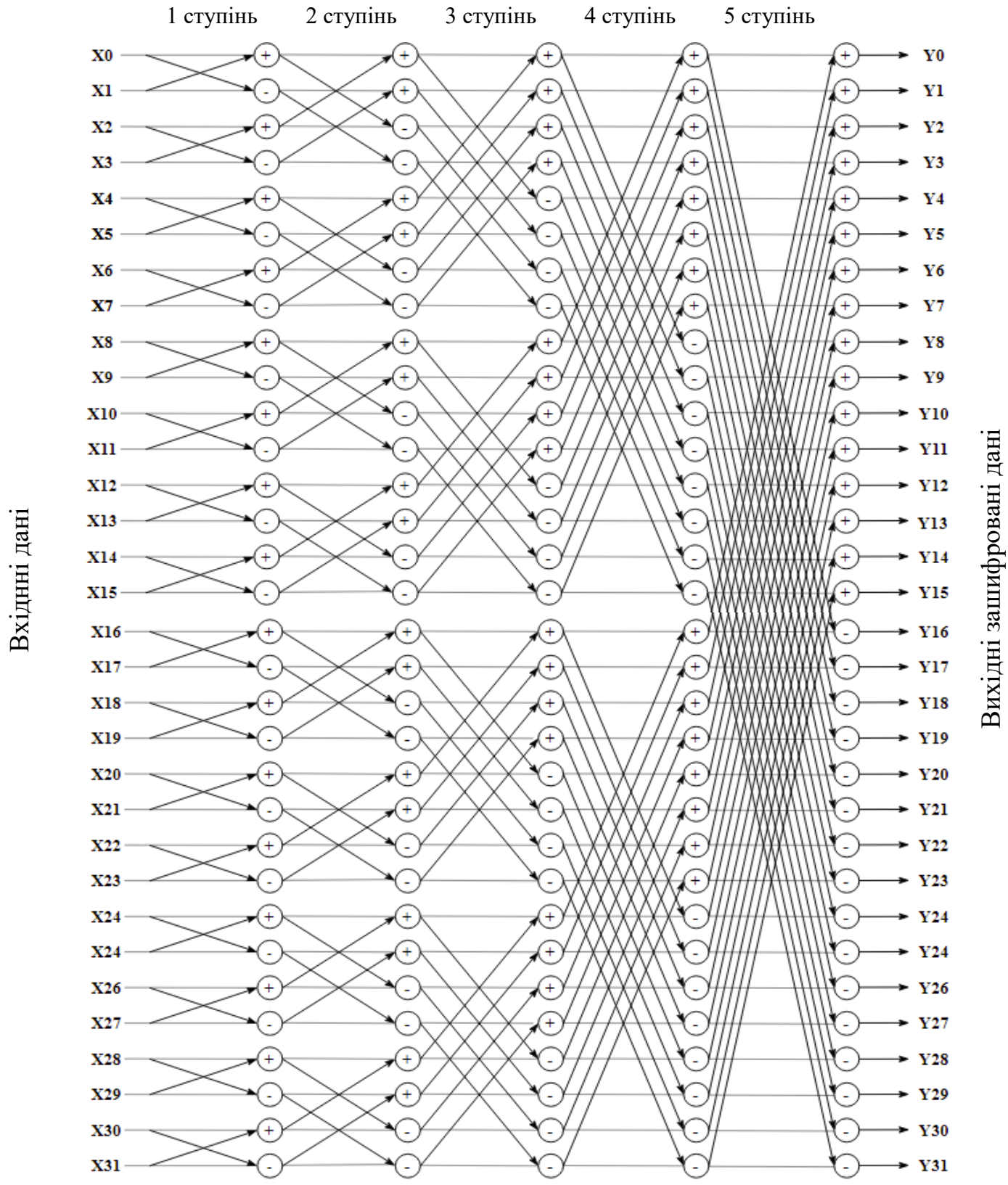

Рисунок 25 «Дерево ШПФ для 32 вхідних бітів»

3.2 Стохастичного шифрування даних

Для прикладу шифрування чотирьохбайтного блоку візьмемо випадкові вхідні 32 біти – 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 1. Наступним етапом є генерація стохастичного ключа шифрування, який співпадає з ключем розшифрування k_{32} = 19 25 14 13 22 17 10 26 24 3 5 18 11 28 31 30 9 6 16 1 8 0 2 7 20 21 27 4 15 23 12 30. Переставляємо вхідні біти згідно з секретним «Ключем шифруванням» і отримуємо на вході процесора біти які потрібно зашифрувати ШПФ – Уолша - 1 1 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0 0 0. Дану перестановку можна побачити в табл. 9.

Таблиця 9

Стохастична перестановка ключем шифрування

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	1	1	0	1	1	0	1	0	1	0	0	1	1

↓

Стохастична перестановка ключем шифрування

↓

19	25	14	13	22	17	10	26	24	3	5	18	11	28	31	30
1	1	1	0	1	1	0	0	1	1	0	0	1	1	1	0

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	0	1	1	0	1	0	1	1	0	1	1	0	0	1

↓

Стохастична перестановка ключем шифрування

↓

9	6	16	1	8	0	2	7	20	21	27	4	15	23	12	30
1	1	1	0	0	1	1	1	1	0	1	1	1	0	0	0

Процес стохастичного шифрування вхідних 32 бітів, дерем ШПФ-Уолша можна побачити на рис. 14, 15, 16.

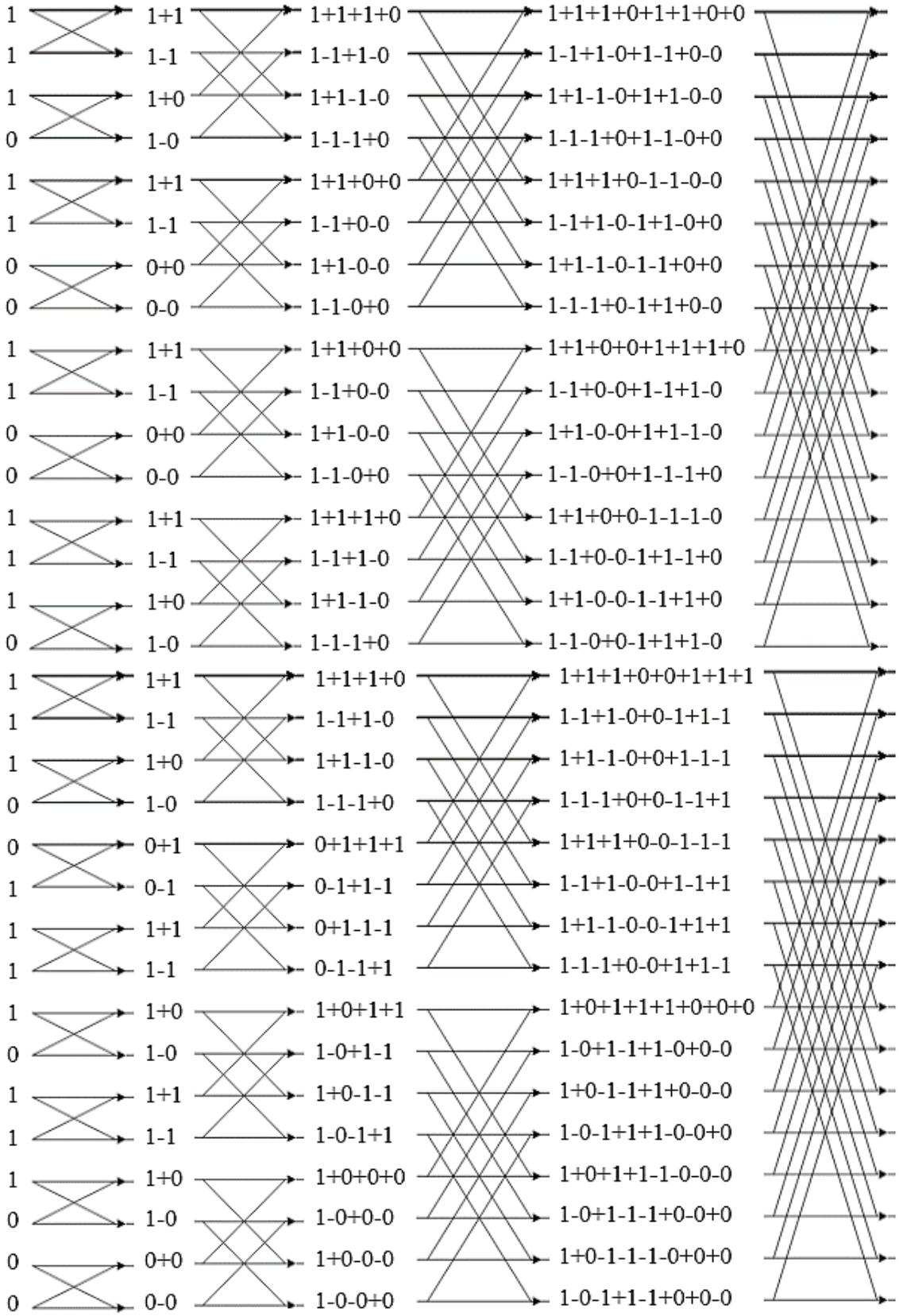

Рисунок 26 «Перші три етапи шифрування даних деревом ШПФ - Уолша»

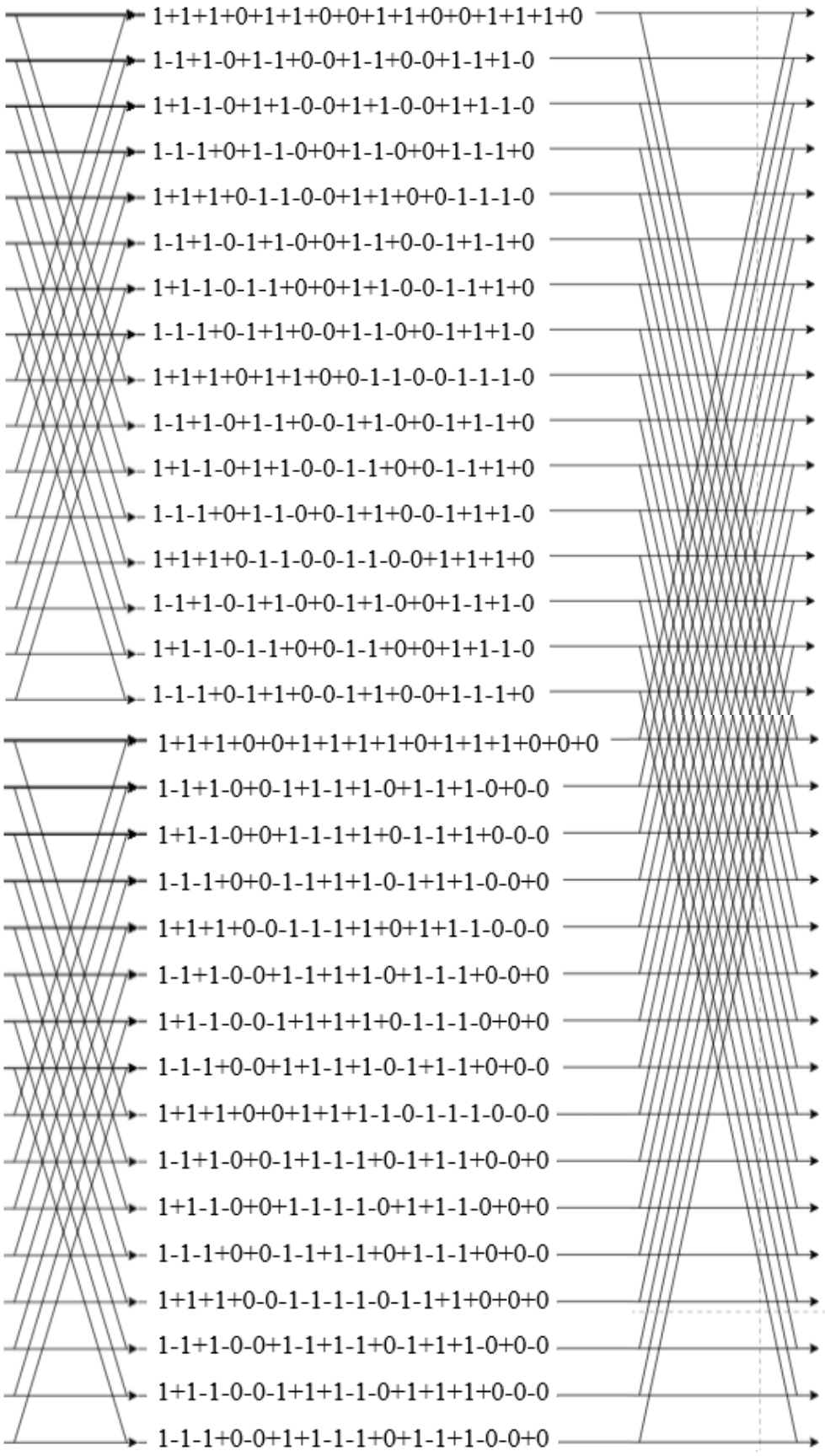

Рисунок 27 «Четвертий етап шифрування даних деревом ШПФ - Уолша»

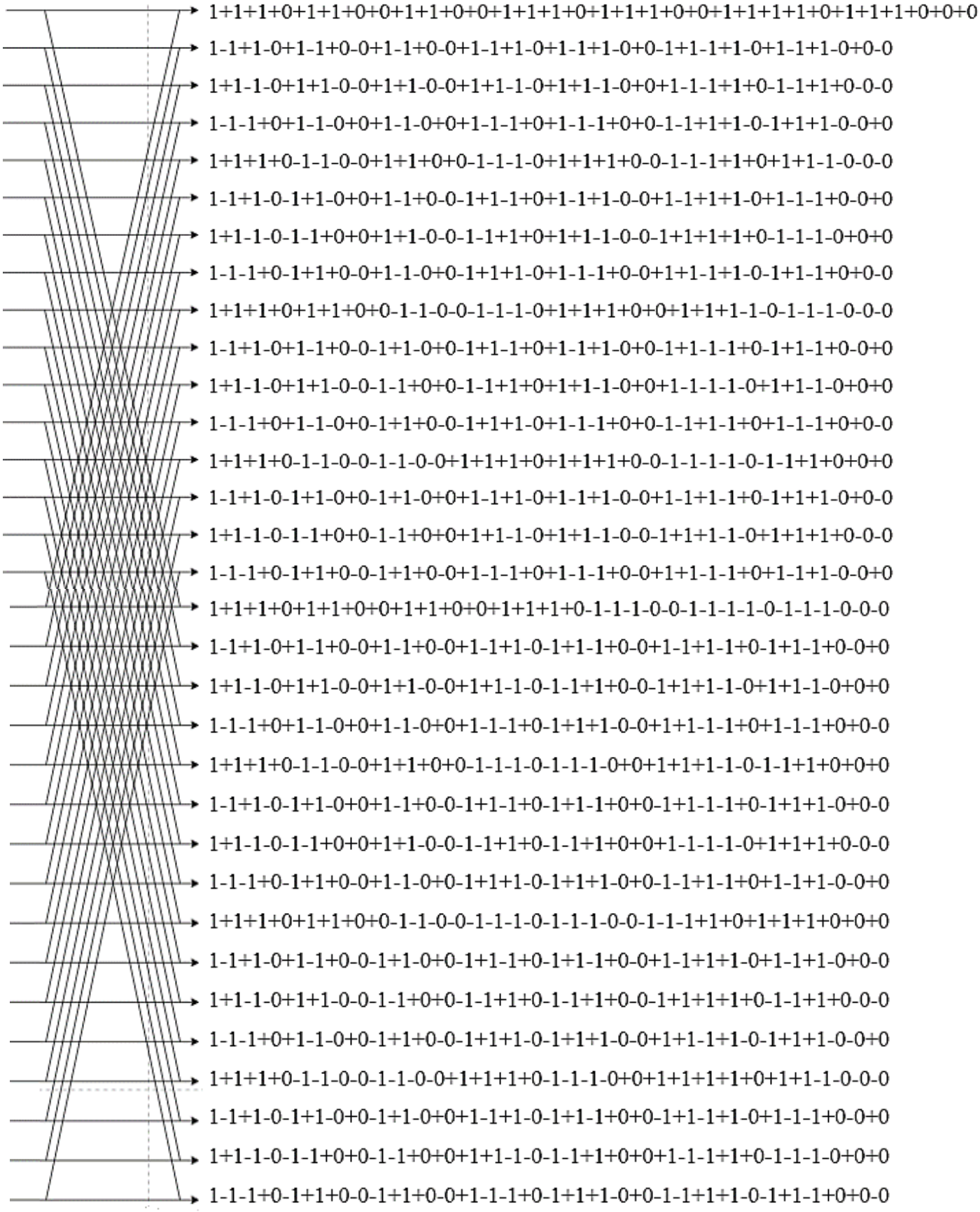

Рисунок 28 «П’ятий етап шифрування даних деревом ШПФ - Уолша»

Виконавши всі обрахунки після п’ятого етапу шифрування деревом ШПФ – Уолша, на виході отримаємо такі зашифровані біти – 20 4 6 -2 2 0 0 2 -2 0 -4 0 4 2 -2 0 0 6 -2 -2 -2 0 0 -2 2 0 4 4 0 -6 -2. Перший біт який ми отримали дорівнює 20, що відповідає кількості одиниць на вході процесора. Це означає коректну роботу стохастичного алгоритму шифрування даних за допомогою ШПФ – Уолша.

3.3 Стохастичне розшифрування даних

Для розшифрування даних використовується все те ж дерево ШПФ-Уолша на вхід якого ми подаємо зашифровані біти – 20 4 6 -2 2 0 0 2 -2 0 -4 0 4 2 -2 0 0 6 -2 -2 -2 0 0 -2 2 0 4 4 0 -6 -2 і виконуємо алгоритм з 5 етапів, який представляє собою операції додавання і віднімання вхідних даних. Після чого, виконавши всі обрахунки, на виході зворотного ШПФ-Уолша ми отримуємо вихідні 32 біти – 32 32 32 0 32 32 0 0 32 32 0 0 32 32 32 0 32 32 32 0 0 32 32 32 32 0 32 32 32 0 0 0, які діляться на N = 32. Результати ділення (1 або 0) стають розрядами 32-бітних вихідних даних. Потім вихідні біти піддаються зворотній перестановці, керованої секретним «Ключем розшифровування». Вкінці ми отримуємо наші зашифровані 32 біти – 1 1 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0 0 0. Процес розшифрування даних алгоритмом ШПФ-Уолша можна побачити на рис. 17, 18, 19.

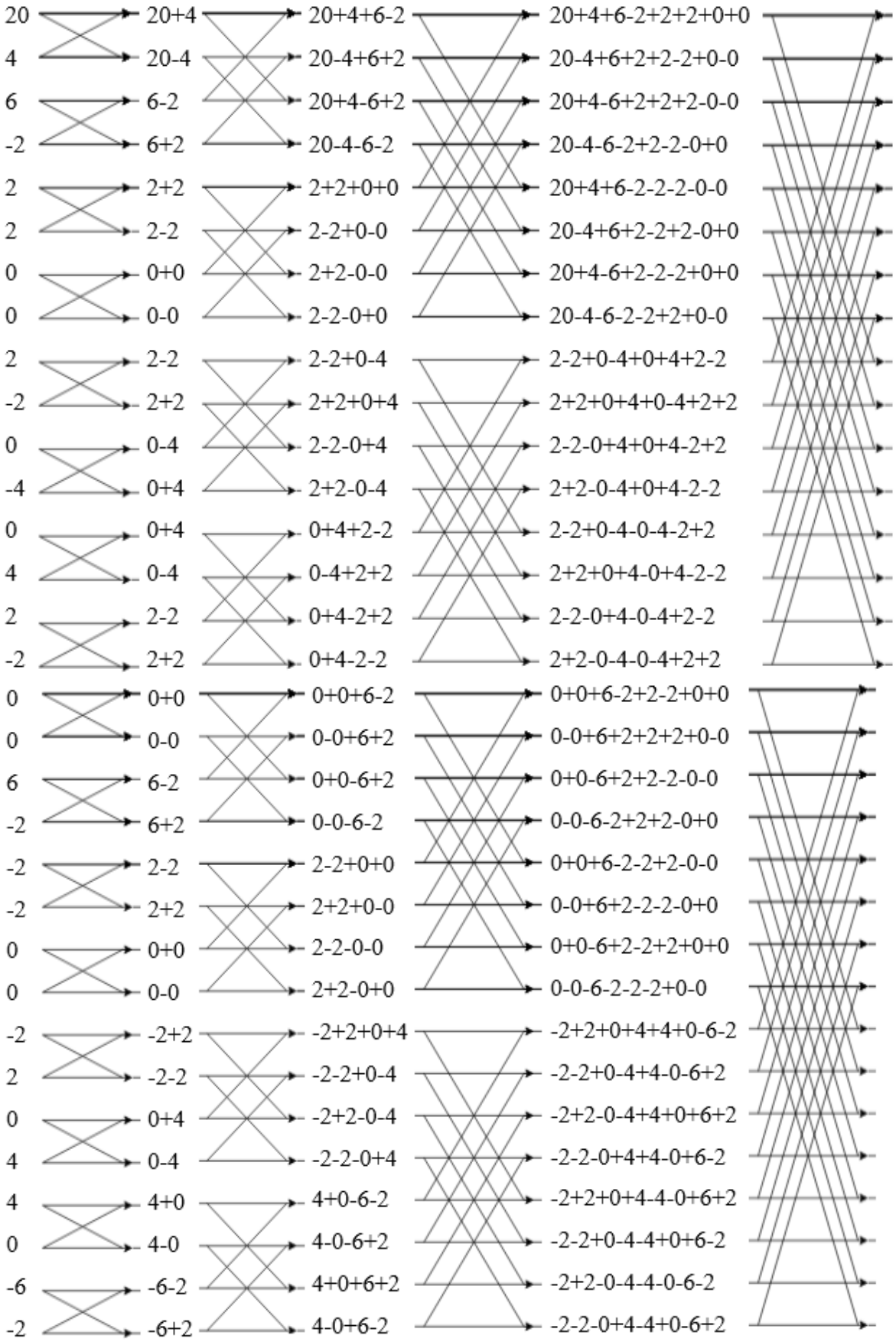

Рисунок 29 «Перші три етапи розшифрування даних Дерем ШПФ - Уолша»

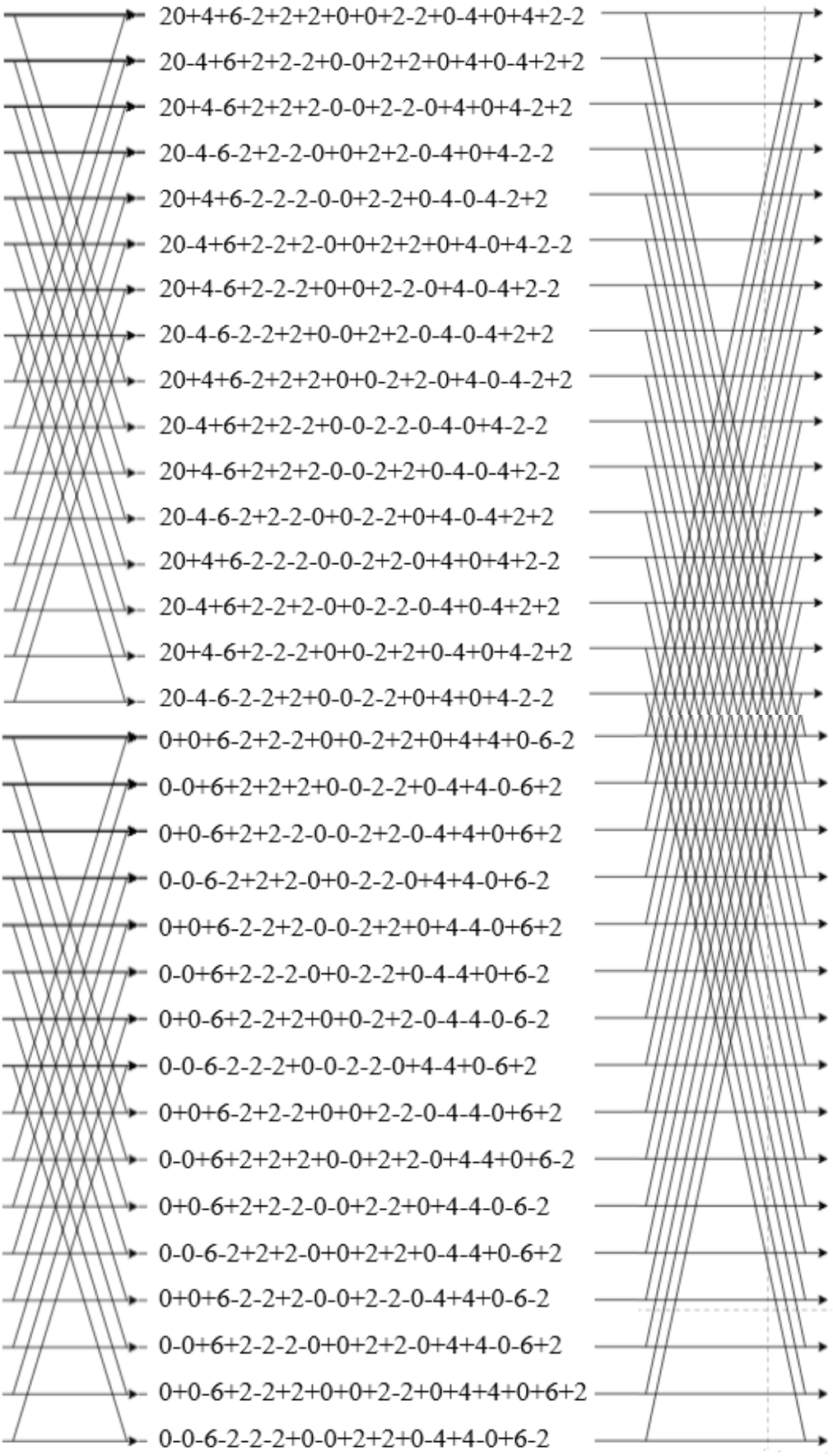

Рисунок 30 «Четвертий етап розшифрування даних Дерем ШПФ - Уолша»

Рисунок 31 «П’ятий етап розшифрування даних Дерем ШПФ - Уолша»

ВИСНОВКИ

Безсумнівно, криптографія буде розвиватися далі. Одна з її завдань на майбутнє це розробка швидкісних методів шифрування з високим рівнем секретності. Це завдання обумовлена великою кількістю каналів зв'язку по яких передаються дуже великі обсяги інформації. Дана робота присвячена розробці програмного комплексу потокового шифрування даних на основі стохастичних матриць Уолша.

За рахунок того, що всі перетворення, які виконуються алгоритмом, стають залежними не лише від секретного ключа, але і від даних, що шифруються, зашифрована інформація має більший рівень захисту від злому.

На основі запропонованих алгоритмічних рішень розроблено методи криптографічної обробки даних (блокового, потокового), шифрування даних, що сприяло підвищенню стійкості, швидкодії і зменшенню необхідної кількості обчислювальних ресурсів для забезпечення криптографічного захисту інформації. В основі потокового шифрування покладені, так звані, матриці Уолша та алгоритм швидкого перетворення Фур’є.

Розроблено програмне забезпечення і проведені випробування шифрування даних з різним об’ємом інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білецький А. Я. Основы теории помехоустойчивого кодирования/ А. Я. Белецкий. – К.: навчальний посібник. / НАУ. Київ, 2018. 38-53 с. 79-89с
2. Білецький А. Я. Обобщённые преобразования Грея и синтез симметричных систем функций Уолша / А. Я. Белецкий. С.1-19.
3. Білецький А. Я. Криптографические приложения индикаторных матриц систем функций Уошла: Захист інформації / А. Я. Белецкий // Буд-во НАУ. Київ. – 2016. - ТОМ 18. - №1. – С. 5-15. – Бібліогр.: 1 назв.
4. Білецький А. Я. Сравнительній анализ эфективности алгоритмов быстрого преобразования Фурье в базисах систем функций Уолша и золотого сечения: Захист інформації / А. Я. Белецкий // Буд-во НАУ. Київ. – 2017. - ТОМ 19. - №1. – С. 23-32. – Бібліогр.: 4 назв.
5. Білецький А. Я. Оптимальные Уолша и Уолша-подобные базисы дискретного преобразования Фурье: Захист інформації / А. Я. Белецкий // Буд-во НАУ. Київ. – 2018. - ТОМ 20. - №.2. – С. 104-119. – Бібліогр.: 5 назв.

ДОДАТКИ

Додаток А

Вихідний код графічного інтерфейсу

```
<Window x:Class="ScramblerWalsh.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:local="clr-namespace:ScramblerWalsh"
      mc:Ignorable="d"
       Title="MainWindow" Height="480" Width="800">
   <Window.Resources>
      <Style TargetType="Control" x:Key="baseStyle">
         <Setter Property="FontSize" Value="14pt"/>
         <Setter Property="FontFamily" Value="Times new Roman"/>
         <Setter Property="FontStyle" Value="Normal" />
         <Setter Property="Margin" Value="5"/>
      </Style>
      <Style TargetType="Control" x:Key="StyleRadioButton">
         <Setter Property="Margin" Value="0"/>
         <Setter Property="FlowDirection" Value="RightToLeft"/>
      </Style>
      <Style TargetType="RadioButton"  BasedOn="{StaticResource StyleRadioButton}"></Style>
      <Style TargetType="Label"  BasedOn="{StaticResource baseStyle}"></Style>
      <Style TargetType="ComboBox" BasedOn="{StaticResource baseStyle}"></Style>
      <Style TargetType="TextBox" BasedOn="{StaticResource baseStyle}"></Style>
      <Style TargetType="ListBox" BasedOn="{StaticResource baseStyle}"></Style>
      <Style TargetType="Button" BasedOn="{StaticResource baseStyle}"></Style>
      <Style TargetType="CheckBox" BasedOn="{StaticResource baseStyle}"></Style>
      <Style TargetType="GroupBox" BasedOn="{StaticResource baseStyle}"></Style>
   </Window.Resources>
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="153"/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition Height="80"/>
         <RowDefinition Height="75"/>
         <RowDefinition/>
         <RowDefinition Height="39"/>
      </Grid.RowDefinitions>
      <GroupBox Header="Вхідний файл"
            Margin="5,10,5,5" Grid.ColumnSpan="2">
         <Grid>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="300*"/>
               <ColumnDefinition Width="47*"/>
            </Grid.ColumnDefinitions>
            <TextBox x:Name="TextBoxInputFile"
```

```
            Margin="5"
            TextChanged="TextBoxInputFile_TextChanged"/>
        <Button x:Name="ButtonInputFile"
            Grid.Column="1"
            Margin="5"
            Content="Файл" Click="ButtonInputFile_Click"  />
    </Grid>
</GroupBox>
<GroupBox Header="Вихідний файл"
          Margin="5"
          Grid.Row="1" Grid.ColumnSpan="2">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="300*"/>
            <ColumnDefinition Width="47*"/>
        </Grid.ColumnDefinitions>
        <TextBox x:Name="TextBoxOutputFile"
                 Margin="5" />
        <Button x:Name="ButtonOutputFile"
                Grid.Column="1"
                Margin="5"
                Content="Файл" Click="ButtonOutputFile_Click"/>
    </Grid>
</GroupBox>
<GroupBox
    Grid.Row="2"
    Header="Матриця" Margin="5,5,5,0" Height="177" VerticalAlignment="Top">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <RadioButton
            Tag="0"
            Grid.Column="0"
            Grid.Row="0"
            Content="H"
            Checked="RadioButton_Checked"/>
        <RadioButton
            Tag="2"
            Grid.Column="0"
            Grid.Row="1"
            Content="K"
            Checked="RadioButton_Checked"/>
        <RadioButton
            Tag="1"
```

```
            Grid.Column="1"
            Grid.Row="0"
            Content="P"
            Checked="RadioButton_Checked"/>
        <RadioButton
            Tag="3"
            Grid.Column="1"
            Grid.Row="1"
            Content="C"
            Checked="RadioButton_Checked"/>
        <RadioButton
            Tag="4"
            Grid.Column="0"
            Grid.Row="2"
            Grid.ColumnSpan="2"
            Margin="60,0,42,0"
            HorizontalAlignment="Center"
            Content="S"
            Checked="RadioButton_Checked" />
    </Grid>
</GroupBox>
<Grid
    Grid.Row="2"
    Grid.Column="1">
    <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="75"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <GroupBox Header="Режим роботи"
            Grid.RowSpan="1"
            Margin="5,5,5,4" >
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="40"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="31*"/>
            </Grid.ColumnDefinitions>
            <WrapPanel
                Margin="5">
                <RadioButton x:Name="RadiobuttonEncrypt"
                    Margin="5"
                    VerticalContentAlignment="Center"
                    IsChecked="True"
                    Content="Шифрування"/>
                <RadioButton x:Name="RadiobuttonDecipher"
                    Margin="5"
                    VerticalContentAlignment="Center"
                    Content="Розшифрування"/>
            </WrapPanel>
        </Grid>
```

```
        </GroupBox>
        <GroupBox x:Name="GroupBoxKey"
          Grid.Row="1"
          Header="Ключ шифрування"
              Margin="5,6,5,3" >
          <Grid>
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="31*"/>
              <ColumnDefinition Width="8*"/>
            </Grid.ColumnDefinitions>
            <TextBox x:Name="TextBoxKey"
              Margin="5" />
            <Button x:Name="ButtonGenerateVector"
              Grid.Column="1"
              Margin="5"
              Content="Генерувати"
              Click="ButtonGenerateVector_Click_1"/>
          </Grid>
        </GroupBox>
        <Button x:Name="ButtonStart"
            Grid.Row="2"
            Margin="0,5,5,0"
            Height="30"
            Width="154"
            VerticalAlignment="Top"
            HorizontalAlignment="Right"
            Content="Пуск"
            Click="ButtonStart_Click"/>
        <Label x:Name="LabelResult"
            Grid.Row="2" Margin="5,5,162,5"/>
      </Grid>
      <Grid Grid.Row="3" Grid.ColumnSpan="2">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="120"/>
          <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <ProgressBar x:Name="ProgressBarProgress"
            Grid.Column="1"
            Margin="5"/>
        <Label x:Name="Label_time"
            Grid.Column="0"
            Margin="5"
            Content="00:00:00:000"/>
        <Label x:Name="Label_progres"
            Grid.Column="1"
            Margin="5"
            HorizontalContentAlignment="Center"/>
      </Grid>
    </Grid>
</Window>
```

Вихідний код шифрування даних різними системами Уолша(Адамар, Пелі, Качмаж, Кула)

```
public enum TypeCrypt : short
   {
      Adamar,
      Peli,
      Kachmazh,
      Kyli,
   }
   class Encrypting
   {
      public event Action<int> Progress;
      private List<Func<byte[],byte[]>> listActions;
      private CryptoKey Key;
      public Encrypting(CryptoKey key)
      {
         Key = key;
         listActions = new List<Func<byte[], byte[]>>
         {
            EncryptAdamar,
            EncryptPeli,
            EncryptKachmazh,
            EncryptKyli,

            DecipherAdamar,
            DecipherPeli,
            DecipherKachmazh,
            DecipherKyli,
         };
      }
      public void Encrypt(string fileInput, string fileOutput, TypeCrypt type)
      {
         byte[] inputByte = File.ReadAllBytes(fileInput);
         byte[] outputByte = listActions[(int)type].Invoke(inputByte);
         File.WriteAllBytes(fileOutput, outputByte);
      }
      public void Decipher(string fileInput, string fileOutput , TypeCrypt type)
      {
         byte[] inputByte = File.ReadAllBytes(fileInput);
         byte[] outputByte = listActions[(int)type+4].Invoke(inputByte);
         File.WriteAllBytes(fileOutput, outputByte);
      }
      private byte LeftGray(byte value)
      {
         return (byte)(value ^ (value >> 1));
      }
      private byte RightGray(byte value)
      {
```

```
            return (byte)(value ^ (value << 1));
        }
        private byte LeftGrayBack(byte value)
        {
            byte result = 0;
            while (value > 0)
            {
                result ^= value;
                value >>= 1;
            }
            return result;
        }
        private byte RightGrayBack(byte value)
        {
            byte result = 0;
            while (value > 0)
            {
                result ^= value;
                value <<= 1;
            }
            return result;
        }
        private int[] EncryptWalsh(int[] inputArray)
        {
            for (int j = 1; j <= inputArray.Length / 2; j <<= 1)
            {
                inputArray = BPFAdd(inputArray, j);
            }
            return inputArray;
        }
        private byte[] DecipherKyli(byte[] inputByte)
        {
            List<byte> outputByte = new List<byte>();
            int persent = Persent(inputByte.Length);
            for (int i = 0; i < inputByte.Length; i += 8)
            {
                int[] array = inputByte.Skip(i).Take(8).Select(x => (int)unchecked((sbyte)x)).ToArray();
                array = EncryptWalsh(array);
                outputByte.Add(RightGrayBack(Convert.ToByte(string.Join("", array.Select(x => x /
array.Length)), 2)));
                SetProgress(i, persent, inputByte.Length);
            }
            return outputByte.ToArray();
        }
        private byte[] EncryptKyli(byte[] inputByte)
        {
            List<byte> outputByte = new List<byte>();
            int persent = Persent(inputByte.Length);
            for (int i = 0; i < inputByte.Length; i++)
            {
                int[] array = EncryptWalsh(ByteToIntArray(RightGray(inputByte[i])));
                outputByte.AddRange(array.Select(x => (byte)x).ToArray());
```

```
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    private byte[] DecipherKachmazh(byte[] inputByte)
    {
        List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);
        for (int i = 0; i < inputByte.Length; i += 8)
        {
            int[] array = inputByte.Skip(i).Take(8).Select(x => (int)unchecked((sbyte)x)).ToArray();
            array = EncryptWalsh(array);
            outputByte.Add(LeftGrayBack(Convert.ToByte(string.Join("", array.Select(x => x /
array.Length)), 2)));
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    private byte[] EncryptKachmazh(byte[] inputByte)
    {
        List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);

        for (int i = 0; i < inputByte.Length; i++)
        {
            int[] array = EncryptWalsh(ByteToIntArray(LeftGray(inputByte[i])));
            outputByte.AddRange(array.Select(x => (byte)x).ToArray());
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    private byte[] DecipherPeli(byte[] inputByte)
    {
        List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);
        for (int i = 0; i < inputByte.Length; i += 8)
        {
            int[] array = inputByte.Skip(i).Take(8).Select(x => (int)unchecked((sbyte)x)).ToArray();
            array = EncryptWalsh(array);
            outputByte.Add((byte)~Convert.ToByte(string.Join("",array.Select(x => x / array.Length)), 2));
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    private byte[] EncryptPeli(byte[] inputByte)
    {
        List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);
        for (int i = 0; i < inputByte.Length; i++)
        {
            int[] array = ByteToIntArray((byte)~inputByte[i]);
            array = EncryptWalsh(array);
```

```
            outputByte.AddRange(array.Select(x => (byte)x).ToArray());
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    private byte[] OpenFile(string fileName)
    {
        return File.ReadAllBytes(fileName);
    }
    private void SaveFile(string fileName, byte[] byteFile)
    {
        File.WriteAllBytes(fileName, byteFile);
    }
    public byte[] EncryptAdamar(byte[] inputByte)
    {
        List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);
        for (int i = 0; i < inputByte.Length; i++)
        {
            int[] array = EncryptWalsh(ByteToIntArray(inputByte[i]));
            outputByte.AddRange(array.Select(x => (byte)x).ToArray());
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    public byte[] DecipherAdamar(byte[] inputByte)
    {
        List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);

        for (int i = 0; i < inputByte.Length; i += 8)
        {
            int[] array = inputByte.Skip(i).Take(8).Select(x => (int)unchecked((sbyte)x)).ToArray();
            array = EncryptWalsh(array);
            outputByte.Add(Convert.ToByte(string.Join("", array.Select(x => x / array.Length)), 2));
            SetProgress(i, persent, inputByte.Length);
        }
        return outputByte.ToArray();
    }
    private int[] ByteArrayToBitArray(byte[] array)
    {
        long tmp = 0;
        for(int i = 0; i < array.Length; i++)
        {
            tmp <<= 8;
            tmp = tmp | array[i];
        }
        int[] result = new int[32];
        for(int i = result.Length-1; i >= 0; i--)
        {
            if (tmp == 0)
                break;
```

```
            result[i] = (int)(tmp & 1);
            tmp >>= 1;
        }
        return result;
    }
    private int[] BPFAdd(int[] array, int power)
    {
        int[] result = new int[array.Length];
        int i = 0;
        while (i < result.Length)
        {
            for(int j = 0; j < power; j++)
            {
                result[i] = array[i] + array[i + power];
                i++;
            }
            for (int j = 0; j < power; j++)
            {
                result[i] = array[i-power] - array[i];
                i++;
            }
        }
        return result;
    }
    private int[] ByteToIntArray(byte value)
    {
        int[] result = new int[8];
        for (int i = 7; i >= 0; i--)
        {
            if (value == 0)
                break;
            result[i] = (byte)(value & 1);
            value >>= 1;
        }
        return result;
    }
    private void SetProgress(int iterator, int persent, int length)
    {
        if (iterator % persent == 0)
        {
            int progress = (int)((double)iterator / length * 100);
            Progress.Invoke(progress);
        }
    }
    private int Persent(int length)
    {
        int persent = length / 100 == 0 ? 1 : length / 100;
        persent++;
        return persent;
    }
}
```

Вихідний код стохастичного шифрування даних.

```
public byte[] EncryptStohas(byte[] inputByte)
      {
         List<byte> outputByte = new List<byte>();
         int persent = Persent(inputByte.Length);
         for (int i = 0; i < inputByte.Length; i+=4)
         {
            int[] array = ByteArrayToBitArray(inputByte.Skip(i).Take(4).Select(x=> x).ToArray());
            array = new Transposition(Key.Key).Encrypt(array);
            array = EncryptWalsh(array);
            outputByte.AddRange(array.Select(x => (byte)x).ToArray());
            SetProgress(i, persent, inputByte.Length);
         }
         return outputByte.ToArray();
      }
      public byte[] DecipherStohas(byte[] inputByte)
      {
         List<byte> outputByte = new List<byte>();
        int persent = Persent(inputByte.Length);
         for (int i = 0; i < inputByte.Length; i += 32)
         {
            int[] array = inputByte.Skip(i).Take(32).Select(x => (int)unchecked((sbyte)x)).ToArray();
            array = EncryptWalsh(array).Select(x => x / array.Length).ToArray();
            array = new Transposition(Key.Key).Decrypt(array);
            for (int j = 0; j < array.Length; j += 8)
            {
               outputByte.Add(Convert.ToByte(string.Join("", array.Skip(j).Take(8)), 2));
            }
            SetProgress(i, persent, inputByte.Length);
         }
         return outputByte.ToArray();
      }
```

Вихідний код генерації ключа шифрування.

```
public int[] KeyGenerate()
{
         var random = new Random();
         key = Enumerable.Range(0, 32).ToArray();
         for (int i = key.Length - 1; i >= 1; i--)
         {
            int j = random.Next(i + 1);
            var temp = key[j];
            key[j] = key[i];
            key[i] = temp;
         }
         return key;
}
```