DOI: 10.18372/2410-7840.21.14266
УДК 004.056:061.68

ЕФЕКТИВНА ІМПЛЕМЕНТАЦІЯ ТА ПОРІВНЯННЯ ШВИДКОДІЇ ШИФРІВ «КАЛИНА» ТА ГОСТ 28147-89 ЗА ВИКОРИСТАННЯ ВЕКТОРНИХ РОЗШИРЕНЬ SSE, AVX ТА AVX-512

Ярослав Совин, Володимир Хома, Юрій Наконечний, Марта Стахів

Дуже важливою властивістю блокових шифрів є забезпечення високої продуктивності для широкого класу мікропроцесорних архітектур і, насамперед, для домінуючих x86-64 платформ. Недостатня швидкодія ДСТУ ГОСТ 28147:2009 на сучасних обчислювальних архітектурах загального призначення стала однією з причин проведення національного криптоконкурсу з обрання нового блокового шифру, в якому переміг алгоритм «Калина», швидкодія якого, за умовами конкурсу, повинна була бути не меншою, ніж швидкодія чинного державного стандарту шифрування. Щоб досягти високої швидкодії наявні реалізації шифру «Калина» використовують табличний одноблоковий підхід, який не позбавлений низки недоліків: не використовуються можливості сучасних процесорів щодо розпаралелювання виконання коду, векторизації обробки даних, вразливий до кеш-атак. У роботі запропоновано основні підходи до розроблення мультиблокових векторних реалізацій шифрів «Калина» та ГОСТ 28147-89, у тому числі стійких до кеш-атак, з використанням SIMD-інструкцій SSE, AVX/AVX2, AVX-512. Особливу увагу приділено виконанню операції нелінійної заміни, яка визначає швидкодію реалізації загалом. Проведено експериментальні дослідження, які довели ефективність запропонованих підходів щодо збільшення швидкодії та дали змогу визначити доцільність застосування відповідних векторних розширень у тому чи іншому випадку. Встановлено, що за максимально досяжною швидкодією векторні реалізації ГОСТ 28147-89 відчутно випереджають шифр «Калина». Використання запропонованих підходів дозволяє підвищити швидкодію вітчизняних програмних криптографічних засобів та безпеку їх функціонування.

Ключові слова: *шифр «Калина», шифр ГОСТ 28147-89, векторні розширення системи команд SSE, AVX, AVX-512, x86-64 архітектура, ефективна імплементація криптоалгоритмів, вимірювання швидкодії.*

1. Вступ

Блокові симетричні шифри (БСШ) є основним криптографічним засобом гарантування конфіденційності та цілісності даних під час їх обробки в інформаційно-телекомунікаційних системах. На сьогодні відомо багато алгоритмів БСШ із різними принципами побудови (мережа Фейстеля, SPN, ARX), параметрами (довжина ключа, блоку, число раундів), складністю імплементації, деякі з них затверджені в якості національних і міжнародних стандартів.

З огляду на постійно зростаючі обсяги та швидкості передачі даних у сучасних інформаційно-телекомунікаційних системах дуже важливою властивістю БСШ є забезпечення достатньо високої продуктивності шифрування для широкого класу мікропроцесорних архітектур і, насамперед, для домінуючих на ринку x86-64 платформ.

В Україні до 2015 року діяв стандарт ДСТУ ГОСТ 28147:2009 [7], що визначав як БСШ алгоритм ГОСТ 28147-89 з розміром блоку 64 біти та 32-ма раундами. Низка виявлених вразливостей, певна моральна застарілість та недостатня швидкодія ГОСТ 28147-89 на сучасних обчислювальних архітектурах загального призначення стали причиною оголошення Державною службою спеціального зв'язку та захисту інформації України відкритого національного конкурсу з обрання нового стандарту БСШ для поступової заміни ДСТУ ГОСТ 28147:2009. Головними вимогам до кандидатів були високий рівень криптографічної стійкості, підтримка 128-, 256- та 512-бітних розмірів блоків і довжин ключів, висока продуктивність у разі програмної реалізації, зокрема [5]:

− швидкодія криптоалгоритму повинна бути не меншою, ніж швидкодія чинного державного стандарту шифрування;

− криптоалгоритм повинен бути орієнтованим для реалізації на 32- або 64-розрядних процесорах;

− операції криптоалгоритму повинні мати ефективну програмну та апаратну реалізацію;

− давати змогу паралельного виконання декількох операцій (за можливості).

Переможцем став шифр «Калина», затверджений із невеликими модифікаціями як стандарт ДСТУ 7624:2014 і набувший чинності з 1 липня 2015 року [1].

Відомі реалізації шифрів «Калина» й ГОСТ 28147-89 для досягнення високої продуктивності замість покрокового виконання операцій використовують передобчислені таблиці, так звані T-таблиці, що реалізують операції раундового перетворення. Робота криптоалгоритму в такому випадку зводиться до пошуку в T-таблиці для кожного байту стану, який виступає індексом, відповідного

значення, їх накладанням між собою та раундовим ключем. Отже, за такого підходу задіяні регістри загального призначення (General Purpose Registers, GPR) та базові інструкції процесора.

Такий підхід не позбавлений низки недоліків:

1. Не повністю використовує можливості суперскалярності сучасних процесорних архітектур.

Суперскалярність передбачає збільшення продуктивності завдяки одночасній роботі декількох однакових функціональних блоків процесорного ядра (ALU, FPU, AGU і т. ін.), що незалежно виконують інструкції. У разі одноблокового шифрування, коли дані обробляються послідовно – блок за блоком, частина вузлів процесора буде незадіяна й замість паралельної роботи простоювати. Щоб сповна скористатися можливостями суперскалярності процесора потрібно застосувати мультиблокове шифрування, коли один програмний потік на кожному раунді обробляє декілька блоків. Таке розпаралелювання обробки даних можливе в тих режимах шифрування, де немає зворотного зв'язку між блоками, що обробляються (напр. ECB, CTR). Проте навіть для режимів, які мають зворотній зв'язок між блоками (напр. CFB) можна використати паралельну обробку декількох блоків від різних потоків шифрування. За такого універсального підходу швидкодію шифрування можна вимірювати в режимі ECB із екстраполюванням результатів на інші режими.

2. Не використовує можливості векторизації коду присутні в сучасних процесорах.

Векторні інструкції реалізують SIMD-технологію, коли однією інструкцією паралельно обробляється кілька одиниць даних (вектор). Це дозволяє опрацьовувати однією інструкцією всі байти стану шифру, а в певних випадках і декілька блоків (якщо довжина векторних регістрів більша розміру блоку). Сучасні мікропроцесори з архітектурою x86-64 підтримують кілька наборів векторних інструкцій: SSE, AVX/AVX2, AVX-512 з розміром вектору 128-, 256- та 512-біт відповідно.

Станом на сьогодні нам не відомі реалізації шифру «Калина» з використанням векторних розширень системи команд x86-процесорів.

3. Не забезпечує константний час виконання й, отже, є вразливим до кеш-атак.

Кеш-атаки є найпоширенішим видом програмних атак проти криптоалгоритмів через сторонні канали витоку інформації, які враховують різницю в часі доступу до даних, які присутні або відсутні в кеш-пам'яті. Оскільки кеш це спільний ресурс, який розділяється багатьма програмами, такі атаки не потребують фізичного доступу, а лише здатності виконувати певний код на цільовій системі [12].

У процесі виконання криптоалгоритму T-таблиці кешуються процесором, щоби зменшити час доступу. Раунд полягає в накладанні раундового ключа *key* на матрицю стану *state* і зчитування з T-таблиць відповідних значень *T[index]*, де *index= state^key*. Маніпулюючи кешем і вимірюючи час доступу до даних можливо визначити адреси комірок пам'яті, до яких йде звертання (*index*). Тоді знаючи *state* (напр., *state=plaintext* на початку шифрування) можна отримати інформацію про ключ *key*. Тому табличні реалізації є дуже вразливими до кеш-атак, хоча й забезпечують високу швидкодію.

Отже, є об'єктивна потреба в розробленні таких програмних реалізацій шифру «Калина», які б максимально використовували можливості сучасних мікропроцесорів щодо збільшення швидкодії внаслідок розпаралелювання як виконання коду (суперскалярність), так і обробки даних (векторизація) та за потреби були невразливі до кеш-атак (константний час).

2. Аналіз останніх досліджень і публікацій

У [14] наведено вихідний код і результати вимірювання швидкодії шифру «Калина» для одноблокової табличної реалізації на x86-64 процесорі Intel Core i5-4670 з тактовою частотою 3,40 ГГц. Для зниження впливу переключення контексту процесора блок пам'яті був багатократно перешифрований. Вимірювання швидкодії через шифрування однакового обсягу відкритих текстів (в режимі ECB) виконувалося для шифру «Калина» та ще кількох шифрів, зокрема ГОСТ 28147-89 в однакових умовах. Для отримання найбільшої швидкодії була обрана мова програмування C++, використаний компілятор gcc 4.9.2, тестування здійснювалося на ПК під управлінням 64-бітної ОС Linux (Ubuntu) [6].

Результати тестування швидкодії наведені в таблиці 1. Оскільки в роботі нами для вимірювання швидкодії використовуються такі одиниці як кількість тактів на шифрування одного байту (тактів/байт або cycles per byte, *cpb*), а в [14] як одиниці вимірювання використано Мбіт/с (Mb/s) ми перерахували їх у *cpb* для подальшого порівняння.

У кросплатформенній бібліотеці криптографічних примітивів «Шифр+» v2 [2] досягнуто показників швидкодії представлених у таблиці 1. Зазначимо, що особливістю даної бібліотеки є підтримка різних апаратних (x86-32, x86-64, ARM, ARM64 та ін.) і програмних (Windows, Linux, MacOS, Android, iOS) платформ, покращена підтримка багатопотоковості на рівні ОС, мов програмування та компіляторів. Також задекларовано

підтримку векторних розширень SSE/ AVX/ AVX2, проте без вказання конкретики для яких алгоритмів і як ця підтримка реалізована.

Вимірювання проводилися для x86-64 CPU Intel Core i7-6700HQ 2,6 ГГц з 16 Гбайт ОЗП у середовищі ОС Windows 10 шифруванням блоку 16 Кбайт під час 1 млн. повторів.

У документації криптобібліотеки cppcrypto [10] наведені результати вимірювання швидкодії представлені в таблиці 1 (для 64-бітної архітектури). Це компактна кросплатформена C++ бібліотека орієнтована на досягнення максимальної швидкодії. Підтримуються компілятори Visual C++, gcc і clang та архітектури як x86-64 так і x86-32.

Вимірювання швидкодії здійснювалося шифруванням файлу розміром 130 Кбайт 100000 разів у режимі CBC на процесорі Xeon E5-1650 v3 з тактовою частотою 3,50 ГГц.

Оскільки результати [2, 10] були представлені в Мбайт/с (MB/s), тому для зручності зіставлення їх також перераховано у *cpb*.

Таблиця 1

Швидкодія шифру «Калина» та ГОСТ 28147-89

Блоковий шифр	**Робота [14]**		**Бібліотека «Шифр+» v2 [2]**		**Компілятор VC++2015 [10]**		**Компілятор gcc 5.2 [10]**		**Компілятор clang 3.7 [10]**	
	Mb/s	**cpb**	**MB/s**	**cpb**	**MB/s**	**cpb**	**MB/s**	**cpb**	**MB/s**	**cpb**
Калина-128/128	2611,77	10,41	128,22	20,28	179	19,55	236	14,83	206	16,99
Калина-128/256	1809,70	15,03	97,93	26,55	132	26,52	175	20,00	148	23,65
Калина-256/256	2017,97	13,48	128,12	20,29	177	19,77	206	16,99	167	20,96
Калина-256/512	1560,89	17,43	111,64	23,29	140	25,00	135	25,93	131	26,72
Калина-512/512	1386,46	19,62	120,22	21,63	155	22,58	124	28,23	119	29,41
ГОСТ 28147-89	639,18	42,55	44,65	58,23	-	-	-	-	-	-

Отже, аналізуючи результати наведені в табл. 1, можна зазначити, що найвищу швидкодію 10,41 такти/байт забезпечує таблична реалізація Калини-128/128 представлена в [14]. Водночас вона в 4 рази випереджає ГОСТ 28147-89 зі швидкодією 42,55 такти/байт.

Розглянемо тепер відомі реалізації шифру ГОСТ 28147-89 для x86-64 архітектур.

У статті [3] описано мультиблокові реалізації ГОСТ 28147-89 з використанням, як табличного підходу на регістрах загального призначення, так і векторних інструкцій SSE/AVX. Реалізація алгоритму ГОСТ 28147–89 з допомогою AVX-інструкцій дає змогу отримати швидкодію 8,5 тактів/байт, а для архітектури Haswell з підтримкою AVX2 продуктивність зростає до 6,7 тактів/байт.

Також у [3] відзначено, що за умови застосування мультиблокового шифрування на регістрах загального призначення з використанням 8-бітних передобчислених S-box таблиць швидкодія ГОСТ 28147–89 зростає із 60 тактів/байт (1 блок) до 30 тактів/байт (2 блоки).

У [4] наведено результати щодо швидкодії ГОСТ 28147-89 під час шифрування на регістрах загального призначення (**Poly GPR**), з використанням табличного підходу для 8-бітних передобчислених S-box таблиць (**Table 4Kb**), на базі векторних SSE2-інструкцій (**Poly SSE2**) та SSSE3/AVX-інструкцій (**SSSE3/AVX**), які представлені в таблиці 2.

Таблиця 2

Швидкодія ГОСТ 28147-89
за різних способів реалізації [4]

Реалізація	**Швидкодія**	
	MB/s	**cpb**
Poly GPR	31	83,87
Table 4Kb	52	50,00
Poly SSE2	117	22,22
SSSE3/AVX	375	6,93

Основні підходи до реалізації ГОСТ 28147-89 на базі SSE/AVX-інструкцій описані в публікаціях [8, 9]. Ключовою в збільшенні швидкодії є інструкція перемішування даних у 128-бітному векторному регістрі *pshufb (vpshufb* – 256-бітна AVX-версія*)*, яка копіює байти з регістра-джерела в регістр-приймач у порядку, описаному в індексному регістрі. Це дозволяє легко реалізувати 4-бітні вузли заміни. Слід зазначити, що ця інструкція реалізує тільки 4-бітні вузли заміни, бо використовує лише молодші 4 біти кожного байту індексного регістру. У разі певного комбінування вузлів заміни та вхідних даних у векторних регістрах однією інструкцією *pshufb* можна пропустити дані через два S-блоки, а для 8 S-блоків потрібно виконати 4 такі інструкції відповідно.

Отже, ми бачимо, що використання для ГОСТ 28147-89 мультиблокового підходу в поєднанні з векторними SSE/AVX-інструкціями дало змогу збільшити швидкодію з 42,55 до 6,7 тактів/байт.

3. Постановка завдання

Метою статті є:

1) уперше представити реалізацію шифру «Калина» з використанням векторних розширень системи команд SSE-128, AVX-256, AVX-512;

2) оцінити та порівняти швидкодію реалізацій шифрів «Калина» та ГОСТ 28147-89 для процесорів x86-64 з використанням ресурсів суперскалярності й паралелізму у вигляді мультиблокового шифрування та SIMD-технології на основі векторних розширень системи команд SSE-128, AVX-256, AVX-512;

3) розглянути як табличні реалізації, вразливі до кеш-атак, так і стійкі до кеш-атак реалізації з константним часом виконання.

4. Основні параметри й операції шифру «Калина»

Національний стандарт шифрування ДСТУ 7624:2014 «Калина» [1] належить до SPN, байт-орієнтованих шифрів. Основні параметри шифру, такі як довжина ключа *k* і блоку даних *l*, число раундів *t* та число стовпців матриці стану *c* пов'язані залежностями представленими в таблиці 3. Довжина блоку і ключа використовуються в позначенні шифру у форматі Калина-*l*/*k*.

Таблиця 3

Основні параметри шифру «Калина»

Довжина ключа *k*, біт	**Довжина блоку *l*, біт**	**Кількість раундів *t***	**Кількість стовпців матриці стану *c***
128, 256	128	10	2
256, 512	256	14	4
512	512	18	8

Під час шифрування операції виконуються над двовимірним масивом байт, названим поточним станом шифру (*State)*. Поточний стан шифру можна представити у вигляді матриці розмірністю $8 \times c$ байтів (вісім рядків по *c* байт): $State = (s_{i,j})$, де $i = 0...7, j = 0...c-1$.

Структура алгоритму «Калина» представлена на рис. 1. В алгоритмі використовуються операції арифметичного додавання ($\boxplus$) та віднімання ($\boxminus$) за модулем 2^{64}, додавання за модулем 2 ($\oplus$), нелінійної заміни (***SubBytes***, ***InvSubBytes***), циклічного зсуву рядків (***ShiftRows***, ***InvShift Rows***) та лінійного перетворення (***MixColumns***, ***InvMixColumns***).

Операції додавання та віднімання реалізують арифметичне додавання або віднімання стовпців матриці стану *State* і стовпців циклового підключа за модулем 2^{64}. Числа в стовпцях вважаються представленими у форматі little-endian.

Операція додавання за модулем 2 виконується над матрицею стану *State* і цикловим підключем.

Операції ***SubBytes*** та ***InvSubBytes*** виконують підстановку кожного байту матриці стану на відповідний йому байт з однієї з чотирьох таблиць заміни *S0-S3* та *inv_S0-inv_S3* для операцій зашифрування й розшифрування відповідно. Кожна таблиця має розмір 256 байт. Номер таблиці заміни визначається як індекс рядку байту за модулем 4: $s_{i,j} = S_{i \bmod 4}(s_{i,j})$ або $s_{i,j} = inv\_S_{i \bmod 4}(s_{i,j})$.

Операції ***ShiftRows*** та ***InvShiftRows*** циклічно зсувають байти рядків вправо чи вліво відповідно. Число позицій δ_i, на яку зсувається рядок, залежить від номера рядку *i* та довжини блоку *l* і обчислюється за формулою: $\delta_i = \left[(i \cdot l)/512\right]$.

Операції ***MixColumns*** та ***InvMixColumns*** перетворюють стовпці матриці стану шляхом виконання операцій множення й додавання в скінченному полі *GF(2⁸)* за модулем незвідного многочлена $\psi = x^8 + x^4 + x^3 + x^2 + 1 = 0x11d$.

Кожен елемент результуючої матриці стану $W = (w_{i,j})$ обчислюється в полі *GF(2⁸)* як скалярний добуток рядка матриці *v (inv_v)* на стовпець матриці стану *State* відповідно до формул:

$$w_{i,j} = (v >>> i) \otimes S_j,$$

$$w_{i,j} = (inv\_v <<< i) \otimes S_j,$$

де *v = (0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04), inv_v = (0xAD, 0x95, 0x76, 0xA8, 0x2F, 0x49, 0xD7, 0xCA*); S_j – *j*-й стовпець матриці стану *State*; $v >>> i$ та $v <<< i$ – операції циклічного зсуву байт вектора *v* вправо і вліво на *i* позицій відповідно.

Для одержання раундових підключів із вихідного майстер-ключа використовується процедура розгортання ключа, в якій задіяно операції розглянуті вище.

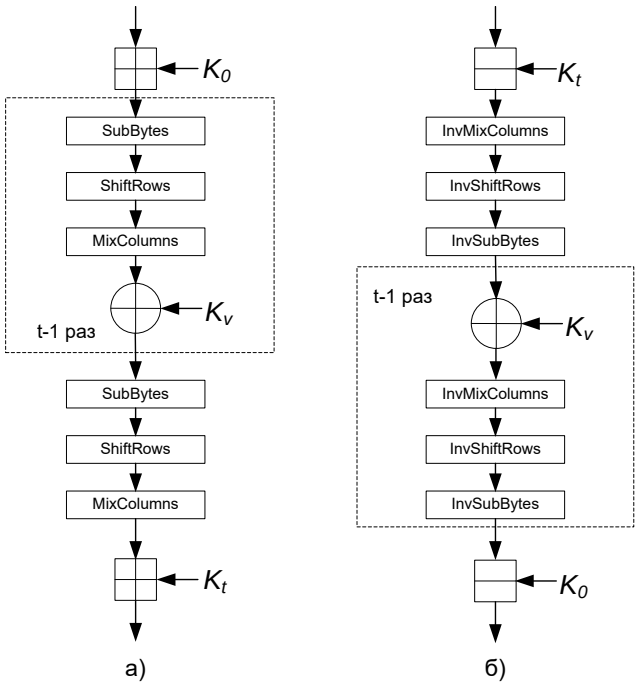

Рис. 1. Структурна схема шифру «Калина» в режимах зашифрування (а) і розшифрування (б)

5. Основні параметри й операції шифру ГОСТ 28147-89

Алгоритм ГОСТ 28147-89 шифрує дані поблоково, з розміром блоку 64 біти, довжина ключа становить 256 біт [7]. Алгоритм має структуру мережі Фейстеля з 32-х раундів та використовує операції додавання за модулем 2^{32}, додавання за модулем 2, нелінійної заміни *S* та циклічного зсуву, які легко реалізуються в мікропроцесорах завдяки підтримці на рівні системи команд.

Структурна схема раунду алгоритму ГОСТ 28147-89 наведена на рис. 2.

Перед початком шифрування 64-бітне повідомлення заноситься в 32-бітні регістри $L_0 \| R_0$.

Алгоритм шифрування складається з 32 раундів. У кожному *i*-му раунді вміст регістру R_i арифметично додається з 32-бітним підключем K_i. Результат підсумовування перетворюється в блоці підстановки *S* і циклічно зсувається на 11 розрядів вліво. Результат раундової функції $F(K_i, R_i)$ додається за модулем 2 з 32-бітним вмістом регістру L_i. Отриманий результат записується в R_{i+1} , водночас попереднє заповнення регістру R_i переписується в L_{i+1}:

$$L_{i+1} = R_i,$$

$$R_{i+1} = L_i \oplus (S(K_i + R_i \bmod 2^{32}) <<< 11),$$

де $\oplus$ – позначає побітову суму за модулем 2, <<< 11 – циклічний зсув вліво на 11 розрядів.

В останньому раунді обмін не здійснюється, тобто $R_{32} = R_{31}$ та $L_{32} = L_{31} \oplus (S(K_{31} + R_{31} \bmod 2^{32}) <<< 11)$.

В алгоритмі ГОСТ 28147-89 ключ *K* довжиною 256 біт розглядається як вісім 32-бітних підключів: $K = K_0 \| K_1 \| K_2 \| K_3 \| K_4 \| K_5 \| K_6 \| K_7$. У раундах зашифрування з номерами $0 \le r \le 23$ раундовий підключ K_i визначається як $K_i = K_{(r \bmod 8)}$, а для останніх вісьмох раундів $24 \le r \le 31$ як $K_i = K_{7-(r \bmod 8)}$. Під час розшифрування порядок підключів зворотний.

Блок підстановки *S* складається з восьми вузлів заміни *S1-S8* з пам'яттю на 64 біти кожен. Вузол заміни представляє собою таблицю з 16 елементів, по 4 біти кожен. На вхід блоку підстановки поступає 32-бітний вектор, який розбивається на вісім послідовних 4-бітних векторів, кожен із яких перетворюється в 4-бітний вектор відповідним вузлом заміни. Вхідний вектор визначає адресу елементу всередині вузла заміни, вміст елементу є вихідним вектором.

6. Векторні розширення системи команд x86-64 процесорів

Сучасні мікропроцесори з архітектурою x86-64 підтримують кілька наборів векторних інструкцій: SSE, AVX/AVX2, AVX-512 [13]. Коротко проаналізуємо їх можливості та доцільність використання в контексті статті.

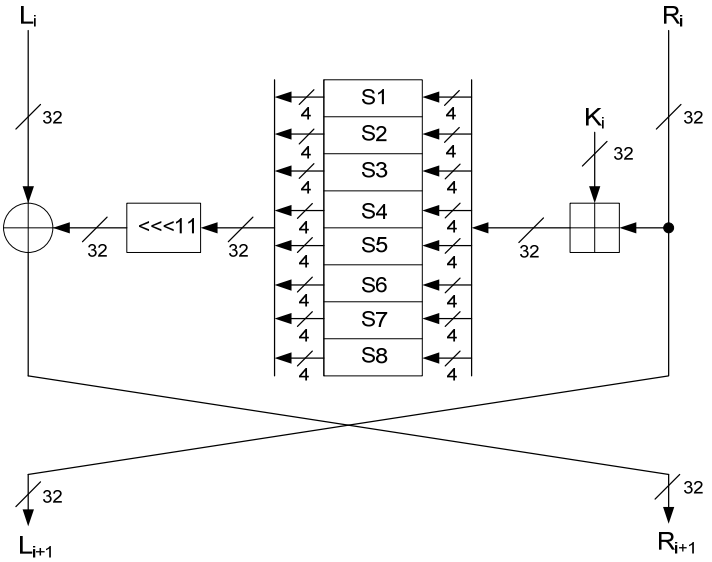

Рис. 2. Структурна схема раунду алгоритму ГОСТ 28147-89

SSE (Streaming SIMD Extensions) – це набір SIMD-інструкцій, вперше представлений у процесорах Pentium III. SSE додає в архітектуру процесора вісім 128-бітних регістрів *xmm0-xmm7* та понад 70 інструкцій для обробки 32-бітних даних типу *float*. У подальшому SSE-технологія доповнилася новими розширеннями, такими як SSE2, SSE3, SSSE3 та SSE4 (у складі SSE4.1 і SSE4.2), які додавали нові інструкції, що значно збільшило її ефективність. З появою 64-бітних процесорів число векторних регістрів у них було збільшено з 8 до 16 (*xmm0-xmm15*). Надалі під SSE чи SSE-128 ми будемо розуміти всі 128-бітні розширення SSE, SSE2, SSE3, SSSE3 та SSE4.

Варто зазначити деякі обмеження SSE-розширень під час реалізації шифру «Калина». SSE-технологія не дозволяє обробляти декілька блоків шифру одночасно, оскільки мінімальний розмір блоку в «Калині» становить 128 біт і регістри *xmm* теж 128-бітні. Також у SSE відсутні команди індексного зчитування даних із пам'яті *gather*, необхідні для реалізації табличного підходу чи операції ***SubBytes***. SSE-інструкції використовують двоопераднний формат ($a = a + b$), за якого вміст одного з операндів втрачається, що вимагає додаткових пересилок даних.

Перевагою SSE-технології є те, що вона підтримується фактично всіма актуальними на сьогодні x86-64 процесорами, оскільки останнє розширення SSE4 з'явилося в процесорах із 2008 року.

AVX (Advanced Vector Extensions) – розширення системи команд x86-мікропроцесорів, запропоноване Intel у 2008 р. та доступне в процесорах із 2011 року. AVX надає різні удосконалення, зокрема нові інструкції. Розширення AVX2 це подальший розвиток AVX для роботи з цілочисельними операндами, реалізоване в процесорах починаючи з 2013 року. Надалі під позначенням AVX чи AVX-256 ми будемо розуміти всі 256-бітні векторні розширення AVX та AVX2.

У AVX розрядність шістнадцяти SIMD-регістрів збільшується з 128 (*xmm*) до 256 біт (*ymm0-ymm15*), з'являються нові інструкції, з яких у нашому контексті варто виділити команди *vpgather* для завантаження цілих чисел із пам'яті на підставі індексів у векторному регістрі, вводиться неруйнівний формат інструкцій ($c = a + b$), послаблюються вимоги до вирівнювання операндів у пам'яті.

AVX-технологія підтримується більшістю масових процесорів, забезпечує високий рівень паралелізму, надає багатий вибір інструкцій тому розглядається нами в якості основної для реалізації шифру «Калина».

AVX-512 є подальшим розширенням 256-бітних інструкцій AVX запропонованим Intel у 2013 році. Число регістрів збільшується до 32 (*zmm0-zmm31*), а їхня розрядність із 256 до 512 біт, також додано багато нових інструкцій та зросли можливості наявних.

AVX-512 це узагальнена назва для багатьох розширень, які не всі повинні підтримуватися процесорами, що їх впроваджують, за винятком AVX-512F (Foundation). Важливими розширеннями в контексті реалізації криптоалгоритмів є AVX-512 BW (Byte and Word Instructions) – для операцій із 8- та 16-бітними цілими числами і AVX-512 VBMI (Vector Byte Manipulation Instructions) – додає інструкції байтових перестановок *permute*, зокрема *vpermi2b/vpermt2b*, що дають змогу дуже ефективно реалізувати операцію нелінійної заміни ***SubBytes***.

Поки що порівняно незначний відсоток процесорів підтримує розширення AVX-512 (особливо VBMI), що почали з'являтися в процесорах лише з 2017 року. Тому технологія AVX-512 розглядається нами наразі для оцінювання потенційної швидкодії вітчизняних криптоалгоритмів.

7. Інструменти, апаратні засоби й методика вимірювань

Для досягнення найбільшої швидкодії всі реалізації «Калина» і ГОСТ 28147-89 написано на мові C++ з використанням компіляторів із комплекту Microsoft Visual Studio 2019 (надалі MSVC) та gcc 7.3.0 (надалі GCC). Компіляція проводилася за умови максимальної оптимізації з параметрами */O2* (Maximize Speed) та *–O3* (Optimize fully for speed) для MSVC і GCC відповідно.

Для векторних інструкцій використовувалися intrinsic-функції [11], які представляють собою високорівневу С-обгортку навколо асемблерних команд. Для збільшення швидкодії раундові операції представлялися макросами та здійснювалося розгортання циклів. Під час шифрування використовувалися попередньо обчислені раундові підключі.

Програмна платформа: ОС Microsoft Windows Server (10.0) 64-bit на Google Cloud Platform.

Апаратна платформа: x86-64 процесор Intel Xeon Skylake-SP 2,0 ГГц (*machine types = n1-highcpu-2*) на Google Cloud Platform з підтримкою векторних розширень: SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AVX2, AVX512F, AVX512DQ, AVX512BW, AVX512VL.

Одиниця вимірювання швидкодії: число тактів на шифрування одного байту, що дозволяє коректно зіставляти швидкодію CPU з різними тактовими частотами.

Вимірювання числа тактів здійснювалося з допомогою інструкції *rdtsc* для зчитування лічильника TSC (Time Stamp Counter) відповідно до методики описаної в [15].

Для зниження впливу переключення контексту процесора виконувалося багатократне шифрування (100 млн. повторів), за результат вимірювання приймалося мінімальне значення.

Параметри шифру «Калина»: розмір блоку й довжина ключа 128 біт (Калина-128/128), оскільки такі параметри забезпечують найбільшу швидкодію. Вимірювання швидкодії для обох шифрів велося в режимі простої заміни (ECB).

Табличні реалізації шифрів, які є вразливі до кеш-атак, надалі позначимо (**Table-based, Not Constant Time**), а стійкі до кеш-атак реалізації з програмним виконанням операцій шифру позначимо (**Software-based, Constant Time**). У програмних реалізаціях відбувається покрокове виконання з допомогою векторних інструкцій операцій криптоалгоритму, що забезпечує час виконання, який не залежить від значень оброблюваних даних.

8. Імплементація ГОСТ 28147-89

8.1. Табличні реалізації ГОСТ 28147-89

Для зменшення звертань до пам'яті та операцій обчислення адреси два сусідні 4-бітні вузли заміни (рис. 3.а) можна об'єднати в один 8-бітний (**8-bit Sbox**) розміром 256 елементів (рис. 3.б). З огляду на те, що наступною операцією є циклічний зсув на 11 розрядів вліво, то в таблицях доцільно зберігати 32-бітні елементи вже зсунуті циклічно вліво. Такий підхід вимагає чотирьох таблиць *T1-T4* розміром 256×4 байт (1 Кбайт) кожна.

Подальшим кроком для зменшення кількості операцій є об'єднання трьох 4-бітних вузлів заміни в один 12-бітний (**12-bit Sbox**), розміром 4096 елементів (рис. 3.в). За такої умови будемо мати дві таблиці *T1-T2,* отримані об'єднанням 3 вузлів та одну 8-бітну таблицю *T3* отриману об'єднанням 2 вузлів розміром 256 елементів. Такий підхід вимагає двох таблиць *T1-T2* розміром 4096 ×4 байт (16 Кбайт) кожна та однієї *T3* розміром 256×4 байт (1 Кбайт). Загальний розмір таблиць буде становити 33 Кбайт і вони здебільшого попадають у кеш даних L1 розміром 32 Кбайт.

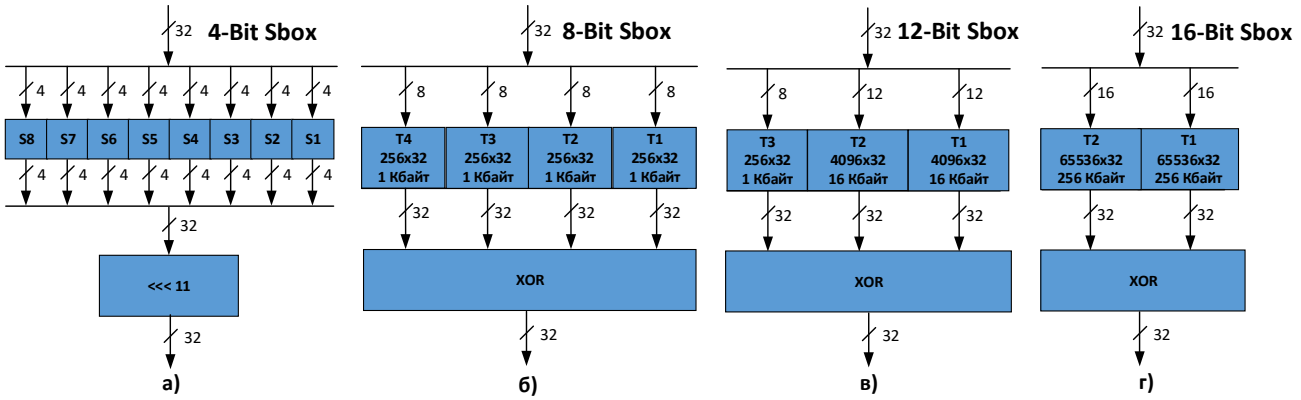

Рис. 3. Перехід від 4-бітних вузлів (а) заміни до 8-бітних (б), 12-бітних (в) й 16-бітних (г)

Останнім кроком для зменшення кількості операцій є об'єднання чотирьох 4-бітних вузлів заміни в один 16-бітний (**16-bit Sbox**), розміром 65536 елементів (рис. 3.г). Такий підхід вимагає двох таблиць *T1-T2* розміром 65536×4 байт (256 Кбайт) кожна. Загальний розмір таблиць буде становити 512 Кбайт і вони не поміщаються в кеш даних L1, що теоретично може призводити до падіння швидкодії, якщо зменшення звертань до пам'яті з 3-х до 2-х не компенсується довшим часом доступу до кешу наступного рівня L2.

Щоби мати базу для порівняння швидкодії нами реалізовано мультиблокове шифрування з використанням регістрів загального призначення (**GPR**) без застосування векторних інструкцій та з використанням AVX-256 і AVX-512 інструкцій для 8-, 12- і 16-бітних таблиць.

У AVX-256 і AVX-512 табличні методи реалізуються завдяки появі інструкцій *vpgather*. Відповідні intrinsic-функції *_mm256_i32gather_epi32 (int const* base_addr, __m256i vindex, const int scale)* та *_mm512_i32gather_epi32(__m512i vindex, void const* base_addr, int scale)* дозволяють прочитати 8 (AVX-256) або 16 (AVX-512) 32-бітних значень із масиву пам'яті заданим адресою *base_addr* (Т-таблиці) на основі 32-бітних індексів у векторному регістрі *vindex* (зберігає регістри R_i) промасштабованих на *scale* (4) байт.

Оскільки в ГОСТ 28147-89 обробка ведеться над 32-бітними частинками L_i і R_i, то інструкцією *vpgather* в одному *ymm*-регістрі можна одночасно опрацьовувати 8, а *zmm*-регістрі – 16 блоків R_i.

Слід зазначити, що *vpgather* досить повільні інструкції і для процесорів Skylake мають Latency на рівні 22 тактів, а Throughput біля 5 тактів (AVX-256).

У таблиці 4 представлено результати вимірювання швидкодії за різної кількості блоків, які одночасно шифруються в межах раунду, від 1 (**1-way**) до 64 (**64-way**), де зеленим кольором відзначено найкращі результати для кожної реалізації.

Для GPR-варіанту зауважимо, що загалом 12-бітні Т-таблиці дають дещо кращі результати за 8-бітні. Для одного блоку (1-way) внаслідок сучаснішої архітектури CPU вдалося досягти *cpb* = 35,00 проти 42,55 в [14]. 16-бітні Т-таблиці, як і очікувалося, поступаються у швидкодії 8- і 12-бітним. Наявність у мікроархітектурі Skylake двох портів завантаження даних із пропускною здатністю 512 біт/такт та трьох AGU (Address Generation Unit) дозволяє отримувати відчутний приріст швидкодії в разі збільшення числа блоків до 3.

Загалом AVX-реалізації випереджають відповідні GPR-реалізації, а найкращі результати теж спостерігаються для варіанту **12-bit Sbox**. Перехід від AVX-256 до AVX-512 не дає суттєвого збільшення швидкодії. Два порти завантаження даних забезпечують відчутне зростання швидкодії за умови одночасного виконання до двох інструкцій *vpgather*, що відповідає 16 і 32 оброблюваним блокам для AVX-256 та AVX-512.

За даними табл. 4 найвища швидкодія ГОСТ 28147-89 у 6,45 тактів/байт досягається під час шифрування 64 блоків AVX-512 інструкціями для варіанту **12-bit Sbox**.

8.2. Програмні реалізації ГОСТ 28147-89

Програмні реалізації ГОСТ 28147-89 базуються на підході представленому в [8, 9], який передбачає виконання чотирьох інструкцій *shufb* для 4-бітних вузлів заміни. Зауважимо, що у AVX-512 з'явилася інструкція циклічного зсуву *rol*, яка використовується для зсуву на 11 біт, а у SSE-128 та AVX-256 циклічний зсув повинен реалізуватися командами зсуву вліво *slli* і вправо *srli*:

AVX-256	**AVX-512**

`tmp = _mm256_slli_epi32(sum, 11);`
`sum = _mm256_srli_epi32(sum, 21);`
`sum = _mm256_xor_si256(sum, tmp0);`
	`sum = _mm512_rol_epi32(sum, 11);`

У таблиці 5 представлено результати вимірювання швидкодії за різної кількості блоків, які одночасно шифруються в межах раунду, де зеленим кольором відзначено найкращі результати для кожної реалізації, а червоним – найкращий серед усіх реалізацій ГОСТ 28147-89.

Зазначимо, що SSE-128 реалізація є швидшою за табличні GPR-варіанти (*cpb* = 7,66 проти 10,90). Завдяки простоті раундової функції ГОСТ, яка використовує швидкі SIMD-інструкції (логічні, зсуву, перемішування), векторні програмні реалізації значно випереджають відповідні табличні (*cpb* = 3,39 проти 6,66 для AVX-256 та 1,88 проти 6,45 для AVX-512). Перехід від 256-бітних регістрів до 512-бітних забезпечує майже двократне зростання швидкодії (1,88 проти 3,39).

Таблиця 4

Табличні реалізації шифру ГОСТ 28147-89 (Table-based, Not Constant Time)

К-ть блоків	8-bit Sbox		12-bit Sbox		16-bit Sbox	
	GCC, cpb	MSVC, cpb	GCC, cpb	MSVC, cpb	GCC, cpb	MSVC, cpb
GPR						
1-way	35,00	35,25	38,25	37,00	73,50	72,00
2-way	18,63	18,63	20,13	19,00	36,88	35,25
3-way	13,50	14,83	14,58	13,75	24,58	24,00
4-way	12,19	15,94	11,75	12,06	18,88	18,75
5-way	12,20	18,95	10,90	12,45	15,30	15,05
AVX-256						
1-way	91,25	91,75	106,25	104,50	127,25	124,75
2-way	46,13	47,25	54,50	57,75	67,13	65,88
4-way	23,69	24,63	28,63	30,88	36,31	37,25
8-way	15,53	16,03	15,09	16,03	20,69	20,75
16-way	8,70	9,11	8,23	8,42	11,30	11,16
24-way	8,21	8,56	6,95	7,02	8,90	8,97
32-way	8,45	8,86	6,99	6,66	8,79	8,52
AVX-512						
1-way	129,25	130,25	131,75	138,25	139,25	141,25
2-way	69,75	71,63	71,50	74,50	77,50	79,38
4-way	35,19	38,00	36,69	39,06	43,81	47,38
8-way	20,78	21,56	19,47	21,41	25,13	27,59
16-way	12,73	12,48	11,16	11,39	15,03	15,53
32-way	8,01	8,04	6,72	7,05	9,24	9,38
48-way	7,95	8,04	6,60	6,80	7,72	7,86
64-way	7,93	8,07	6,45	6,61	7,06	7,17

Таблиця 5

Програмні реалізації шифру ГОСТ 28147-89 (Software-based, Constant Time)

К-ть блоків	SSE-128		AVX-256		AVX-512	
	GCC, cpb	MSVC, cpb	GCC, cpb	MSVC, cpb	GCC, cpb	MSVC, cpb
1-way	68,50	52,25	54,00	55,00	68,25	70,00
2-way	34,50	26,25	27,63	27,88	35,88	36,50
4-way	13,25	10,75	13,88	14,13	18,19	21,13
8-way	9,50	7,66	6,06	6,03	9,22	9,84
12-way	9,90	8,83	-	-	-	-
16-way	9,77	8,58	3,73	3,94	3,42	3,42
24-way	-	-	3,57	3,93	-	-
32-way	-	-	3,39	3,70	2,24	2,17
48-way	-	-	-	-	1,95	2,07
64-way	-	-	-	-	1,88	2,28

9. Імплементація «Калини»

10. Табличні реалізації «Калини»

Табличний підхід вибраний розробниками шифру «Калина» [14] та криптобібліотеками [2, 10] для досягнення високої швидкодії. У цьому випадку використовуються 8 таблиць *T1-T8*, кожна розміром 256×8 байт (2 Кбайти), а загальний розмір таблиць становить 16 Кбайт. Індексами в таблицях виступають відповідні елементи матриці стану *State*.

У AVX-256/512 табличні методи теж реалізуються завдяки інструкції *vpgather* і відповідній intrinsic-функції, але вже для зчитування 64-бітних значень: *_mm256_i64gather _epi64* та *_mm512_i64 gather_epi64*.

Для Калини-128/128 матриця стану займає 16 байт, що дозволяє одночасно розміщувати й обробляти в *ymm*-регістрі два блоки даних (AVX-256), а в *zmm*-регістрі – 4 блоки (AVX-512).

У таблиці 6 представлено результати вимірювання швидкодії за різної кількості блоків, які одночасно шифруються в межах раунду, де зеленим кольором відзначено найкращі результати для кожної реалізації, а червоним – найкращий результат серед усіх реалізацій «Калини».

Таблиця 6

Табличні реалізації шифру «Калини» (Table-based, Not Constant Time)

К-ть блоків	GPR				AVX-256				AVX-512			
	GCC, cpb		MSVC, cpb		GCC, cpb		MSVC, cpb		GCC, cpb		MSVC, cpb	
	ENC	DEC	ENC	DEC	ENC	DEC	ENC	DEC	ENC	DEC	ENC	DEC
1-way	9,00	11,25	9,00	10,88	19,50	22,38	19,50	22,00	23,63	26,88	25,38	28,25
2-way	7,75	8,81	7,69	9,00	10,88	12,69	10,63	12,44	13,69	15,94	14,75	16,88
4-way	7,88	8,97	11,06	12,66	9,22	10,41	8,72	9,94	8,25	9,97	8,84	10,59
6-way	-	-	-	-	9,25	10,38	8,52	9,73	7,67	8,81	7,81	9,25
8-way	-	-	-	-	9,34	10,69	8,42	10,88	5,64	6,95	5,83	7,30
12-way	-	-	-	-	-	-	-	-	5,47	6,84	5,52	7,52
16-way	-	-	-	-	-	-	-	-	5,61	7,09	5,49	7,78

У разі використання AVX-512 таблична реалізація шифру «Калина» випереджає за швидкодією всі табличні реалізації ГОСТ 28147-89 в режимі зашифрування, проте незначно (*cpb* = 5,47 проти 6,45) і дещо поступається в режимі розшифрування (*cpb* = 6,84 проти 6,45).

Використання GPR-регістрів теж забезпечує шифру «Калина» кращі результати порівнюючи з відповідними реалізаціями ГОСТ 28147-89: 7,69/8,81 тактів/байт на зашифрування/розшифрування проти 10,90. Але використання мультиблокового підходу дозволяє компенсувати майже чотирикратний програш у швидкодії ГОСТ 28147-89, відзначений в [14].

Що стосується AVX-256 реалізацій, то тут «Калина» поступається у швидкодії як ГОСТ 28147-89 (8,42/9,73 проти 6,66 тактів/байт), так і GPR-варіанту шифру «Калина».

Отже, застосування для шифру «Калина» векторних розширень у поєднанні з табличною реалізацією доцільне лише за використання технології AVX-512.

9.2. Програмні реалізації шифру «Калини»

Розглянемо основні підходи до виконання раундових перетворень шифру «Калина» з використанням векторних інструкцій. Водночас оскільки технологія AVX-256 розглядається як базова, то здебільшого саме для неї будуть наведені відповідні фрагменти коду, які з незначними модифікаціями переносяться на SSE-128 та AVX-512 інструкції.

Накладання циклового підключа. Операції арифметичного додавання та віднімання за модулем 2^{64}, а також додавання за модулем 2 з цикловими підключами легко реалізуються відповідними векторними інструкціями представленими такими intrinsic-функціями: *_mm256_add_ epi64*, *_mm256_sub_epi64*, *_mm256_xor_si256*.

SubBytes* та *InvSubBytes. Ці операції є найважчими в реалізації та загалом визначають швидкодію алгоритму, оскільки для них не має відповідних готових інструкцій як у випадку 4-бітних вузлів заміни ГОСТ 28147-89. Використання інструкцій *vpgather* не допустиме, бо за такої умови відбуваються звертання за адресами пам'яті, які залежать від даних і реалізація стає вразливою до кеш-атак. Оскільки реалізація ***SubBytes*** на рівні коду ідентична ***InvSubBytes***, надалі будемо використовувати лише назву операції ***SubBytes***.

Нами розроблена реалізація цих операцій з імунітетом до кеш-атак, яка полягає в покроковому зчитуванні рядків таблиці заміни та виконанні операції підстановки в певному діапазоні індексів. За цих умов рядки таблиці завжди зчитуються з пам'яті в однаковому порядку й незалежно від даних, які обробляються, тому не створюють сторонній канал витоку інформації.

Основою ***SubBytes*** є intrinsic-функція *_mm256_shuffle_epi8(a, b)*. Вона здійснює заміни в межах 16-байтних фрагментів *lane0-lane1*, де 4 молодших біти кожного байту в *b* визначають індекс елемента *a* на який буде проведена заміна. Якщо старший 7-й біт байту *b* рівний 1, тоді елемент замінюється 0.

SubBytes відбувається за 16 кроків на кожному з яких зчитується черговий *i*-й рядок таблиці заміни *sb_row[i]* та обробляється 16 індексів (рис. 4).

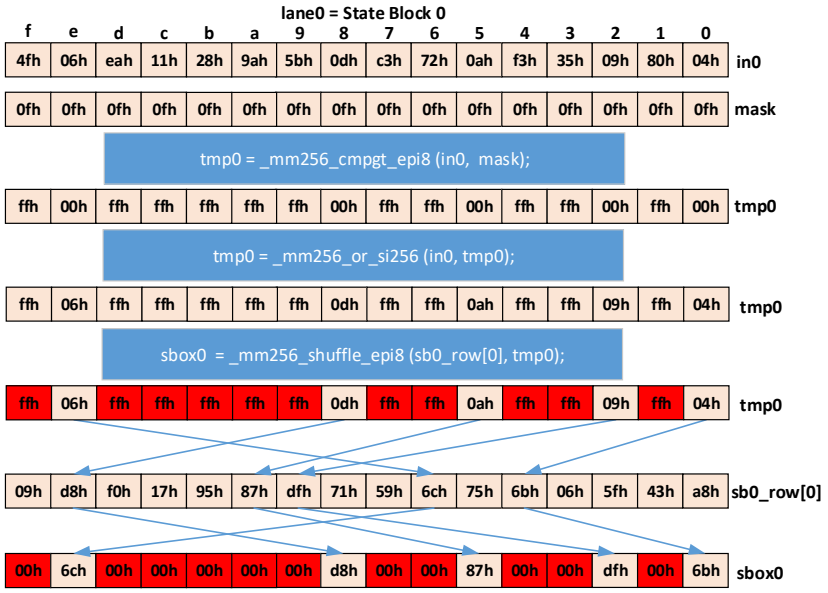

Рис. 4. Підстановка в межах одного 16-байтного рядку таблиці *SBox0*

На першому кроці в регістрі *in0* заміняються елементи зі значеннями від 0 до 15, на другому від 16 до 31 і т. д. Щоби виділити в регістрі *in0* елементи з кроком 16 використовується інструкція порівняння з умовою більше-ніж (greater-than) *_mm256_cmpgt_epi8(in0, mask)*, де регістр *mask* містить байти зі значенням 15. Елементи менші та рівні 15 заміняються в *tmp0* значеннями 0x00, а більші – значенням 0xff.

Далі отриманий результат порівняння накладається на вхідний регістр *in0* і виконується підстановка командою *shuffle* (рис. 4), де регістр *sbox0* зберігає результат *sbox0 =SBox0(in0)*. Ця дія повторюється для кожної з чотирьох таблиць заміни Після цього значення кожного байту регістру *in0* зменшується на 16 операцією *_mm256_sub_epi8(in0, step)*, щоб перейти до обробки наступних індексів.

Оскільки є 4 таблиці заміни *SBox0-SBox3* то на кожному раунді дані пропускаються через них і нагромаджуються в регістрах *sbox0-sbox3*. У регістрі *sbox0* буде формуватися результат проходження регістру *in0* через таблицю *SBox0* (*sbox0 = SBox0(in0)*), *sbox1* – через таблицю *SBox1* і т. д., а наприкінці відповідні байти з регістрів *sbox0-sbox3* комбінуються для отримання результату (рис. 5).

Для випадку AVX-256 та AVX-512 потрібно продублювати кожен 16-байтний рядок вихідної таблиці заміни два або чотири рази, оскільки заміна виконується в межах *lane* по 16 байт.

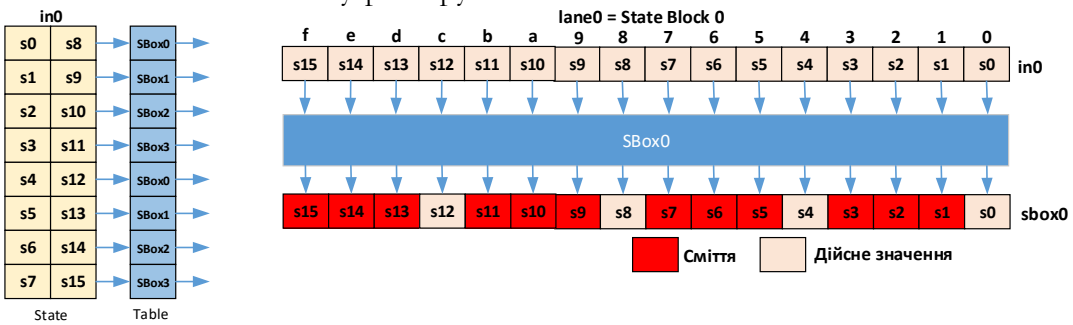

Рис. 5. Розташування даних у регістрі *sbox0* після проходження таблиці *SBox0*

Порівнюючи з AVX-256 реалізація ***SubBytes*** для AVX-512 додатково спрощується завдяки збільшеним можливостям інструкцій *blend* та *shuffle* (розширення AVX-512F + AVX512BW), які дають змогу виконувати операції лише для потрібних байтів заданих маскою. Якщо в процесорі доступне розширення AVX-512VBMI то це дає змогу суттєво спростити ***SubBytes*** внаслідок використання інструкції перестановки *permute* (*_mm512 _maskz_permutex2var_epi8*). Інструкція *permute* дає змогу здійснювати перестановку в межах 128 байт (байти вибираються з двох 64-байтних *zmm*-регістрів), на відміну від інструкції *shuffle*, що здійснює перестановку лише в межах 16 байт. Отже, 256-байтна таблиця представлена в пам'яті 64-

байтними рядками *r[0]-r[3]* для заміни потребує лише дві команди *permute*. Спочатку з допомогою *_mm512_cmplt_epu8_mask(in0, step)* визначаються байти в *in0* менші за 128 (де регістр *step* побайтно містить значення 128). Далі відбувається їх заміна та інверсія маски *m0* (після чого вона буде вказувати на елементи зі значенням ≥128) і заміна решти байтів.

З урахуванням вищесказаного макроси нелінійної заміни матимуть такий вигляд:

AVX-256

```
#define SBOX_AVX256_STEP(in0, n) {          \
tmp0 =_mm256_cmpgt_epi8(in0,mask);          \
tmp0 =_mm256_or_si256(in0,tmp0);            \
tmp1 =_mm256_shuffle_epi8(sb0_row[n],tmp0);\
sbox0=_mm256_xor_si256(sbox0,tmp1);         \
tmp1 =_mm256_shuffle_epi8(sb1_row[n],tmp0);\
sbox1=_mm256_xor_si256(sbox1,tmp1);         \
tmp1 =_mm256_shuffle_epi8(sb2_row[n],tmp0);\
sbox2=_mm256_xor_si256(sbox2,tmp1);         \
tmp1 =_mm256_shuffle_epi8(sb3_row[n],tmp0);\
sbox3=_mm256_xor_si256(sbox3,tmp1);         \
in0  =_mm256_sub_epi8(in0,step);            \
} while (0)

#define SBOX_AVX256(in0, out0) {            \
tmp0 =_mm256_cmpgt_epi8(in0,mask);          \
tmp0 =_mm256_or_si256(in0,tmp0);            \
sbox0=_mm256_shuffle_epi8(sb0_row[0],tmp0); \
sbox1=_mm256_shuffle_epi8(sb1_row[0],tmp0); \
sbox2=_mm256_shuffle_epi8(sb2_row[0],tmp0); \
sbox3=_mm256_shuffle_epi8(sb3_row[0],tmp0); \
in0  =_mm256_sub_epi8(in0,step);            \
SBOX_AVX256_STEP(in0,1);                    \
...
SBOX_AVX256_STEP(in0, 14);                  \
tmp0 =_mm256_cmpgt_epi8(in0,mask);          \
tmp0 =_mm256_or_si256(in0,tmp0);            \
tmp1 =_mm256_shuffle_epi8(sb0_row[15],tmp0);\
out0 =_mm256_xor_si256(sbox0,tmp1);         \
tmp1 =_mm256_shuffle_epi8(sb1_row[15],tmp0);\
sbox1=_mm256_xor_si256(sbox1,tmp1);         \
tmp1 =_mm256_shuffle_epi8(sb2_row[15],tmp0);\
sbox2=_mm256_xor_si256(sbox2,tmp1);         \
tmp1 =_mm256_shuffle_epi8(sb3_row[15],tmp0);\
sbox3=_mm256_xor_si256(sbox3,tmp1);         \
mask =_mm256_slli_epi16(mask,12);           \
out0 =_mm256_blendv_epi8(out0,sbox1,mask);  \
mask =_mm256_slli_epi32(mask,8);            \
out0 =_mm256_blendv_epi8(out0,sbox2,mask);  \
mask =_mm256_slli_epi32(mask,8);            \
out0 =_mm256_blendv_epi8(out0,sbox3,mask);  \
mask =_mm256_set1_epi8(15);                 \
} while (0)

// mask = 0x0f0f0f0f...0f0f
// step = 0x10101010...1010
```

AVX-512 (F, BW)

```
#define SBOX_AVX512_STEP(in0, n) {                      \
m0   =_mm512_cmplt_epu8_mask(in0,step);                 \
sbox0=_mm512_mask_shuffle_epi8(sbox0,m0,sb0_row[n],in0);\
sbox1=_mm512_mask_shuffle_epi8(sbox1,m0,sb1_row[n],in0);\
sbox2=_mm512_mask_shuffle_epi8(sbox2,m0,sb2_row[n],in0);\
sbox3=_mm512_mask_shuffle_epi8(sbox3,m0,sb3_row[n],in0);\
in0  =_mm512_sub_epi8(in0,step);                        \
} while (0)

#define SBOX_AVX512(in0, out0, m0) {                    \
m0   =_mm512_cmplt_epu8_mask(in0,step);                 \
sbox0=_mm512_maskz_shuffle_epi8(m0,sb0_row[0],in0);\
sbox1=_mm512_maskz_shuffle_epi8(m0,sb1_row[0],in0);\
sbox2=_mm512_maskz_shuffle_epi8(m0,sb2_row[0],in0);\
sbox3=_mm512_maskz_shuffle_epi8(m0,sb3_row[0],in0);\
in0 =_mm512_sub_epi8(in0,step);                         \
SBOX_AVX512_STEP(in0, 1);                               \
...
SBOX_AVX512_STEP(in0, 15);                              \
out0=_mm512_mask_blend_epi8(0x1111111111111111,out0,sbox0);\
out0=_mm512_mask_blend_epi8(0x2222222222222222,out0,sbox1);\
out0=_mm512_mask_blend_epi8(0x4444444444444444,out0,sbox2);\
out0=_mm512_mask_blend_epi8(0x8888888888888888,out0,sbox3);\
} while(0)
```

AVX-512 (F, BW, VBMI)

```
#define SBOX_AVX512(in0, out0, m0) {                    \
m0   =_mm512_cmplt_epu8_mask(in0,step);                 \
sbox0=_mm512_maskz_permutex2var_epi8(m0,r0[0],in0,r0[1]);\
sbox1=_mm512_maskz_permutex2var_epi8(m0,r1[0],in0,r1[1]);\
sbox2=_mm512_maskz_permutex2var_epi8(m0,r2[0],in0,r2[1]);\
sbox3=_mm512_maskz_permutex2var_epi8(m0,r3[0],in0,r3[1]);\
m0   =_knot_mask64(m0);                                 \
tmp0 =_mm512_maskz_permutex2var_epi8(m0,r0[2],in0,r0[3]);\
out0 =_mm512_or_si512(sbox0,tmp0);                      \
tmp0 =_mm512_maskz_permutex2var_epi8(m0,r1[2],in0,r1[3]);\
sbox1=_mm512_or_si512(sbox1,tmp0);                      \
tmp0 =_mm512_maskz_permutex2var_epi8(m0,r2[2],in0,r2[3]);\
sbox2=_mm512_or_si512(sbox2,tmp0);                      \
tmp0 =_mm512_maskz_permutex2var_epi8(m0,r3[2],in0,r3[3]);\
sbox3=_mm512_or_si512(sbox3,tmp0);                      \
out0 =_mm512_mask_blend_epi8(0x2222222222222222,out0,sbox1);\
out0 =_mm512_mask_blend_epi8(0x4444444444444444,out0,sbox2);\
out0 =_mm512_mask_blend_epi8(0x8888888888888888,out0,sbox3);\
} while(0)
```

У мультиблокових реалізаціях у яких задіяні 2/4/… векторних регістрів *in* для зберігання даних є можливість оптимізувати перетворення ***SubBytes*** і зменшити кількість операцій. Пояснимо це на прикладі молодших чотирьох байтів матриць стану *s00-s03*, де у виразі *smn m* – номер регістру, *n* – номер байту. У неоптимізованому випадку двох *ymm*-регістрів *in0-in1* процедура ***SubBytes*** для байтів *s00-s03* здійснюється для кожного регістру окремо, як показано на рис. 6. Водночас видно, що значна кількість байтів у регістрах формування результату *sbox0-sbox3* не задіяна й містить сміття. Тому якщо обробляються два регістри *in0-in1* то їхні дані доцільно перед процедурою ***SubBytes*** згрупувати, виконати нелінійну заміну з меншою кількістю операції та знову перегрупувати результат.

Схема перегрупування даних та виконання операції ***SubBytes*** для випадку двох регістрів показана на рис. 7. За такої умови, якщо не враховувати невеликі накладні витрати на початкове й кінцеве перегрупування даних у регістрах, кількість операцій зменшується вдвічі.

Ще більшої оптимізації можна досягнути у випадку 4-х регістрів *in0-in3,* що забезпечує майже чотирикратне зменшення числа операцій незважаючи на відчутно складнішу процедуру перегрупування. Схема перегрупування та виконання операції ***SubBytes*** для випадку чотирьох регістрів показана на рис. 8.

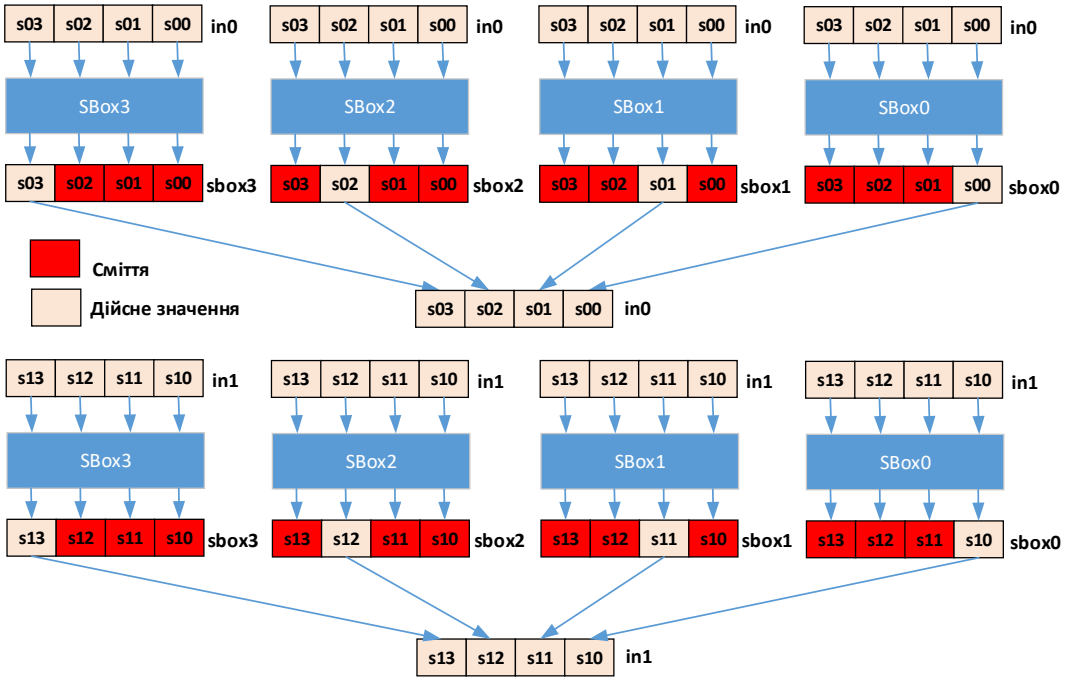

Рис. 6. Неоптимізоване виконання операцій ***SubBytes*** для двох векторних регістрів *in0-in1*

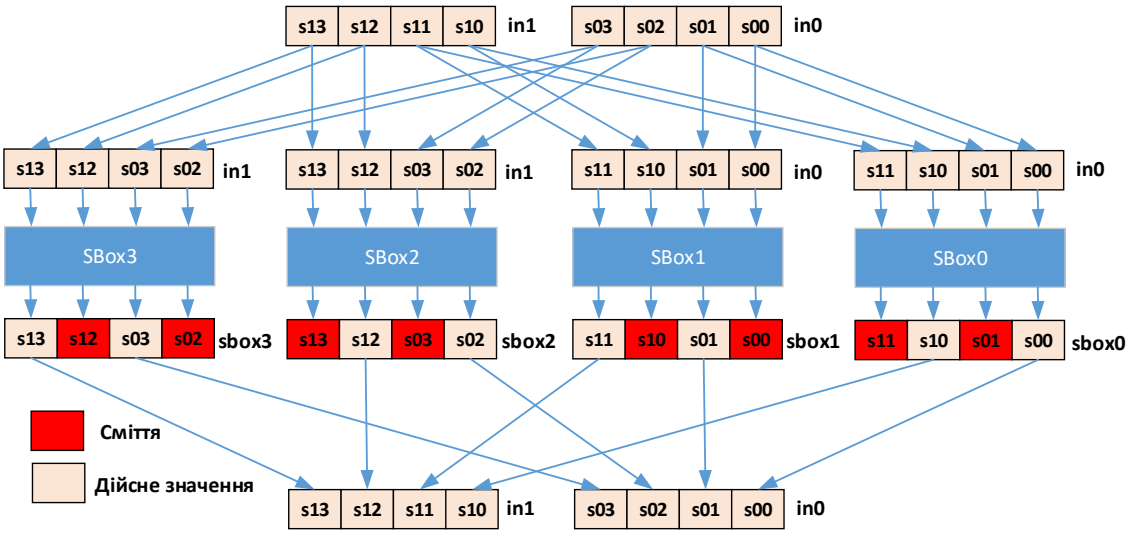

Рис. 7. Оптимізоване виконання операції ***SubBytes*** для двох векторних регістрів *in0-in1*

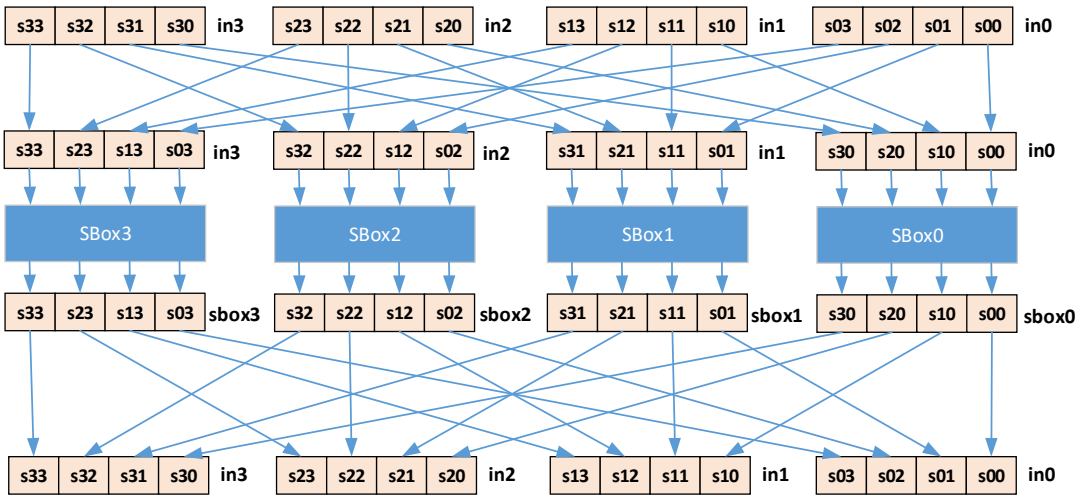

Рис. 8. Оптимізоване виконання операції ***SubBytes*** для чотирьох векторних регістрів *in0-in3*

Операції ***ShiftRows*** та ***InvShiftRows*** реалізуються однією інструкцією, яка здійснює перестановку 32-бітних слів *_mm256_shuffle_epi32(state, 0x6c)*.

Операції ***MixColumns*** та ***InvMixColumns*** здійснюють множення матриці стану *State* (*s0-s15*) на матрицю коефіцієнтів за модулем твірного поліному $poly = x^8 + x^4 + x^3 + x^2 + 1$ (0x11d). Розглянемо виконання ***MixColumns*** на прикладі одного стовпця матриці стану *s0-s7*.

$$\begin{bmatrix} 0x01 & 0x01 & 0x05 & 0x01 & 0x08 & 0x06 & 0x07 & 0x04 \\ 0x04 & 0x01 & 0x01 & 0x05 & 0x01 & 0x08 & 0x06 & 0x07 \\ 0x07 & 0x04 & 0x01 & 0x01 & 0x05 & 0x01 & 0x08 & 0x06 \\ 0x06 & 0x07 & 0x04 & 0x01 & 0x01 & 0x05 & 0x01 & 0x08 \\ 0x08 & 0x06 & 0x07 & 0x04 & 0x01 & 0x01 & 0x05 & 0x01 \\ 0x01 & 0x08 & 0x06 & 0x07 & 0x04 & 0x01 & 0x01 & 0x05 \\ 0x05 & 0x01 & 0x08 & 0x06 & 0x07 & 0x04 & 0x01 & 0x01 \\ 0x01 & 0x05 & 0x01 & 0x08 & 0x06 & 0x07 & 0x04 & 0x01 \end{bmatrix} \otimes \begin{bmatrix} s0 \\ s1 \\ s2 \\ s3 \\ s4 \\ s5 \\ s6 \\ s7 \end{bmatrix}.$$

Під час модульного множення враховується, що коефіцієнти матриці можна представити через степені 2: $4 = 2^2$, $5 = 2^2 + 2^0$, $6 = 2^2 + 2^1$, $7 = 2^2 + 2^1 + 2^0$, $8 = 2^3$, тоді:

$$1 \otimes \begin{bmatrix} s0 \oplus s1 \oplus s2 \oplus s3 \oplus s6 \\ s1 \oplus s2 \oplus s3 \oplus s4 \oplus s7 \\ s2 \oplus s3 \oplus s4 \oplus s5 \oplus s0 \\ s3 \oplus s4 \oplus s5 \oplus s6 \oplus s1 \\ s4 \oplus s5 \oplus s6 \oplus s7 \oplus s2 \\ s5 \oplus s6 \oplus s7 \oplus s0 \oplus s3 \\ s6 \oplus s7 \oplus s0 \oplus s1 \oplus s4 \\ s7 \oplus s0 \oplus s1 \oplus s2 \oplus s5 \end{bmatrix} \oplus 2 \otimes \begin{bmatrix} s5 \oplus s6 \\ s6 \oplus s7 \\ s7 \oplus s0 \\ s0 \oplus s1 \\ s1 \oplus s2 \\ s2 \oplus s3 \\ s3 \oplus s4 \\ s4 \oplus s5 \end{bmatrix} \oplus 4 \otimes \begin{bmatrix} s2 \oplus s5 \oplus s6 \oplus s7 \\ s3 \oplus s6 \oplus s7 \oplus s0 \\ s4 \oplus s7 \oplus s0 \oplus s1 \\ s5 \oplus s0 \oplus s1 \oplus s2 \\ s6 \oplus s1 \oplus s2 \oplus s3 \\ s7 \oplus s2 \oplus s3 \oplus s4 \\ s0 \oplus s3 \oplus s4 \oplus s5 \\ s1 \oplus s4 \oplus s5 \oplus s6 \end{bmatrix} \oplus 8 \otimes \begin{bmatrix} s4 \\ s5 \\ s6 \\ s7 \\ s0 \\ s1 \\ s2 \\ s3 \end{bmatrix}.$$

Отже, потрібно здійснити три модульні множення, байтові перестановки intrinsic-функцією *_mm256_shuffle_epi8* та сумування за модулем два функцією *_mm256_xor_si256*.

Множення числа x на 2 в полі $GF(2^8)$ за модулем $poly = 0x11d$ еквівалентно зсуву числа x вліво й додавання за модулем два з незвідним поліномом *poly*, якщо старший біт x був рівний 1. Відповідний фрагмент коду здійснює дану операцію для вектора *in0,* позначаючи в масці *tmp0* байти зі старшим бітом рівним 1 значенням 0xff, і далі накладаючи цю маску на регістр *poly*, що заповнений байтами 0x1d. Операцією *xor* здійснюється модульна редукція результату:

```
tmp0 = _mm256_cmpgt_epi8(zero, in0);   // zero = 0x0000...00
in0  = _mm256_add_epi8(in0,  in0);
tmp0 = _mm256_and_si256(tmp0, poly); // poly = 0x1d1d...1d
in0  = _mm256_xor_si256(in0,  tmp0);
```

Для вхідного вектора s обчислюються значення: $2s$, $4s$, $8s$ та потім комбінуються відповідним чином. З урахуванням вищесказаного макрос нелінійної заміни матиме такий вигляд:

```
// pm[0] = 0x0007060504030201, 0x080f0e0d0c0b0a09, 0x0007060504030201, 0x080f0e0d0c0b0a09
// pm[1] = 0x0100070605040302, 0x09080f0e0d0c0b0a, 0x0100070605040302, 0x09080f0e0d0c0b0a
// pm[2] = 0x0201000706050403, 0x0a09080f0e0d0c0b, 0x0201000706050403, 0x0a09080f0e0d0c0b
// pm[3] = 0x0302010007060504, 0x0b0a09080f0e0d0c, 0x0302010007060504, 0x0b0a09080f0e0d0c
// pm[4] = 0x0403020100070605, 0x0c0b0a09080f0e0d, 0x0403020100070605, 0x0c0b0a09080f0e0d
// pm[5] = 0x0504030201000706, 0x0d0c0b0a09080f0e, 0x0504030201000706, 0x0d0c0b0a09080f0e
// pm[6] = 0x0605040302010007, 0x0e0d0c0b0a09080f, 0x0605040302010007, 0x0e0d0c0b0a09080f
// poly  = 0x1d1d1d1d1d1d1d1d, 0x1d1d1d1d1d1d1d1d, 0x1d1d1d1d1d1d1d1d, 0x1d1d1d1d1d1d1d1d
// zero  = 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000
#define AVX256_MIX_COLUMN(in0, out0){\
tmp1 =_mm256_shuffle_epi8(in0,pm[0]);\
out0 =_mm256_xor_si256(in0,tmp1);    \
tmp1 =_mm256_shuffle_epi8(in0,pm[1]);\
out0 =_mm256_xor_si256(out0,tmp1);   \
tmp1 =_mm256_shuffle_epi8(in0,pm[2]);\
out0 =_mm256_xor_si256(out0,tmp1);   \
in0  =_mm256_shuffle_epi8(in0,pm[5]);\
out0 =_mm256_xor_si256(out0,in0);    \
tmp0 =_mm256_cmpgt_epi8(zero,in0);   \
in0  =_mm256_add_epi8(in0,in0);      \
tmp0 =_mm256_and_si256(tmp0,poly);   \
in0  =_mm256_xor_si256(in0,tmp0);    \
out0 =_mm256_xor_si256(out0,in0);    \
in0  =_mm256_shuffle_epi8(in0,pm[6]);\
out0 =_mm256_xor_si256(out0,in0);    \
tmp0 =_mm256_cmpgt_epi8(zero,in0);   \
in0  =_mm256_add_epi8(in0,in0);      \
tmp0 =_mm256_and_si256(tmp0,poly);   \
in0  =_mm256_xor_si256(in0,tmp0);    \
out0 =_mm256_xor_si256(out0,in0);    \
in0  =_mm256_shuffle_epi8(in0,pm[0]);\
out0 =_mm256_xor_si256(out0,in0);    \
in0  =_mm256_shuffle_epi8(in0,pm[2]);\
out0 =_mm256_xor_si256(out0,in0);    \
tmp0 =_mm256_cmpgt_epi8(zero,in0);   \
in0  =_mm256_add_epi8(in0,in0);      \
tmp0 =_mm256_and_si256(tmp0,poly);   \
in0  =_mm256_xor_si256(in0,tmp0);    \
tmp0 =_mm256_shuffle_epi8(in0,pm[1]);\
out0 =_mm256_xor_si256(out0,tmp0);   \
} while (0)
```

Під час розшифрування для операції ***InvMixColumns*** потрібно обчислювати не тільки 2*s*, 4*s*, 8*s*, але і 16*s*, 32*s*, 64*s*, 128*s*. Отже, операція розшифрування завжди буде повільніша за операцію зашифрування у програмній реалізації.

У таблиці 7 представлено результати вимірювання швидкодії з різною кількістю блоків, де зеленим кольором відзначено найкращі результати для кожної реалізації.

Для AVX-512 результати подано лише для розширень AVX-512F + AVX512BW. На жаль, у нас не було технічної можливості оцінити швидкодію в разі використання розширення AVX-512 VBMI для ***SubBytes***, оскільки відповідні процесори тільки з'являються на ринку.

Таблиця 7

Програмні реалізації шифру «Калина» (Software-based, Constant Time)

К-ть блоків	SSE-128				AVX-256				AVX-512 (F, BW)			
	GCC, cpb		MSVC, cpb		GCC, cpb		MSVC, cpb		GCC, cpb		MSVC, cpb	
	ENC	DEC	ENC	DEC	ENC	DEC	ENC	DEC	ENC	DEC	ENC	DEC
1-way	84,25	107,13	48,75	57,75	38,13	45,38	35,50	43,88	52,38	74,00	57,25	78,63
2-way	50,31	77,88	31,75	44,44	34,75	42,69	24,13	30,94	28,31	43,38	30,00	47,50
4-way	37,94	61,47	24,00	35,88	21,59	29,34	15,09	22,50	18,13	24,69	19,72	25,13
8-way	-	-	-	-	16,42	25,91	10,63	17,19	11,08	16,64	11,73	18,25
12-way	-	-	-	-	-	-	-	-	10,14	17,22	10,75	17,86
16-way	-	-	-	-	-	-	-	-	8,32	15,14	8,78	15,55

Перехід від табличної до програмної реалізації у випадку AVX-256 помірно зменшує швидкодію зашифрування з 8,42 до 10,63 тактів/байт, проте відчутно розшифрування з 9,73 до 17,19. Така ж ситуація спостерігається і для AVX-512: швидкодія зашифрування зменшується з 5,47 до 8,32, а розшифрування із 6,84 до 15,14 тактів/байт. Загалом стійка до кеш-атак програмна реалізація AVX-512 може конкурувати за швидкодією зашифрування з традиційними табличними GPR-реалізаціями (8,32 проти 7,69 тактів/байт), проте є майже вдвічі повільніша під час розшифрування (15,14 проти 8,81 тактів/байт).

10. Висновки

У роботі представлено як вразливі, так і стійкі до кеш-атак реалізації шифрів ГОСТ 28147-89 та «Калини» на базі векторних інструкцій SSE-128, AVX-256 і AVX-512 та здійснено оцінку їхньої швидкодії для однакової програмно-апаратної платформи.

Загалом за умови використання мультиблокового векторного шифрування максимально досяжна швидкодія «Калини» поступається ГОСТ 28147-89. Отже конкурсна вимога, що швидкодія нового криптоалгоритму не повинна бути меншою за швидкодію ГОСТ 28147-89 у цій частині наразі не досягнута.

Використання AVX-512 інструкцій дає змогу збільшити швидкодію «Калини» до 5,47/6,84 тактів/байт порівнюючи з табличними підходами на регістрах загального призначення представлених в [14] (10,41 тактів/байт) чи нашій роботі (8,81/7,69 тактів/байт). Поява на ринку процесорів із підтримкою AVX-512VBMI дозволить ще більше підвищити швидкодію «Калини».

Використання технології AVX-256 для імплементації шифру «Калини» виправдане за умови необхідності гарантування стійкості до кеш-атак. За цих умов вдається забезпечити прийнятні показники швидкодії на рівні 10,63/17,19 тактів/байт. Це особливо актуально для більшості процесорів, які не мають підтримки AVX-512, тим більше що використання AVX-512 у цьому випадку не дає відчутного приросту швидкодії (8,32/15,14 тактів/байт).

Коли ж вимоги до безпеки реалізації «Калини» не ставляться і допустиме використання передобчислених таблиць технологія AVX-256 не має переваг порівняно з традиційним використанням GPR-регістрів (8,42/9,73 проти 7,69/8,81 тактів/байт відповідно).

Зазначимо високу швидкодію ГОСТ 28147-89 досягнуту завдяки застосуванню AVX-256 та AVX-512 інструкцій (3,39 та 1,88 тактів/байт відповідно), що пояснюється простотою векторизації раундового перетворення з використанням 4-бітних вузлів заміни.

ЛІТЕРАТУРА

[1]. ДСТУ 7624:2014. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення. К.: Мінекономрозвитку України, 2015. [Електронний ресурс]. Режим доступу: http://ukrndnc.org.ua/ downloads/ new_view/?i=dstu-7624-2014&pz=% C4%D1%D2%D3+7624%3A2014.

[2]. В. Ковтун, А. Охрименко, "Особенности построения кроссплатформенной библиотеки криптографических примитивов "Шифр+" v2". [Електронний ресурс]. Режим доступу: https://cipher. com.ua/media/%D0%9F%D1%80%D0%BE%D0 %B4%D1%83%D0%BA%D1%82%D1%8B/%D0 %A8%D0%B8%D1%84%D1%80%2Bv2.1/Present ation_Cipher_Plus.pdf.

[3]. А. Кролевецкий, "Производительность ГОСТ-шифрования на x86- и GPU-процессорах", *Storage News*, № 4 (60), С. 28-29, 2014.

[4]. А. Кролевецкий, "Эффективная реализация алгоритма ГОСТ 28147-89 с помощью технологии GPGPU", *Материалы XVI международной конференции "РусКрипто'2014".*

[5]. О. Кузнецов, Р. Олійников, Ю. Горбенко, А. Пушкарьов, О. Дирда, І. Горбенко, "Обґрунтування вимог, побудування та аналіз перспективних симетричних криптоперетворень на основі блочних шифрів", *Вісн. "Комп'ютерні системи та мережі" Нац. ун-ту "Львів. політехніка",* № 806, С. 124-140, 2014.

[6]. Р. Олійников, І. Горбенко, О. Казимиров, В. Руженцев, Ю. Горбенко, "Принципи побудови і основні властивості нового національного стандарту блокового шифрування України", *Захист інформації*, 17, № 2, С. 142-157, 2015.

[7]. Системы обработки информации. Защита криптографическая. Алгоритмы криптографического преобразования: ДСТУ ГОСТ 28147:2009. – К.: Держспоживстандарт України, 2008. [Електронний ресурс]. Режим доступу: http://ukrndnc.org. ua/downloads/new_view/?i=dstugost28147-2009& pz=%C4%D1%D2%D3+%C3%CE%D1%D2+28 147%3A2009.

[8]. Л. Тычина, *Способ шифрования данных для вычислительных платформ с SIMD-архитектурой,* Евразийский патент № 021803, 2015.

[9]. Хардкорный путь к производительности. Достигаем феноменальной скорости на примере шифрования ГОСТ 28147-89, Хакер, № 08 (163), С. 90-94, 2012.

[10]. Cppcrypto library. Encryption performance. [Електронний ресурс]. Режим доступу: http:// cppcrypto. sourceforge.net/.

[11]. Intel Intrinsics Guide. [Електронний ресурс]. Режим доступу: https://software.intel.com/sites/ landingpage/IntrinsicsGuide/.

[12]. D. Gruss, *Software-based Microarchitectural Attacks. PhD Thesis*, Graz University of Technology, June 2017.

[13]. D. Kusswurm, *Modern x86 Assembly Language Programming 32-bit 64-bit SSE and AVX*, Apress, 2014, 667 p.

[14]. R. Oliynykov, O. Kazymyrov, O. Kachko, R. Mordvinov, et al. *Source code for performance estimation of 64-bit optimized implementation of the block ciphers Kalyna, AES, GOST, BelT, Kuznyechik.* [Електронний ресурс]. Режим доступу: https:// github.com/Roman-Oliynykov/ ciphers- speed/.

[15]. *Using the RDTSC Instruction for Performance Monitoring*, Intel, Application Note, 1998, 12 p.

ЭФФЕКТИВНАЯ ИМПЛЕМЕНТАЦИЯ И СРАВНЕНИЕ БЫСТРОДЕЙСТВИЯ ШИФРОВ «КАЛИНА» И ГОСТ 28147-89 ПРИ ИСПОЛЬЗОВАНИИ ВЕКТОРНЫХ РАСШИРЕНИЙ SSE, AVX И AVX-512

Очень важным свойством блочных шифров является обеспечение высокой производительности для широкого класса микропроцессорных архитектур и, прежде всего, для доминирующих x86-64 платформ. Недостаточное быстродействие ГОСТ 28147:2009 на современных вычислительных архитектурах общего назначения стало одной из причин проведения национального криптоконкурса по избранию нового блочного шифра, в котором победил алгоритм «Калина», быстродействие которого, по условиям конкурса, должно было быть не меньше, чем быстродействие действующего государственного стандарта шифрования. Чтобы достичь высокого быстродействия существующие реализации шифра «Калина» используют табличный одноблочный подход, который не лишен ряда недостатков: не используются возможности современных процессоров по распараллеливанию выполнения кода, векторизации обработки данных, уязвим к кэш-атакам. В работе предложены основные подходы к разработке мультиблочных векторных реализаций шифров «Калина» и ГОСТ 28147-89, в том числе устойчивых к кэш-атакам с использованием SIMD-инструкций SSE, AVX/AVX2, AVX-512. Особое внимание уделено выполнению операции нелинейной замены, которая определяет быстродействие реализации в целом. Проведены экспериментальные исследования, которые доказали эффективность предложенных подходов по увеличению быстродействия и позволили определить целесообразность применения соответствующих векторных расширений в том или ином случае. Установлено, что по максимально достижимому быстродействию векторные реализации ГОСТ 28147-89 ощутимо опережают шифр «Калина». Использование предложенных подходов позволяет повысить быстродействие

отечественных программных криптографических средств и безопасность их функционирования.
Ключевые слова: шифр «Калина», шифр ГОСТ 28147-89, векторные расширения системы команд SSE, AVX, AVX-512, x86-64 архитектура, эффективная имплементация криптоалгоритмов, измерения быстродействия.

EFFECTIVE IMPLEMENTATION AND PERFORMANCE COMPARISON OF «KALYNA» AND GOST 28147-89 CIPHERS WITCH THE USE OF VECTOR EXTENSIONS SSE, AVX AND AVX-512

A very important feature of block ciphers is the provision of high performance for a wide range of microprocessor architectures and, above all, for the dominant x86-64 platforms. Insufficient performance of DSTU GOST 28147:2009 on modern general-purpose computing architectures was one of the reasons for holding a national crypto competition for the choice of a new block cipher, in which the «Kalyna» algorithm won, whose performance, according to the conditions of the competition, was to be no less than the current one state encryption standard. In order to achieve high performance, existing implementations of the «Kalyna» cipher utilize a table-based, one-block approach that is devoid of drawbacks: it does not use the capabilities of modern processors to parallelize code execution, vectorization of data processing, and be vulnerable to cache attacks. The main approaches to the development of «Kalyna» and GOST 28147-89 ciphers multiblock vectorised implementations, including those resistant to cache attacks, using the SSE, AVX/AVX2, AVX-512 SIMD instructions are proposed. Particular importance is given to performing a non-linear substitution operation, which determines the speed of implementation in general. Experimental studies have been conducted to prove the effectiveness of the proposed approaches to increasing performance and to determine the feasibility of using the appropriate vector extensions in one case or another. It is established that according to the most achievable speed vectorized implementations of GOST 28147-89 significantly exceed the cipher «Kalyna». The use of the proposed approaches allows to increase the speed of domestic cryptographic software tools and their security.
Keywords: Kalina cipher, GOST 28147-89 cipher, vector extensions of instruction set architecture SSE, AVX, AVX-512, x86-64 architecture, effective implementation of crypto algorithms, performance measurement.

Совин Ярослав Романович, кандидат технічних наук, доцент, доцент кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: sovynjarosl@gmail.com.
Orcid ID: 0000-0002-5023-8442.
Совын Ярослав Романович, кандидат технических наук, доцент, доцент кафедры защиты информации Национального университета «Львовская политехника».
Sovyn Yaroslav, Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Information Security, Lviv Polytechnic National University.

Хома Володимир Васильович, доктор технічних наук, професор, професор кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: volodymyr.v.khoma@lpnu.ua.
Orcid ID: 0000-0001-9391-6525.
Хома Владимир Васильевич, доктор технических наук, профессор, профессор кафедры защиты информации Национального университета «Львовская политехника».
Khoma Volodymyr, Dr. Sci. Tech., Professor, Professor at the Department of Information Security, Lviv Polytechnic National University.

Наконечний Юрій Маркіянович, кандидат технічних наук, доцент, доцент кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: nak15@ukr.net.
Orcid ID: 0000-0002-6046-6190.
Наконечный Юрий Маркиянович, кандидат технических наук, доцент, доцент кафедры защиты информации Национального университета «Львовская политехника».
Nakonechny Yurii, Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Information Security, Lviv Polytechnic National University.

Стахів Марта Юріївна, кандидат технічних наук, доцент кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: martast75@gmail.com.
Orcid ID: 0000-0002-4094-2081.
Стахив Марта Юрьевна, кандидат технических наук, доцент кафедры защиты информации Национального университета «Львовская политехника».
Stakhiv Marta, Candidate of Technical Sciences, Associate Professor at the Department of Information Security, Lviv Polytechnic National University.