DOI 10.31891/2307-5732-2022-311-4-191-197
УДК 004.056.55

ПАТЛАНЬ Д. В.
Черкаський державний технологічний університет
https://orcid.org/0000-0003-3773-1619
e-mail: d.v.patlan.fetr18@chdtu.edu.ua
ПАЛАГІНА О. А.
Черкаський державний технологічний університет
https://orcid.org/0000-0002-5395-5092
e-mail: palahina@ukr.net
ІВЧЕНКО О. В.
Черкаський державний технологічний університет
https://orcid.org/0000-0002-6716-1939
e-mail: sania_ivchenko@ukr.net
ПАЛАГІН В. В.
Черкаський державний технологічний університет
https://orcid.org/0000-0003-1903-6022
e-mail: palahin@ukr.net

МЕТОД ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ СИМЕТРИЧНОГО БЛОКОВОГО ШИФРУВАННЯ

В роботі показана модифікація алгоритму блокового симетричного шифрування AES (Advanced Encryption Standard). При практичній реалізації модифікованого алгоритму показана можливість об'єднання ряду математичних операцій, які мають схожий принцип опрацювання елементів, що дозволило скоротити час обробки даних при їх шифруванні та дешифруванні у порівнянні з відомою реалізацією. Крім того, в удосконаленій реалізації операція MixColumns використовує попередньо обчислені таблиці пошуку необхідного значення, що також сприяє підвищенню швидкодії перетворень, але за рахунок більшого обсягу пам'яті для розміру коду.

Ключові слова: блоковий шифр, симетричне кодування, об'єднання операцій обробки даних, оптимізація часу обробки даних.

Dana PATLAN, Elena PALAHINA, Oleksandr IVCHENKO, Volodymyr PALAHIN
Cherkassy State Technological University

THE METHOD OF INCREASING THE EFFICIENCY OF SYMMETRICAL BLOCK ENCRYPTION

Ensuring data safety and security are relevant in any sphere of human activity, which is directly related to the implementation of modern and advanced symmetric and asymmetric algorithms and methods of cryptographic protection of information. One of the well-known and safe encryption algorithms, known as AES (Advanced Encryption Standard), which is endowed with the properties of ease of practical implementation and high reliability, has found its wide application for block symmetric data encryption. A feature of symmetric encryption is the processing of large amounts of information, the use of the same key for encryption and decryption, high reliability of data protection. This paper proposes to improve this algorithm to provide faster data processing. All modifications of the AES transformations do not reduce the high cryptographic strength of the cipher, and their principles are described in detail in the paper. This practical implementation of the optimized algorithm shows the possibility of combining a number of mathematical operations that have a similar principle of processing elements, which allowed to reduce the processing time. In addition, in the improved implementation the MixColumns operation uses pre-calculated lookup tables of the required value, which also helps to increase the speed of transformations but at the cost of more memory for the code size. A comparative analysis of the practical implementation of the standard and optimized AES cryptoalgorithms has been carried out, and numerical indicators of the processing time of test data have been obtained, which are presented in the form of tables. Graphs of the dependence of the encryption and decryption procedure execution time on the file size for the standard and optimized AES algorithms have been constructed. The results show a significant percentage of optimization - up to 50% for encryption and up to 75% for decryption of data.

Keywords: block cipher, symmetric coding, combining data processing operations, optimization of data processing time.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Захист інформації від несанкціонованого доступу та її використання має велике значення і реалізується методами криптології. Сучасні проблеми захисту інформації пов'язані з надійністю та швидкодією алгоритмів, які використовуються для обробки даних. Правильний вибір та практичне застосування конкретного алгоритму шифрування залежить безпосередньо від задачі захисту даних. Одним із поширених, надійних та відомих алгоритмів шифрування є AES (Advanced Encryption Standard), також відомий як Rijndael, який є симетричним блоковим шифром [1]. Він має високу ефективність та криптостійкість і підходить для опрацювання великих розмірів даних, є предметом багатьох сучасних досліджень і розробок [2].

Швидкість виконання процедур шифрування та розшифрування даних за допомогою алгоритму AES залежить від розміру ключа, адже в залежності від його довжини варіюється і кількість раундів шифрування, і час формування раундових ключів. Навіть якщо використовувати ключі довжиною в 128 біт, при великих обсягах даних буде відчутна затримка в часі. Тому вже існує безліч ефективних апаратних та апаратно-програмних реалізацій для оптимізації AES [3-9].

Зазначимо, що результати оптимізації роботи алгоритму AES можуть бути розповсюджені і на функціонування симетричного блокового алгоритму «Калина», який став національним стандартом

шифрування [10, 11], що забезпечує високу розсіюваність даних та їх перемішування.

Разом з тим необхідно відмітити гостру необхідність у підвищенні ефективності симетричних блокових криптоалгоритмів через проведення різноманітних оптимізаційних процедур і впровадженні нових методів для скорочення часу обробки, що пов'язано з постійним зростанням потоків даних, які обробляються, в тому числі в режимі реального часу [12].

Щодо прискорення виконання операцій шифрування та дешифрування у програмному коді без використання апаратного прискорення чи додаткових модулів, то таких сучасних реалізацій доволі мало [13]. Таким чином існує необхідність удосконалення алгоритму AES без значних змін у перетвореннях, які можуть вплинути на криптостійкість шифру.

Формулювання цілей статті

Метою роботи є зменшення часу криптографічної обробки даних у симетричному блоковому криптоалгоритмі AES шляхом оптимізації математичних операцій та вдосконалення програмної реалізації без зниження криптостійкості алгоритму.

Метод оптимізації процедурних перетворень

Шифрування та дешифрування AES засновані на чотирьох різних перетвореннях, які виконуються багаторазово в певній послідовності. Математичні основи побудови алгоритму описані в [1, 14] та представлено загальну структуру AES (рис.1), де відбувають раундові процеси шифрування та дешифрування даних, тривалість обробки яких залежить від довжини ключа шифрування.

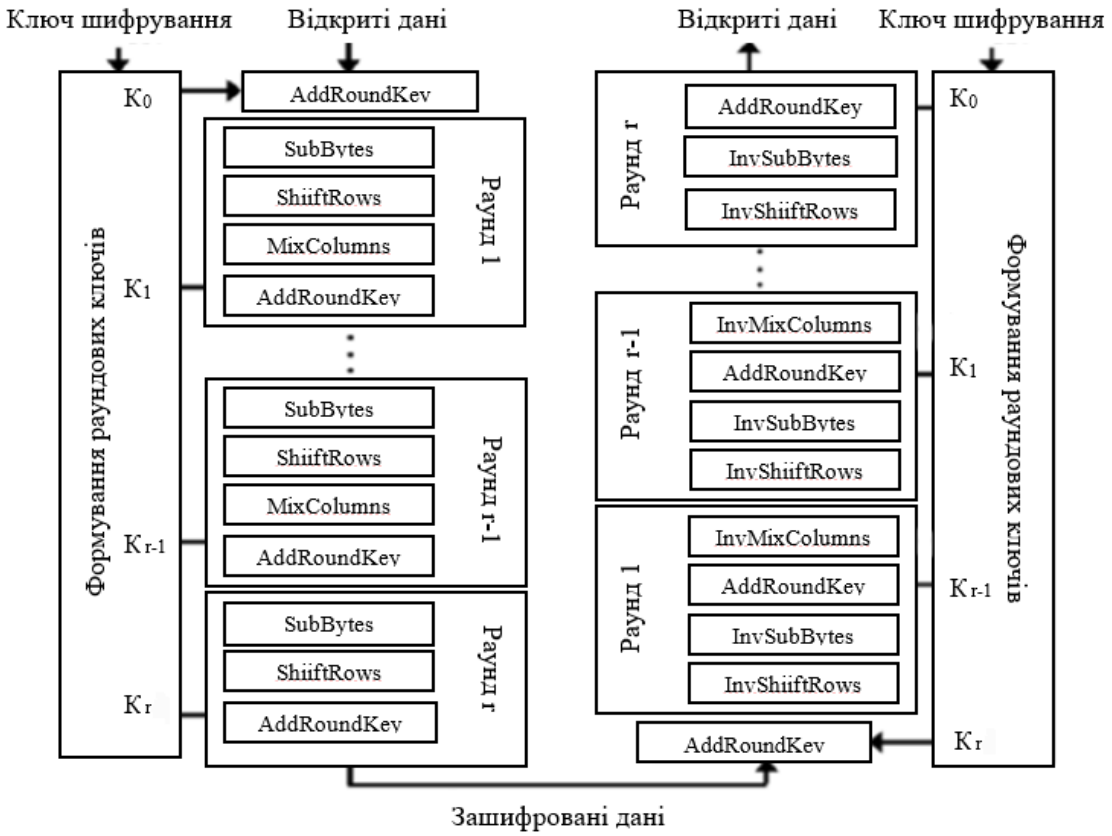

Рис. 1. Загальна структура основних перетворень в криптоалгоритмі AES

Один із шляхів підвищення швидкості обробки даних полягає в удосконаленні принципів роботи з матрицею State – матрицею станів, розмір якої фіксований і дорівнює 128 біт. Така матриця має 4 рядки і 4 стовпці, причому заповнення даного блоку відбувається по стовпцях. Відповідно, після усіх раундових перетворень, результат шифрування потрібно буде також зчитувати по стовпцям вихідного масиву зашифрованих байтів, що вимагає додаткових обчислень та ініціалізацію

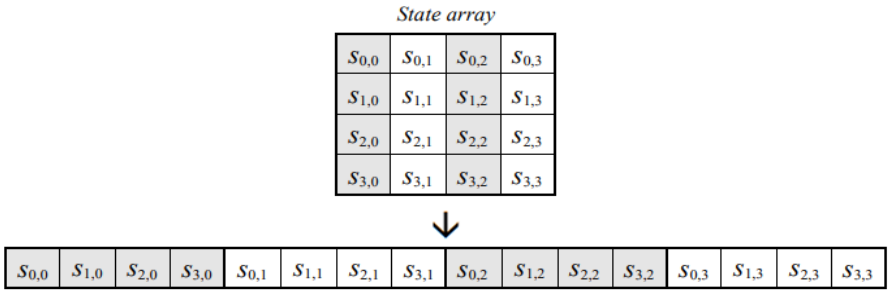

Рис. 2. Заповнення модифікованого масиву State

циклу. Заповнимо блок даних у вигляді одновимірного масиву, але теж по стовпцям. Тоді масив State буде мати принцип заповнення, представлений на рис. 2.

Оскільки операції MixColumns та AddRoundKey працюють зі стовпцями, за такого заповнення модифікація цих перетворень не буде складною. До того ж, тоді буде відсутня необхідність перетворення матриці результату у рядок даних.

Крім модифікацій окремих раундових перетворень, можна прискорити виконання шифрування шляхом їх об'єднання. Наприклад, якщо об'єднати дві операції, які мають схожі цикли у програмному коді, то він буде проходити вдвічі менше ітерацій циклів, що значно скоротить час обробки даних. Отже, при практичній реалізації оптимізованого алгоритму, потрібно за можливості об'єднати операції, які мають схожий принцип опрацювання елементів.

MixColumns – складна процедура змішування колонок, де кожна колонка матриці State перемножується з фіксованим многочленом, який має еквівалентний матричний запис (1).

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad (1)$$

Принцип операцій InvMixColumns той самий, але кожна колонка State перемножується з іншим визначеним многочленом. У матричній формі це виглядає наступним чином (2):

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad (2)$$

Для опрацювання однієї колонки потрібно виконати 8 (адже множення на {01} не змінює числа) операцій множення для шифрування та 16 операцій для дешифрування, тоді тільки для одного раунду – це 24/64 операції множення на визначені константи. Замість реалізації таких операцій множення можна використати попередньо обчислені таблиці для пошуку необхідного значення. Тоді потрібно визначити 6 таких таблиць – дві для процесу шифрування та чотири для процесу розшифрування.

Запишемо у рядок даних результати множення усіх можливих чисел до 1 байту (256 біт) на {02}. Те ж саме потрібно обрахувати для інших констант – {03}, {09}, {0b}, {0d}, {0e} (табл. 1). Отже, в програмній реалізації необхідно ініціювати 6 таблиць по 256 біт [15].

Таблиця 1

Приклад формування допоміжних таблиць для MixColumns

	Число, яке потрібно помножити										
	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**…**
Результат множення на {02}	00	02	04	06	08	0a	0c	0e	0e	12	…
Результат множення на {03}	00	03	06	05	0c	0f	0a	09	18	1b	…
А також множення на {09}, {0b}, {0d}, {0e}											

		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
x	**0**	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рис. 3. Таблиця підстановок для операції SubBytes

Тоді залишається лише знайти в таблиці потрібне число. Значення елементу масиву State і буде індексом, за яким потрібно шукати елемент у визначених таблицях для MixColumns. А оскільки дане перетворення має пройти 4 ітерації, як і перетворення AddRoundKey, можна їх об'єднати в одну функцію. Проте варто зазначити, що при шифруванні спочатку виконується операція MixColumns, а потім AddRoundKey, а при дешифруванні – навпаки. До того ж, в останньому раунді не можна змішувати колонки, для забезпечення розшифрування тексту, тому хорошим рішенням є залишити функцію AddRoundKey, яка буде застосовуватися лише в останньому раунді шифрування. Така реалізація дає змогу позбавитися 36-48 (в залежності від кількості раундів) ітерацій проходження циклів.

SubBytes – це операція заміни байтів, яка працює незалежно від кожного байта матриці даних за допомогою таблиці підстановки S-box (рис. 3). Наприклад, якщо елемент матриці s={53}, то значення підстановки буде визначатися перетином рядка з індексом «5» і стовпця з індексом «3». Значення елементу s перетвориться на {ed}.

Візуально можна легко розділити значення елементу на дві частини і отримати індекси рядка і стовпця таблиці підстановок. Проте програмна реалізація потребує математичних перетворень для визначення окремо першої та другої цифри елементу. Для вирішення цієї проблеми можна записати таблицю S-box у вигляді одновимірного масиву з 256 елементами. Тоді не потрібно обчислювати індекси рядка та стовпця. Наприклад, елемент матриці s={53} стане 53_{16}-тім елементом в масиві S-box, і це значення співпадає зі значенням на перетині рядка з індексом «5» і стовпця з індексом «3». Тоді програма буде відразу шукати потрібний елемент в таблиці констант. І хоча пошук такого елемента вручну здається трудомістким процесом, програмна реалізація позбавиться щонайменше 32 операцій ділення за один раунд шифрування та 22-30 (в залежності від кількості раундів) операцій під час процесу розширення ключа.

ShiftRows – це операція впорядкування елементів матриці State, яка виконує циклічний зсув кожного рядка. Довжина циклічного зсуву в кожному рядку різна. Нульовий рядок залишається без змін, у першому рядку кожен елемент зсувається на одну позицію ліворуч, елементи другого і третього рядка зміщуються на дві та три позиції відповідно.

Дане перетворення працює з рядками, а оновлений блок даних State має лише один, причому елементи одного рядка зі звичайної матриці розкидані по всьому масиву. Отже, необхідно знайти зручний і швидкий спосіб реалізації циклічного зсуву. Для кожного елементу такого блоку можна створити таблицю індексів [16]. Така таблиця буде вказувати, куди повинен переміститися кожний елемент. Наприклад, якщо нульовий рядок не зміниться, отже 0-й, 5-й, 9-й і 13-й елементи повинні так і залишитися на 0-й, 5-й, 9-й і 13-й позиції відповідно. Вирахуємо індекси інших елементів та занесемо їх в таблицю (табл. 2).

Таблиця 2

Зіставлення елементів в операції ShiftRows

Нове значення індексу	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Старе значення індексу	0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

InvShiftRows здійснює циклічний зсув вже вправо на 1 елемент для першого рядка State, на 2 для другого і на 3 для третього. Нульовий рядок знову не змінюється. Відповідно і для інверсного перетворення також потрібно створити таку таблицю (табл. 3).

Таблиця 3

Зіставлення елементів в операції InvShiftRows

Нове значення індексу	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Старе значення індексу	0	13	10	7	4	1	14	11	8	5	2	15	12	9	6	3

Об'єднаємо змінені операції ShiftRows та SubBytes у одну функцію. Це дозволить зменшити кількість циклів у обох операціях; програмна реалізація загалом позбавиться 160-224 (в залежності від кількості раундів) ітерацій проходження циклу. Тобто, завдяки такому об'єднанню, замість 32 ітерацій за раунд вже будемо використовувати 16.

На рис.4. представлена спрощена схема оптимізованого криптоалгоритму AES, де процес дешифрування матиме схожий принцип.

Результати впровадження оптимізації процедурних перетворень та їх обговорення

Для визначення ступеня оптимізації криптоалгоритму AES необхідно порівняти час, який потрібен для шифрування та дешифрування для двох програмних версій алгоритму – традиційної та удосконаленої. Для програмної реалізації алгоритму було обрано мову програмування з відкритим вихідним кодом – Python. Оцінка продуктивності на основі часу виконання програми представлена в операційній системі Windows 10.

Оскільки стандарт підтримує три довжини ключа 128 біт, 192 біт та 256 біт, то для отримання кращої порівняльної характеристики були проведені дослідження для цих трьох випадків. За основу взято

текстові файли різного розміру – 1 кБ, 10кБ, 50кБ, 100кБ та 200кБ, що дозволить оцінити залежність часу обробки від довжини повідомлення.

Складемо порівняльну таблицю часу шифрування (табл. 4) та розшифрування (табл. 5) криптоалгоритмів. Колонка «До» містить час шифрування для класичної реалізації шифру, колонка «Після» - після його удосконалення. Колонка «Економія часу» має розрахований відсоток економії часу, необхідного для обробки файлу завдяки оптимізації суміщення операцій.

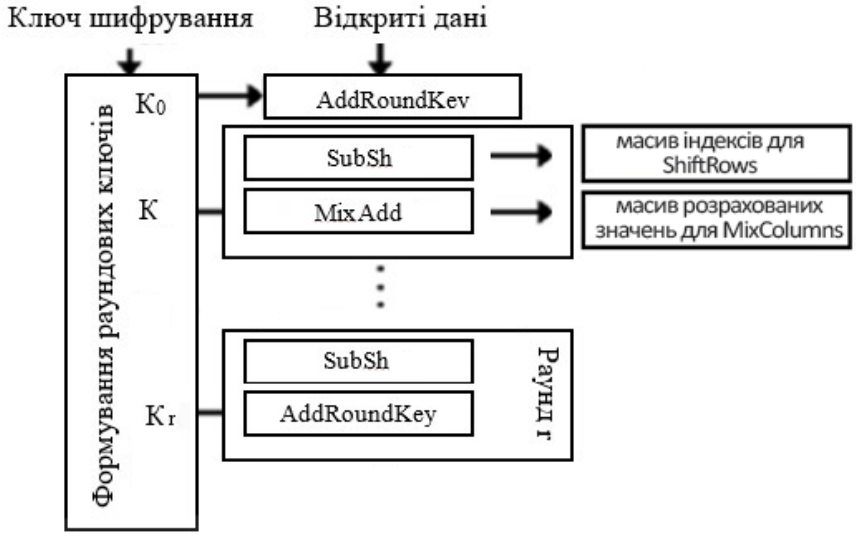

Рис. 4. Спрощена схема оптимізованого методу обробки даних

Таблиця 4

Результати оптимізації алгоритму: шифрування

Шифрування									
	AES-128			**AES-192**			**AES-256**		
Розмір даних, кБ	**До, с**	**Після, с**	**Економія часу, %**	**До, с**	**Після, с**	**Економія часу, %**	**До, с**	**Після, с**	**Економія часу, %**
1	0,02	0,01	50	0,02	0,01	52	0,03	0,01	53
10	0,23	0,12	48	0,27	0,13	52	0,3	0,15	50
50	1,43	0,86	40	1,61	0,95	41	1,78	1,01	43
100	3,57	2,42	33	3,93	2,59	34	4,32	2,75	36
200	9,74	7,52	23	10,48	7,68	27	10,9	7,85	28

Таблиця 5

Результати оптимізації алгоритму: дешифрування

Дешифрування									
	AES-128			**AES-192**			**AES-256**		
Розмір даних, кБ	**До, с**	**Після, с**	**Економія часу, %**	**До, с**	**Після, с**	**Економія часу, %**	**До, с**	**Після, с**	**Економія часу, %**
1	0,05	0,01	76	0,06	0,01	75	0,07	0,01	77
10	0,52	0,13	75	0,62	0,15	76	0,72	0,17	76
50	2,87	0,93	68	3,37	1,06	69	3,89	1,09	72
100	6,45	2,51	61	7,43	2,73	63	8,48	2,93	65
200	15,53	7,74	50	17,35	7,89	55	19,31	8,19	58

Отже, як бачимо з таблиць, дешифрування у звичайному криптоалгоритмі займає майже вдвічі більше часу, ніж шифрування. Очевидно, це пов’язано з більшою кількістю та складністю операцій множення перетворення InvMixColumns. Тому удосконалення саме цієї функції вагомо покращує показники. В цілому, усі незначні модифікації в сукупності дають вагомий результат оптимізації як процесу шифрування, так і процесу дешифрування.

Оскільки для різної довжини ключа значення часу обробки даних дуже схожі, можна взяти середні показники (а саме для середньої довжини ключа – 192 біт) для формування графіків порівняння необхідного часу для шифрування (рис. 5) та для дешифрування (рис. 6) реалізованих криптоалгоритмів.

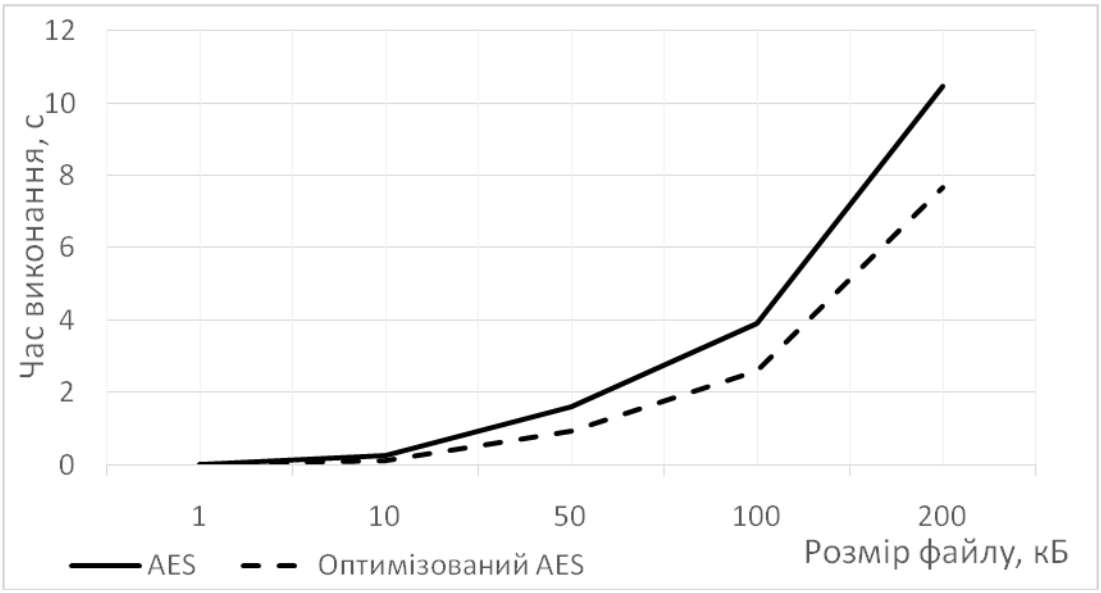

Рис. 5. Залежність часу виконання процедур шифрування від розміру файлу для стандартного і оптимізованого алгоритмів AES.

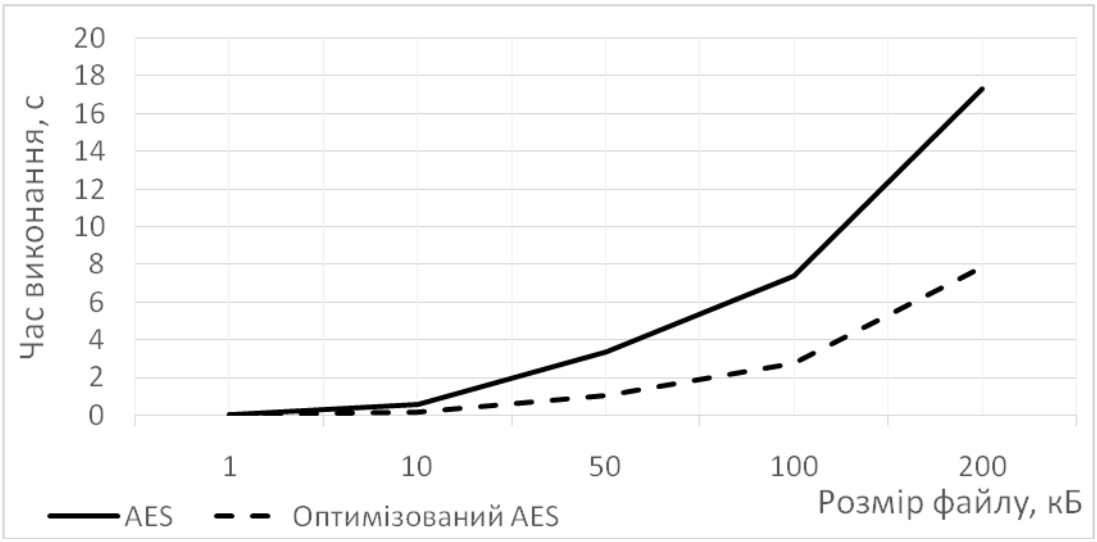

Рис. 6. Залежність часу виконання процедур дешифрування від розміру файлу для стандартного і оптимізованого алгоритмів AES.

Дослідження показали, що запровадження оптимізаційних процедур обробки даних дозволяє суттєво скоротити час шифрування та дешифрування у порівнянні з класичною реалізацією. Скорочення часу обробки залежить як від розміру ключа, так і від обсягу даних, які обробляються. Модифікована реалізація AES демонструє значний відсоток часової оптимізації – до 50% при шифруванні та до 75% при дешифруванні даних.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

В роботі запропонований новий метод оптимізації роботи алгоритму AES, який базується на об'єднанні ряду математичних операцій обробки даних, які мають схожий принцип опрацювання елементів, що дозволило скоротити час обробки даних при їх шифруванні та дешифруванні у порівнянні з відомою реалізацією. Програмна реалізація алгоритму AES, яка базувалась на офіційній специфікації стандарту шифрування, була успішно модифікована для зменшення часу обробки даних з умовою збереження криптографічної стійкості шифру. Загальні принципи запропонованого методу полягають у перетворенні усіх двовимірних масивів на одномірні, додаванні допоміжних таблиць для операцій ShiftRows та MixColumns, об'єднанні операцій зі схожими принципами опрацювання елементів. Проведені дослідження ефективності запропонованого методу удосконалення алгоритму. Модифікована реалізація алгоритму AES демонструє значний відсоток по скороченню часу обробки до 50% при шифруванні та до 75% при дешифруванні даних у порівнянні з відомими результатами.

Література

1. FIPS PUB 197. Specification for the Advanced Encryption Standard (AES), 2001.

2. Daoud L., Hussein F., Rafla N.. Optimization of Advanced Encryption Standard (AES) Using Vivado High Level Synthesis (HLS). EPiC Series in Computing. 2019. V.58, P.36-44.

3. Soltani A., Sharifian S.. An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA. *Microprocessors and Microsystems*. 2015. V.39, No.7, P.480-493.

4. Karthigaikumar P., Anitha Christy N., Siva Mangai N. M.. PSP CO2: an efficient hardware architecture for AES algorithm for high throughput. *Wireless Personal Communications*. 2015. V.85, No.1, P.305–323.

5. Taufik M., Amin D. E., Saifuddin M. A.. Hardware implementation and optimization of advanced encryption standard (AES) algorithm based on CCSDS. *AIP Conference Proceedings.* 2020. V.2226, P.060004

6. Liu Y., Xu X., Su H.. AES Algorithm Optimization and FPGA Implementation. *IOP Conf. Series: Earth and Environmental Science.* 2019. V.267, No.4, P. 042070.

7. Pavithra.S., Vaishnavi.M., Vinothini.M., Umadevi.V.. Optimization of aes algorithm using hardware and software. *International Journal of Scientific & Engineering Research*. 2015. V.6, No.4, P.965-971

8. Li M., Gan J., Cheng S., Hu X.,Yu Y., Li J.. Design of Lightweight AES Algorithm Based on Masked Lookup Table. *IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*. 2019. P.437-441.

9. James M., Kumar D. S.. An Optimized Parallel Mixcolumn and Subbytes design in Lightweight Advanced Encryption Standard. *International Journal of Computational Engineering Research (IJCER).* 2016. V.6, No.3, P. 25-28.

10. Горбенко І. Д., Тоцький О. С., Казьміна С. В. Перспективний блоковий шифр «Калина» – основні положення та специфікація // Прикладна радіоелектроніка. — Харківський національний університет радіоелектроніки, 2007.- т.6, №2. — 195-208 с.

11. Калинин Д.А., Козина Г.Л. Быстродействие шифров «Калина» и «AES». // Радіоелектроніка, інформатика, управління. 2013, №1, с.62-65.

12. Y. Khatri, R. Chhabra, N. Gupta, A. Khanna and D. Gupta, "Secure modified aes algorithm for static and mobile networks", *International Conference on Innovative Computing and Communications*, pp. 389-399, 2020.

13. Sadi Arman; Tanjila Rehnuma; Mahfuzur Rahman. Design and Implementation of a Modified AES Cryptography with Fast Key Generation Technique. 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), 26-27 December 2020, p.191-195.

14. Stallings W.. Cryptography and network security: principles and practice; vol. 6. Pearson Education; 2017.

15. Rijndael MixColumns. URL: https://en.wikipedia.org/wiki/Rijndael_MixColumns

16. Riyaldhi R., Kurniawan A.. Improvement of advanced encryption standard algorithm with shift row and S. box modification mapping in mix column. *Procedia computer science*. 2017. V.116, P.401-407.

References

1. FIPS PUB 197. Specification for the Advanced Encryption Standard (AES), 2001.

2. Daoud L., Hussein F., Rafla N.. Optimization of Advanced Encryption Standard (AES) Using Vivado High Level Synthesis (HLS). *EPiC Series in Computing*. 2019. V.58, P.36-44.

3. Soltani A., Sharifian S.. An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA. *Microprocessors and Microsystems*. 2015. V.39, No.7, P.480-493.

4. Karthigaikumar P., Anitha Christy N., Siva Mangai N. M.. PSP CO2: an efficient hardware architecture for AES algorithm for high throughput. *Wireless Personal Communications*. 2015. V.85, No.1, P.305–323.

5. Taufik M., Amin D. E., Saifuddin M. A.. Hardware implementation and optimization of advanced encryption standard (AES) algorithm based on CCSDS. *AIP Conference Proceedings.* 2020. V.2226, P.060004

6. Liu Y., Xu X., Su H.. AES Algorithm Optimization and FPGA Implementation. *IOP Conf. Series: Earth and Environmental Science.* 2019. V.267, No.4, P. 042070.

7. Pavithra.S., Vaishnavi.M., Vinothini.M., Umadevi.V.. Optimization of aes algorithm using hardware and software. *International Journal of Scientific & Engineering Research*. 2015. V.6, No.4, P.965-971

8. Li M., Gan J., Cheng S., Hu X.,Yu Y., Li J.. Design of Lightweight AES Algorithm Based on Masked Lookup Table. *IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*. 2019. P.437-441.

9. James M., Kumar D. S.. An Optimized Parallel Mixcolumn and Subbytes design in Lightweight Advanced Encryption Standard. *International Journal of Computational Engineering Research (IJCER).* 2016. V.6, No.3, P. 25-28.

10. Gorbenko I. D., Totsky O. S., Kazmina S. V. Promising block cipher "Kalina" - the main provisions of the specification // Applied Radio Electronics. – Kharkiv National University of Radio Electronics, 2007.- т.6, №2. — 195-208 с.

11. Kalinin D.A., Kozina G.L. The speed of the Kalina and AES ciphers. // Radioelectronics, informatics, management, 2013, №1, p.62-65.

12. Y. Khatri, R. Chhabra, N. Gupta, A. Khanna and D. Gupta, "Secure modified aes algorithm for static and mobile networks", *International Conference on Innovative Computing and Communications*, pp. 389-399, 2020.

13. Sadi Arman; Tanjila Rehnuma; Mahfuzur Rahman. Design and Implementation of a Modified AES Cryptography with Fast Key Generation Technique. 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), 26-27 December 2020, p.191-195.

14. Stallings W.. Cryptography and network security: principles and practice; vol. 6. Pearson Education; 2017.

15. Rijndael MixColumns. URL: https://en.wikipedia.org/wiki/Rijndael_MixColumns

16. Riyaldhi R., Kurniawan A.. Improvement of advanced encryption standard algorithm with shift row and S. box modification mapping in mix column. *Procedia computer science*. 2017. V.116, P.401-407.

Рецензія/Peer review : 26.06.2022 р. Надрукована/Printed :02.08.2022 р.